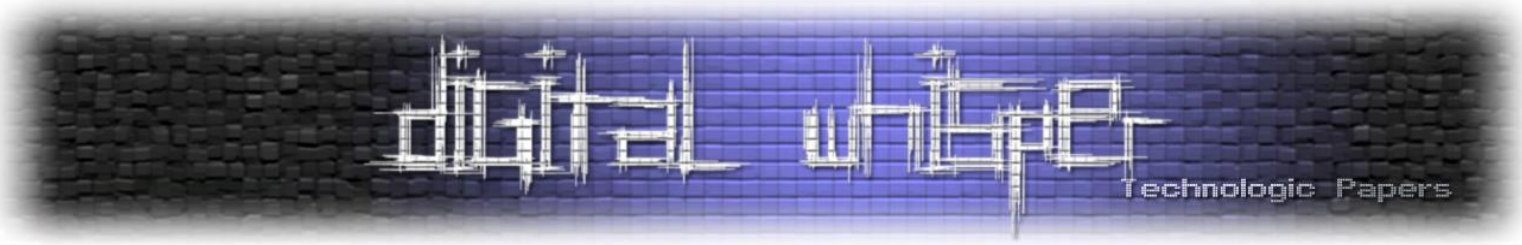


Digital Whisper

גליון 108, יולי 2019

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	יהלי הכט, מתן ונגוש, רון דלאל, אירי מור, עידו פרידמן ועדי מליאנקר (Magen_Gide0n)

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper ו/או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

פניות, תגובות, כתבות וכל הערה אחרת - נא לשלוח אל editor@digitalwhisper.co.il

דבר העורכים

ברוכים הבאים לגליון ה-108 של DigitalWhisper!

החודש, לפני 15 שנים בדיוק, התגלתה התולעת Cabir: התולעת הראשונה לסמארטפונים. בשנת 2004, תחום הסמארטפונים עוד היה בחיתוליו. ובכלל, ביחס למכשירי הסלולאר הקיימים היום, קצת מצחיק להתייחס אל אותם מכשירים כ"חכמים", אך בכל זאת - הם לא היו לגמרי טפשים ☺

Cabir פותחה כ-PoC "במעבדה" - ע"י כותב וירוסים ברזילאי המכונה בשם Vallez, שהיה חבר באחת מקבוצות כותבי וירוסים המפורסמת ביותר בעולם: 29a (666 ב-Hex). ובסיום פיתוחה - שלח אותה לחברת Kaspersky שקפצה על ההזדמנות וכינסה מסיבת עיתונאים ובה הציגה את התולעת.

מערכת ההפעלה הנפוצה ביותר לסמארטפונים באותן השנים, לא הייתה Android, אלא מערכת הפעלה בשם Symbian OS, מערכת הפעלה המבוססת על מערכת ההפעלה EPOC32 שפותחה ע"י חברה בריטית בשם Psion (ונקתנה ב-2012 ע"י חברת Motorola). מערכת ההפעלה הזו הפעילה את הסמארטפונים של היצרניות הגדולות באותה התקופה: Sony Ericsson, Samsung ,Motorola ו-Nokia ורצה על מעבדי ARM ו-x86 באופן לא רשמי.

גם התולעת Cabir לא הייתה חכמה מאוד, היא לא ניצלה חולשות או פרצות במערכת ההפעלה, היא פשוט ניצלה את כרטיס ה-BlueTooth של הסמארטפון עליו היא רצה כדי לסרוק אחר מכשירים נוספים המאפשרים גילוי באמצעות BlueTooth ושלחה את עצמה כקובץ להתקנה על-גבי ממשק זה. הנתקף (קרי: כל מכשיר שהיה נמצא ברדיוס של פחות או יותר 10 מטר, והוגדר כ-Discoverable) היה מקבל הודעה על-כך שקיבל בקשה לקבלת קובץ ב-BlueTooth ממכשיר קרוב, היה נדרש לאשר את הקבלה, ובגמר סיום קבלת הקובץ היה צריך גם לאשר את הרצתו. מי שהיה מספיק סקרן/אדיוט כדי להגיע לשלב זה - זכה בעותק של התולעת ושמכשירו כעת יתחיל לסרוק אחר מכשירים נוספים בסביבה.

Vallez אומנם לא "חימש" את התולעת אלא ישר שלח אותה לחברת ה-AV, אך קוד המקור שלה פורסם באינטרנט וארגוני פשיעה אחרים, שידרגו אותה, קמפלו ושחררו אותה לחופשי, אחד השדרוגים למשל, אפשר לה להתפשט גם על גבי הודעות MMS. אחד הסיפורים מספר על חוקר של חברת F-Secure שחטף את התולעת בעת ששהה באיצדיון בפינלנד שבו הגיע צופה נגוע בתולעת.

מאז ועד היום עברו לא מעט גלים אלקטרומגנטיים בתדרי UHF, ועולם הסמארטפונים השתדרג פלאים. איתו כמובן השתדרגו יכולותיהם של הוירוסים השונים לפלטפורמה זו, המוחות הקרימינלים מאחורי כל אותן הנוזקות פועל בצורה יעילה הרבה יותר מהדרך שבה החברה שלנו מצליחה להגביר את המודעות לכל אותן הכולירות שקיימות באינטרנט. ואם Cabir הצליחה להדביק מספר די מצומצם של משתמשים,

וירוסי סמארטפונים כיום מצליחים להגיע לכמה וכמה מליונים של הדבקות (כדוגמאת HummingBad שהתגלה ע"י חברת Checkpoint ב-2016) ומכניס לבעליו מעל ל-$300,000 בחודש, או Shedun, שהצפי להדבקותיו ע"י ArsTechnica לשנת 2016 היה לא פחות מ-10,000,000 מכשירי אנדרואיד, ואף השתמש בחולשות PE כדי להשיג הרשאות root על המכשיר הנתקף).

15 שנים זה לא מעט, תחום ה-Mobile Security השתנה פלאים, ומתחום שכמעט ולא נחקר ע"י חברות האבטחה באותן השנים, הוא הפך לאחד מתתי-התחומים החמים ביותר של עולם הסייבר, כיום יש אינספור חברות המתמחות בכל אספקט ואספקט בשדה זה. וגם ארגוני הפשיעה הקיברנטיים לא טומנים את ידם בצלחת והבינו שניתן לנצל את העובדה שכל אחד ואחד מאיתנו מחזיק לפחות מכשיר אחד כזה בכיס. אך למרות שנראה שכלל התחום השתנה ללא היכר, נראה שאספקט אחד עדיין נשאר (כמעט) כשהיה בכל 15 השנים האחרונות, והוא: רמת המודעות לנושא בקרב הרוב המוחלט של המשתמשים. גם אחרון המשתמשים היום יודע להזהר מוירוסים כאשר מדובר בעולם ה-PC, אך בכל הזדמנות שיוצא לי לספר על כך שאין שום הבדל, מבחינת הטכנולוגיה בין המחשב לבין הסמארטפון, ולכן אין כמעט שום הבדל ברמת הסבירות של להדבק בוירוס בכל אחת מהפלטפורמות - אני נתקל בפרצופים ספקנים ובגבות מורמות.

מעניין מתי העניין יחלחל גם בתחום זה...

ואיך אפשר לעבור לתוכן הכל כך איכותי שנכתב בשבילכם החודש מבלי להגיד תודה למי שיצר אותו? תודה ל**יהלי הכט**, תודה ל**מתן ונגוש**, תודה ל**רון דלאל**, תודה ל**אירי מור**, תודה ל**עידו פרידמן** ותודה ל**עדי מליאנקר (Magen_Gide0n)**

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

דבר העורכים	2
תוכן עניינים	4
המתמטיקה שמאחורי פרצות במנגנון WEP	5
Re-Assembling the Web	25
תקני Token Ring ו-FDDI, ולמה כבר לא משתמשים בהם	34
iptables unleashed	46
דברי סיכום	55

המתמטיקה שמאחורי פרצות במנגנון WEP

מאת יהלי הכט ומתן ונגוש

הקדמה

ברב המקרים, כאשר נפרוש רשת, לא נרצה שלכל אחד תהיה אליה הגישה. כאשר החיבור הוא קווי, יהיה לנו קל להיות בטוחים שאף אחד לא יתחבר לרשת ללא רשות, משום שהוא צריך גישה פיזית לחומרה עצמה. אך כאשר מדובר ברשת Wireless העניין הופך להיות מעט מורכב יותר.

בדיוק לטובת כך הומצא אלגוריתם האבטחה WEP (קיצור של Wired Equivalent Privacy) שהחל להיכנס לשימוש בשנת 1999.

אלגוריתם זה הוצג כבר בעבר במגזין זה, מאמר מנגנון הצפנה WEP, אך המאמר שלנו מרחיב את הנושא, בדגש על החולשות והמתקפות נגד WEP, בניגוד למאמר שהוזכר, שבו מוצגות האפשרויות השונות לאבטחת הרשת, מאמר זה יתמקד בלוגיקה ובאלגוריתמים המתמטיים ולא רק במה שמתאפשר לבצע בעקבותם.

כאמור, WEP התיימר לאבטח את הרשת ברמה שקולה לחיבור קווי, הדורש גישה פיזית ומבטיח פרטיות. למרות זאת, כבר בשנת 2001 הצליחו לפצח את האבטחה של האלגוריתם, החוקרים Fluhrer, Mantin and Shamir הכינו את ההתקפה FMS שתפורט בהרחבה בהמשך. לכן פותח אלגוריתם ה- Wi-Fi WPA (Protected Access). למרות שלאלגוריתם זה אבטחה יותר חזקה, גם הוא חשוף לפריצות. אחריו הגיעו גם WPA2 ו-WPA3 שהוכרז ב-2018 וצפוי להחליף את WPA2 בעוד כמה שנים.

בהמשך, מסופר בקצרה על IEEE 802.11, תקן לרשתות מקומיות אלחוטיות, נתמקד באלגוריתם RC4, שבאמצעותו מוצפן המידע ב-WEP, בדרך פעולתו של האלגוריתם, פרצות אבטחה שהתגלו בו ודרכים לפרוץ רשת מבוססת WEP.

IEEE 802.11

IEEE 802.11 הוא סטנדרט לרשתות אלחוטיות מקומיות, שנמצא בשכבת הקו במודל חמשת השכבות. לסטנדרט זה יש מספר גרסאות, אך אנחנו נציג את IEEE 802.11b ששוחררה בשנת 1999 ותומכת במנגנון WEP, אך נמצאת בשימוש מועט. לקריאה נוספת על גרסאות נפוצות:

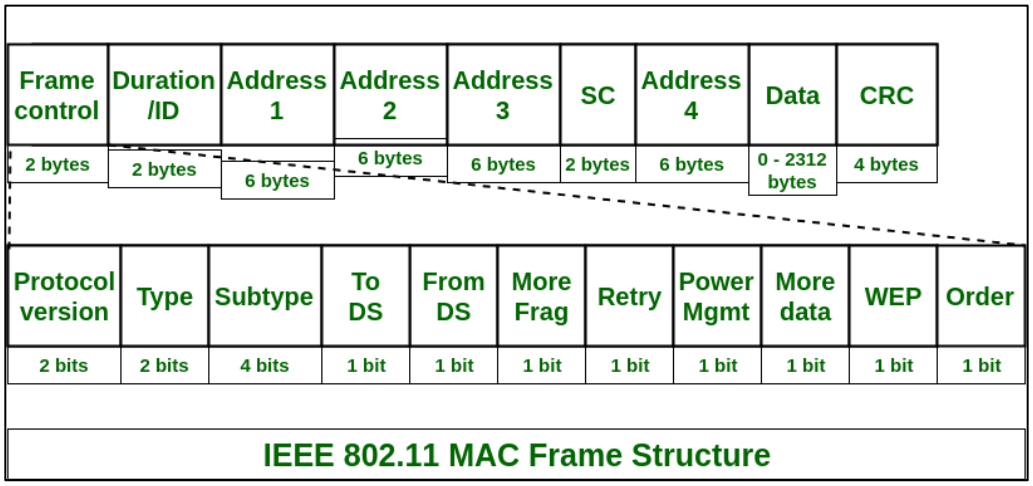
The Most Common Wi-Fi Standards and Types Explained

IEEE Standard	802.11a	802.11b	802.11g	802.11n	802.11ac	802.11ax
Year Released	1999	1999	2003	2009	2014	2019
Frequency	5Ghz	2.4GHz	2.4GHz	2.4Ghz & 5GHz	2.4Ghz & 5GHz	2.4Ghz & 5GHz
Maximum Data Rate	54Mbps	11Mbps	54Mbps	600Mbps	1.3Gbps	10-12Gbps

[מקור: The Most Common Wi-Fi Standards and Types Explained]

בתמונה מופיעים התדרים שבהם משתמש כל תקן, המהירות גלישה מרבית שמתאפשרת והשנה שבה כל תקן נכנס לשימוש

כדי להבין יותר טוב מה ה-IEEE 802.11 מספק לנו, נסתכל על שדות ב-Header שלו והמבנה של ה-Frame.



[מקור: IEEE 802.11 Mac Frame - GeeksforGeeks]

- Frame Control - מטרתו לקבוע את סוג ה-Frame ומידע דרוש. במאמר לא יפורטו הסעיפים ב-Frame Control, למעט הסעיף WEP ולכן מומלץ לקרוא עליהם בקישור למעלה. הסעיף WEP הוא בגודל סיבית אחת - 0 או 1. וקובע האם יש שימוש במנגנון.
- Duration - משפיע על סוג ה-Frame שמופיע בשדה הקודם (לקריאה נוספת)
- Source Address (SA) - כתובת ה-MAC של השולח הראשוני
- Destination Address (DA) - כתובת ה-MAC של היעד הסופי
- Transmitter Address (TA) - כתובת ה-MAC של השולח האחרון, מי ששידר את המידע.
- Receiver Address (RA0 - כתובת ה-MAC של מי שאליו משדרים את המידע
  - לא בהכרח היעד (DA) אלא הישות הבאה במסלול
- Sequence Control (SC) - כמו ה-Frame Control, השדה מחולק:

- Sequence number (12 bits)
- Fragment number (4 bits).
- השדה נועד לשמור על הסדר של הפקטות, כדי שהמידע לא יתבלגן בדרך (כי אם Frame אחד יגיע לפני Frame אחר שמופיע לפניו בסדר המידע, צריך לדאוג שהמידע לא ישורשר לפי זמן ההגעה אלא לפי הסדר הראשוני).

- Data - כולל את המידע שהועבר ב-Frame ואת ה-Headers של השכבות הגבוהות יותר.
- CRC - על ה-CRC Checksum בפרט ועל Checksum בכלל יפורט בהמשך, בהקשר של הצפנה במנגנון WEP.

אלגוריתם ה-RC4

אלגוריתם ה-RC4 הינו צופן זרם, כלומר צופן סימטרי המשמש להצפנת זרמי מידע באורכים משתנים, שפותח בשנת 1987 על ידי מומחה ההצפנה רונלד ריבסט במעבדות RSA, חברה שהקים עם עדי שמיר ולאונרד אדלמן שאיתם פיתח בשנת 1977 את אלגוריתם RSA (מומלץ לקרוא) שאת שמו החברה נושאת. משמעות השם RC4 היא Rivest Cipher 4, אך יש כאלו שמפרשים אותו בתור Ron's Code 4 או Ron's Cipher 4. האלגוריתם הינו אלגוריתם דטרמיניסטי, כלומר אלגוריתם המתנהג בצורה צפויה וניתנת לחיזוי. בהינתן קלט זהה, המכונה עליה רץ האלגוריתם תבצע את אותם צעדים והפלט הסופי יהיה אותו פלט. האלגוריתם אינו נתמך או נמצא בשימוש ברוב מערכות האינטרנט נכון להיום, אך בעבר שימש להצפנה בפרוטוקול SSL 4 (מאוחר יותר הפך אופציונלי ואז הוצא מכלל שימוש כפי שמתואר במסמך RFC 7465 שיצא בפברואר של שנת 2015), הצפנת ה-Seed (עליו נסביר בהמשך) ב-WEP, ועד לשנת 2015 עוד היה בשימוש בדפדפן Internet Explorer ובדפדפן Microsoft Edge.

האלגוריתם מזוהה עם הקלות שבה ניתן ליישם ולהטמיע אותו בתוכנה ובחומרה, אך גם בפרצות האבטחה הרבות שלו שהוציאו אותו משימוש.

לאלגוריתם יש שני חלקים:

1. (KSA (Key Scheduling Algorithm
2. (PRGA (Pseudo Random Generation Algorithm

נסתכל על שלבי הפעולה של האלגוריתם. נתחיל מ-KSA:

כדי שהאלגוריתם יוכל לעבוד, מספק המשתמש מפתח סודי אשר נשמר במערך K שאורכו n. כעת ניצור מערך S שאורכו בדרך כלל 256 אלמנטים, אך לצורך הדוגמה נשתמש במערך שאורכו 10 אלמנטים. את S ניצור באופן פשוט ביותר - באופן עולה. באינדקס 0 יהיה הערך 0, באינדקס 1 יהיה הערך 1 וכן הלאה, עד

לאינדקס 9 שערכו 9. כפי שצוין, המערך K הוא מפתח סודי אשר מסופק על ידי המשתמש. הבה נביט במערך S:

Value	0	1	2	3	4	5	6	7	8	9
Index	0	1	2	3	4	5	6	7	8	9

ובמערך K, שהוא מפתח פרטי שהתקבל כקלט מהמשתמש (לצורך הדוגמה נשתמש במפתח רנדומלי):

Value	5	4	7	1	0	6	3	2
Index	0	1	2	3	4	5	6	7

נשים לב כי n = 8 (נזכיר כי n אורכו של K).

נגדיר את הסמן j = 0. כעת נרוץ על המערך S איבר איבר באמצעות הסמן i = 0 שמצביע על האינדקס הנוכחי בכל איטרציה. נבצע את הפעולות הבאות:

נגדיר j = j + S[i] + K[i mod n] ונבצע מודולו עם אורכו של S. במקרה זה נקבל j = 5. נחליף בין ערכי S[i] ו-S[j]. נמשיך עד שנעבור על כל איברי S.

להלן הפונקציה כתובה בשפת python3, כך שתוכלו להריץ אותה על מחשבכם ולצפות בה פועלת (שימו לב כי במקרה זה אורכי S,K יתקבל כקלט מהמשתמש, וכי המערך K ייווצר באופן רנדומלי בידי התוכנה):

```
import random

s = int(input('enter the length of S: '))
k = int(input('enter the length of K: '))
S = list(range(0, s))
K = list(range(0, k))
random.shuffle(K)
print(f'K is {K}')

j = 0
for i in range(0, s):
    j = (j + S[i] + K[i % k]) % s
    S[i], S[j] = S[j], S[i]
print(f'shuffled S is {S}')
```

פלט לדוגמא:

```
enter the length of S: 10
enter the length of K: 8
K is [6, 0, 7, 1, 4, 5, 3, 2]
shuffled S is [3, 7, 0, 2, 8, 4, 9, 5, 6, 1]
```

כעת נסתכל על פעולתו של PGA:

לאחר שאלגוריתם KSA סיים את פעולתו, נשארנו עם מערך S מעורבב באופן פסודו-רנדומלי, כלומר רנדומלי כביכול (מכיוון שמחשב לא מסוגל לייצר רצפים רנדומליים אמיתיים, ניתן לקרוא על כך בלינק לעיל). נסתכל על המערך שיוצא כפלט כשמכניסים את K מהדוגמה הקודמת לאלגוריתם KSA. השורה האמצעית מייצגת את האינדקס, השורה העליונה מייצג את הערך, והשורה התחתונה מציגה את המיקום של שני סמנים i,j:

Value	1	6	9	2	7	3	5	4	8	0
Index	0	1	2	3	4	5	6	7	8	9
Pointer	i,j=0									

כפי שניתן לראות, המיקום ההתחלתי של i,j הוא 0. נתחיל את פעולת האלגוריתם על ידי הוספה של 1 לערך של i. כדי לא לחרוג מגבולות האינדקס של S נבצע מודולו בין i לבין האורך של S, במקרה זה אנחנו נישאר עם i = 1, אך אם נגיע למצב שבו i = 10, נשים לב שלא קיים אינדקס כזה, ולכן פעולת המודולו תחזיר אותנו חזרה לערך של i = 0:

Value	1	6	9	2	7	3	5	4	8	0
Index	0	1	2	3	4	5	6	7	8	9
Pointer	j=0	i=1								

כעת נוסיף לערך של j את S[i] (הערך של S באינדקס i). נבצע שוב מודולו עם האורך של S כדי לוודא שאיננו חורגים מגבולות המחרוזת. במקרה הזה S[i] = 6 ולכן j = 6 + 0 = 6 ואם נבצע מודולו ונשאר עם הערך j = 6:

Value	1	6	9	2	7	3	5	4	8	0
Index	0	1	2	3	4	5	6	7	8	9
Pointer		i=1					j=6			

נחליף בין הערכים של S[i], S[j]:

Value	1	5	9	2	7	3	6	4	8	0
Index	0	1	2	3	4	5	6	7	8	9
Pointer		i=1					j=6			

כעת נקבע t = S[i] + S[j]. נבצע מודולו עם אורך S, ואז נפנה לערך S[t] ונשמור אותו במערך הפלט. במקרה זה t = 5 + 6 = 11. אחרי פעולת מודולו נישאר עם t = 2. לכן האיבר הראשון בפלט שלנו יהיה:

S[t] = 9

כעת נחזור על פעולת האלגוריתם עד שאורך הפלט יהיה כאורך הטקסט שנרצה להצפין. הפלט מאלגוריתם RC4 נקרא Key Stream, וכפי שנסביר לעומק בהמשך, ההצפנה נעשית בפעולת XOR בינו לבין הטקסט שנרצה להצפין.

להלן קטע הקוד המלא בשפת python3 שמקבל את האורך של S, מקבל את K (המפתח הסודי) ומקבל את הטקסט שברצוננו להצפין, ומדפיס כפלט את הטקסט המוצפן והטקסט המפוענח (מעביר תהליך של פענוח מבלי להשתמש בקלט המקורי של המשתמש):

```
try:
    s = int(input('enter the length of S: '))
except:
    s = 255
K = [ord(c) for c in input('enter secret key: ')]
k = len(K)
text = input('enter text to encrypt: ')
stream_len = len(text)
text_dec = [ord(c) for c in text]
S = list(range(0, s))


# KSA:
j = 0
for i in range(0, s):
    j = (j + S[i] + K[i % k]) % s
    S[i], S[j] = S[j], S[i]

# PRGA:
key_stream = []
i, j = 0, 0
while len(key_stream) < stream_len:
    i = (i + 1) % s
    j = (j + S[i]) % s
    S[i], S[j] = S[j], S[i]
    key_stream.append(S[(S[i] + S[j]) % s])

# encoding:
text_enc = [chr(text_dec[i] ^ key_stream[i]) for i in range(stream_len)]
print(''.join(text_enc))

# decoding:
text_dec = [chr(ord(text_enc[i]) ^ key_stream[i]) for i in
range(stream_len)]
print(''.join(text_dec))
```

פלט לדוגמא:

```
enter the length of S:
enter secret key: Secret Key
enter text to encrypt: This is a secret message!
?"?i_8?xdy;??BDdX
This is a secret message!
```

שימו לב כי ההודעה המוצפנת מוצגת ב-UTF8.

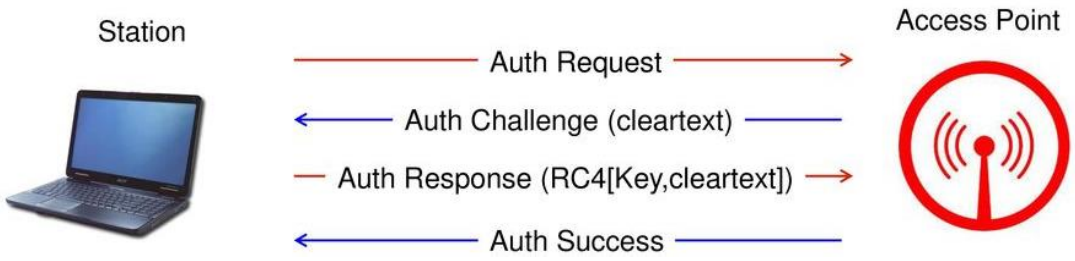
WEP - Wired Equivalent Privacy

אלגוריתם ה-WEP הוא אלגוריתם אבטחה של רשתות נתונים אלחוטיות הפועל מעל שתי השכבות הנמוכות במודל חמשת השכבות, השכבה הפיזית ושכבת הקו. מטרת האלגוריתם היא הצפנה השידור האלחוטי באמצעות סיסמא המוסכמת בין המחשב לנקודת הגישה, כך שתתקבל פרטיות שקולה לזאת של חיבור קווי.

ל-WEP יש מספר גרסאות, שההבדלים ביניהם הוא אורך ה-Seed (גרעין). ה-Seed מורכב מהמפתח הסודי של המשתמש ומחלק נוסף בשם Initialization Vector, או בקיצור IV. בעצם, ה-IV הוא מספר רנדומלי (או במקרה של מחשב, רק סודו-רנדומלי). במידה ולא היה לנו את ה-IV, התוצאה מה RC4 הייתה זהה למפתח, מה שיהפוך התקפות מסוימות (יפורט בהמשך) לפשוטות אפילו יותר.

אורך ה-Seed, בגרסה המקורית של האלגוריתם, הינו 64 סיביות, כאשר 24 סיביות מכילות את ה-IV ו-40 הסיביות האחרות מכילות את המפתח, שנקבע על ידי מנהל הרשת. בגרסאות החדשות יותר, ה-Seed באורך 128 או 256 סיביות. האלגוריתם עובד בשלבים הבאים:

ראשית, מתבצע אימות בין הנקודת גישה והלקוח (אופציונלי):



[מקור: Published by Ethan Montgomery]

- כדי לבצע אימות, הלקוח שולח ל-AP בקשת אימות (authentication).
- ה-AP מחזיר מספר אקראי באורך 128 סיביות.
- הלקוח מצפין את המידע שקיבל באמצעות המפתח המשותף ומחזיר את המידע המוצפן ל-AP.
- ה-AP מפענח את ההודעה ואם המידע ששלח זהה לפענוח הוא מאמת את הלקוח.

ה-AP שולח אתגר, כדי לוודא שיש ללקוח את המפתח האמיתי. למרות זאת, יש דרכים פשוטות לזייף אימות (יפורט בהמשך).

לאחר שהלקוח אומת, התקשורת בין ה-AP ללקוח מוצפנת ומתבצעת כך:

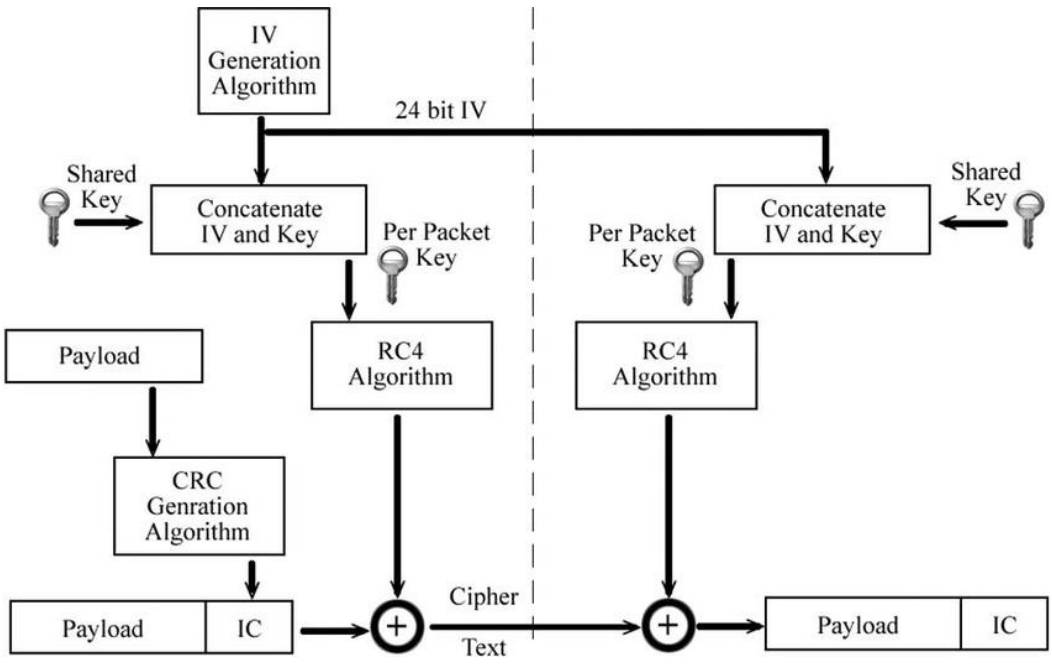
הצפנה:

- לוקחים את ה-Seed שאותו מצפינים באמצעות ה-RC4, ומקבלים רצף סודו-אקראי.
- המידע שישלח משורשר (צֵרוּף מחרוזת בתוכנת מחשב לסופה של מחרוזת אחרת- מילוג) עם ה-Checksum (CRC32) של המידע (מחזיר 32 סיביות שנקבעו על ידי הפלט ומשמש לאימות המידע, המטרה היא לזהות שגיאות בשידור ההודעה), ועובר XOR עם הרצף שהתקבל מה-RC4. זהו ה-Ciphertext, דהיינו המידע המוצפן.
- ל-IV משורשר ה-Ciphertext.

פענוח:

- מוציאים את ה-IV, מייצרים את ה-Seed ובאמצעות RC4 משיגים את ה-keystream (משום ש-RC4 הוא דטרמיניסטי, אותו ה-Seed יוביל לרצף זהה).
- מבצעים ל-Ciphertext את פעולת ה-XOR, מקבלים את ה-data וה-checksum.
- מבצעים על המידע שהתקבל checksum (CRC32) ומוודאים שה-checksum זהה לזה ששורשר ל-data כדי לאמת את המידע. ה-checksum לא משמש בתור חתימה דיגיטלית ולכן השימוש בו לא נועד לאבטחה.

בתרשים ניתן לראות את ההצפנה והפענוח של מנגנון ה-WEP:



[מקור: researchgate.net]

[1] הסימון דמוי גלגל מסמן את הפעולה XOR, שהיא סימטרית, ולכן ביצוע XOR פעמיים עם המחרוזת שהתקבלה מה-RC4 על המידע וה-checksum יוביל לאותו המידע וה-checksum.

[2] לאחר ה-XOR המידע משורשר ל-IV.

CRC

כשאנחנו שולחים מידע ממקום למקום, נרצה לוודא כי המידע, אשר נשלח כראוי, לא מגיע תקול. לכן אנחנו משתמשים בבדיקות Checksum. הבדיקה שנדבר עליה כעת נקראת CRC n-bit, והיא משתמשת בפעולת XOR ובפולינומים יוצרים כדי לייצר מספר שניתן לבדוק בעזרתו את תקינות המידע. ה-CRC n-bit מייצר מספר בן n ביטים, וככל ש-n גדול יותר כך רמת הדיוק תעלה. לכל CRC נקבע פולינום (הפולינום היוצר של CRC n-bit) ממעלה n שמקדמיו, כאשר הם משורשרים, יוצרים רצף בינארי שמשמש לחישוב ה-CRC של רצף מידע כלשהו, כפי שיוסבר בהמשך. לדוגמה, הפולינום היוצר הקבוע של CRC32 הוא:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

וכאמור הוא יוצר את הרצף הבינארי הבא, שישמש כמחלק:

100000100110000010001110110110111 = 0x104C11DB7

כעת, נעבור לפעולת האלגוריתם: תחילה נקבל רצף מידע, לדוגמה hello. נמיר אותו לרצף בינארי. את הרצף הבינארי נמיר למספר בינארי (כלומר נוריד את כל האפסים בתחילת רצף המידע). כעת, כאשר אנו משתמשים ב-CRC n-bit, נשרשר n אפסים לסופו של הרצף הבינארי. כעת, ניקח את n+1 הביטים הראשונים ברצף המידע, ונבצע פעולת xor עם המחלק (שייצרנו ממקדמיו של הפולינום היוצר). לתוצאה נשרשר את שאר הביטים ברצף המידע כמו שהם. נזיז את המחלק לביט הבא שערכו 1 ברצף המידע ונחזור על פעולת האלגוריתם (או באופן שקול, נוריד את כל האפסים מתחילת הרצף ונחזור על פעולת האלגוריתם). בסופו של דבר נקבל רצף בינארי שאורכו לכל היותר n ביטים.

כדי להמחיש את הפעולה, נבצע CRC-3 על רצף של 14 ביטים.

רצף המידע הינו: 11010011101101

נוסיף לו 3 אפסים בסוף ונקבל: 11010011101101000

הפולינום המוסכם עבור CRC-3 הוא $x^3 + x + 1$, שיוצר את המחלק 1011.

נתחיל את פעולת האלגוריתם:

- Data: 11010011101101000
- Divisor: 1011
- Result Data: 01100011101101000

כאן ביצענו את השלב הראשון באלגוריתם. לקחנו את ארבעת הביטים הראשונים של זרם המידע, וביצענו פעולת xor ביניהם לבין המחלק. שאר הביטים נשארו כמו שהם.

נמשיך את פעולת האלגוריתם:

- Result Data: 01100011101101000
- Shifted Divisor: 01011
- Result Data: 00111011101101000
- Shifted Divisor: 001011
- Result Data: 00010111101101000
- Shifted Divisor: 0001011
- Result Data: 00000001101101000
- Shifted Divisor: 00000001011
- Result Data: 00000000110101000
- Shifted Divisor: 000000001011
- Result Data: 00000000011001000
- Shifted Divisor: 0000000001011
- Result Data: 00000000001111000
- Shifted Divisor: 00000000001011
- Result Data: 00000000000100000
- Shifted Divisor: 000000000001011
- Result Data: 00000000000001100
- Shifted Divisor: 00000000000001011
- Result Data: 00000000000000111

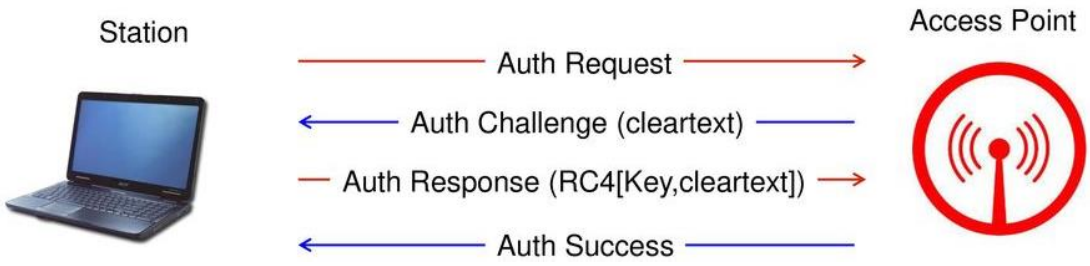
קיבלנו מספר בן 3 ביטים: 111. נצמיד אותו לסופו של זרם המידע המקורי שלנו, ונקבל את זרם המידע 11010011101101111. את זרם המידע החדש נוכל לחלק שוב בפולינום היוצר באותה שיטה בדיוק, ואם נקבל רצף של 0 סימן שזרם המידע תקין ולא השתבש במהלך ההעברה, וכך למעשה הצד השני יכול לבדוק את תקינות המידע.

חולשות ופרצות אבטחה

לאלגוריתם ישנן מספר חולשות מרכזיות, בחלק זה נפרט אודותן.

ניתן לזייף בקלות הודעות אימות

לאלגוריתם WEP קיימות מספר גרסאות: Open System authentication ו-Shared Key Authentication. ההבדל ביניהן הוא האם ה-AP מבצעת אימות. תזכורת:



[מקור: Published by Ethan Montgomery]

למה יש צורך באימות? הרי בכל מקרה המידע מוצפן. ללא אימות, כל אחד יכול להתחבר ולהפיץ את הרשת בפקטות, כדי לפגוע בשירותים שה AP מעניק. כמו כן, תוקף יכול לבצע Replay Attack, בפשטות, לשחזר דבר שכבר קרה. לתוקף יכול לתפוס Frame שאומר לבנק להעביר סכום כסף מהחשבון של אחד המחוברים, ולשלוח אותה שוב...

כדי לזייף הודעת אימות, ניתן להסניף את תעבורת המידע, עד שתימצא בקשת אימות. לאחר מכן, יחזור אתגר ולבסוף הפתרון שלו.

בקלות, ניתן לבודד את ה-Key Stream (לבצע XOR) ולאחר מכן להשתמש ב-Key stream לכל הודעה שתגיע בעתיד, לדוגמא:

challenge= 0x31a1 = 0011 0001 1010 0001

(נשתמש במספר אקראי באורך 16 סיביות בשביל ההדגמה. המספר מוצג בבסיס 16, כלומר בסיס הקסדצימלי):

Encrypted Challenge = 0x3ef0 = 0011 1110 1111 0000

נבצע XOR ונקבל:

Key Stream = 0000 1111 0101 0001

ה- IV קצר מידי

משום שאורך ה-IV הוא 24 סיביות, ל-IV יש 2 בחזקת 24 אפשרויות: 16,777,216 - אפשרויות. ההסתברות לקיום שתי פקטות עם אותו ה-initialization vector מתוך n פקטות: נחשב את ההסתברות למקרה ההופכי - שלכל הפקטות IV ייחודי: (ונחסר)

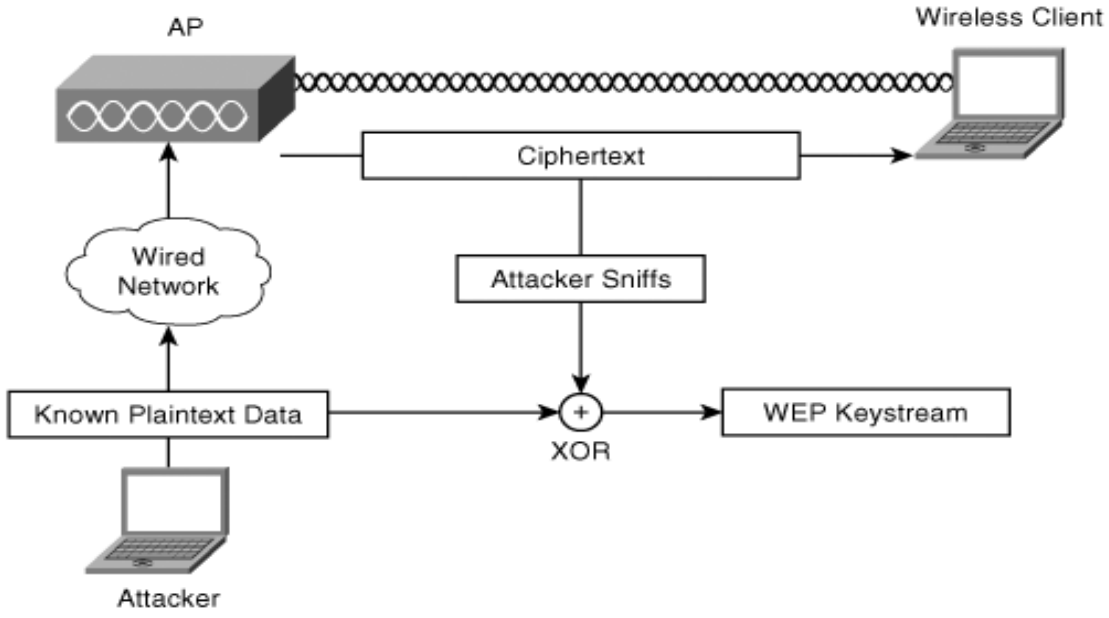
$$\frac{1}{\left(2^{24}\right)^n} \cdot \frac{2^{24}!}{\left(2^{24}-n\right)!} = \frac{2^{24}\left(2^{24}-1\right)\ldots\left(2^{24}-n+1\right)}{2^{24n}}$$

[החישובים נכתבו באמצעות המחשבון symbolab]

(יש 2^{24} אפשרויות ל-IV הראשון, 1 - 2^{24} למספר השני...), למרות שבמבט ראשון, נראה שההסתברות לקיום שתי פקטות היא אפסית, כאשר n = 5000, ההסתברות גדולה מ-50%. ראו - פרדוקס יום ההולדת.

למה זה בעייתי?

אדם שמנסה להשיג גישה ל-AP, יכול להתחבר ל-AP אחר, למשל ה-AP בביתו, לשלוח הודעה, למשל ping, לאחד המחוברים לרשת שהוא רוצה להשיג גישה אליה. הוא בנוסף יסניף את התעבורה וישיג את הפקטה המוצפנת. לאחר ביצוע XOR בין המידע המוצפן שקיבל מההסנפה, למידע הלא מוצפן שנשלח על ידו, המידע יתבטל והתוצאה שתתקבל הינה ה-keystream. כאשר הרצף נתון, ניתן לפענח בקלות כל פקטה שמוצפנת עם אותו ה-IV.



[מקור:etutorials.org]

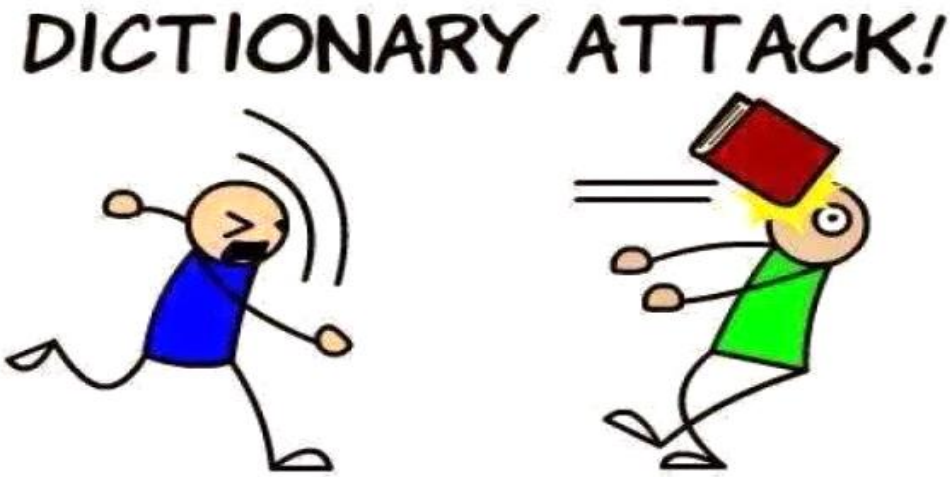
בשיטה זאת ניתן לייצר מילון רצפים (תלוי IV) שיאפשר לפענח לפחות חלק מהפקטות משום שכמות ה-IV האפשריים היא לא כל כך גדולה ואורך רצף מקסימלי הוא 2304 ביטים (ה-MTU של 802.11) לכן, מילון של כל הרצפים ידרוש בערך 38GB, מספר הגיוני בהחלט. כמו כן, ניתן לפענח את המפתח באמצעים שונים. הפתרון המתבקש הוא להאריך את ה-IV, אך הארכת ה-IV תהפוך את התקפת ה-FMS (שתפורט בהמשך) לחזקה יותר, משום שחלק גדול יותר מה- Seed יהיה נתון לפורץ.

ניהול המפתח והאורך שלו

ניהול המפתחות הוא אחד החולשות העיקריות של WEP. לרוב הרשתות המבוססות WEP יש מפתח אחד לכל משתמשי הרשת, מפתח שמוחלף לעיתים רחוקות או בכלל לא, זה בעקבות חוסר הנוחות בהחלפת המפתחות ברשת. כאשר המפתחות לא מוחלפים לעיתים קרובות, רשתות WEP הופכות לקלות לפריצה אפילו יותר.

כמו כן, מפתח באורך 40 סיביות אמנם היה בטוח ב-1997, אך כיום הוא חשוף להתקפות brute force. למפתח WEP 104 יש 2^{104} אפשרויות, מה שמייתר התקפה מסוג זה? לא, כמעט תמיד, יש מחולל סיסמאות, שלוקח סיסמא "נחמדה" כגון password (לא סיסמא מומלצת לכל הזדהות, משום שהיא אחת מהסיסמאות הנפוצות ביותר וקל לנחש אותה) וממיר אותה לסיסמא פחות נחמדה, למשל 20b7e73523.

לכן, ניתן להשתמש ב-Dictionary Attack (בפשטות לקחת מילון של סיסמאות נפוצות, להעביר אותן במחולל ולבדוק אם הן מתאימות)



[מקור: devhumor]

זה לא הכל, למרות שמחולל סיסמאות הוא לא קבוע מראש ושונה אצל ספקים שונים, לעיתים קרובות הספקים השונים משתמשים במחוללים זהים. בעקבות המבנה של המחוללים, הם לא כיסו את כל האפשרויות לסיסמאות WEP.

במקום מעל לטריליון סיסמאות אפשריות, Tim Newsham הציג בכנס BlackHat שישנן רק 2 בחזקת 21 אפשרויות לסיסמא, כלומר בערך 2 מיליון אפשרויות. Newsham 21-bit WEP attack היא התקפת brute force שמפצחת סיסמאות של wep בראוטרים מחלק גדול של הספקים בפחות מדקה.

The KoreK chopchop attack

קורק (השם בדוי) פרסם התקפה נוספת, בשם KoreK Key Recovery Attack, התקפה שמבוססת על רעיונות דומים ל-FMS ומאפשרת לגלות את המפתח, אך היא מודרנית יותר (פורסמה בשנת 2004).

ההתקפה Chopchop שקורק פרסם (בשנת 2004), מאפשרת לפענח את ה-m ביטים האחרונים של המידע, עם 128m חזרות בממוצע ומאמץ חישובי זניח.

ההתקפה משתמשת במימוש לקוי של ה-CRC ומניחה שקיים אורקל, שבודק האם ה-checksum בפקטה מוצפנת הוא נכון או לא.

בניגוד להתקפה FMS, שהיא ההתקפה הראשונה שמפענחת את המפתח, ההתקפה לא תוצג במלואה, אלא רק הרעיונות המרכזיים שלה. לקריאה נוספת, ראו:

https://www.aircrack-ng.org/doku.php?id=chopchoptheory

הסימן ⊕ ייצג את פעולת ה-XOR אלא אם כן צויין אחרת, נסמן את המידע הלא מוצפן בתור D, את ה-checksum בתור J ו- I ואת keystream בתור K. כמו כן, נסמן 5X = I3+D (תזכורת, הכוונה ל-XOR)

Frame 1:

```
___ DATA ____     ___ICV ___
D0 D1 D2 D3 D4 D5 J3 J2 J1 J0
⊕  ⊕  ⊕  ⊕  ⊕  ⊕   ⊕  ⊕  ⊕  ⊕
K0 K1 K2 K3 K4 K5 K6 K7 K8 K9
=  =  =  =  =  =  =  =  =  =
S0 S1 S2 S3 S4 S5 S6 S7 S8 S9
```

Frame 2:

```
___ DATA ___   ____ICV ____
D0 D1 D2 D3 D4 I3 I2 I1 I0
⊕  ⊕  ⊕  ⊕  ⊕   ⊕  ⊕  ⊕  ⊕
K0 K1 K2 K3 K4 K5 K6 K7 K8
=  =  =  =  =  =  =  =  =
R0 R1 R2 R3 R4 R5 R6 R7 R8
```

אנחנו ננסה להגיע מ-Frame1 ל-Frame2 באמצעות הניחוש של X (נתייחס אל X בתור נתון).

מתקיים ש:

R0 - R4 = S0 - S4 (בהתאמה)

$R5 = I3 \oplus K5 = I3 \oplus (D5 \oplus D5) \oplus K5 = (I3 \oplus D5) \oplus (D5 \oplus K5) = X \oplus S5$

את I2 – I0 ניתן לגלות באמצעות J3 – J1, ניתן לגלות לאחר חזרה בשלבים של האלגוריתם CRC, עם שימוש ב-X.

כדי לבדוק שה-X שנבחר נכון, תוקף יבדוק האם ה-checksum של ה-Frame שיוצר נכון, במידה וכן, ה-X שנבחר נוחש נכון, אחרת, צריך לנסות שוב. איך ניתן לבדוק זאת?

1) תוקף יכול להתחבר לרשת עם תחנה A, משום שניתן לזייף בקלות אימות. התוקף ישלח את הפקטה שאותה הוא רוצה לבדוק, מ-A ליעד המקורי. אם ה-checksum נכון, ההודעה תימסר ויהיה ניתן להסניף אותה. אחרת, ההודעה תושלך על ידי ה-AP.

2) דרך נוספת, היא לשלוח את ההודעה עם תחנה שלא מחוברת ל-AP. במידה וה-checksum נכון, ה-AP ישלח הודעת שגיאה ויבקש ממנו להתחבר לרשת, אחרת, ה-AP יפיל את ההודעה.

מה השגנו בעצם?

באמצעות X, ניתן להשיג את J0, שתלוי רק בו (בעקבות המבנה של CRC):

$S9 = J0 \oplus K9$

$K9 = J0 \oplus S9$

ומשום שגם S9 וגם J0 נתונים, השגנו ביט אחד של ה-keystream. על ה-Frame שיצרנו ניתן לבצע את התהליך שוב פעם, כדי לקבל ביט נוסף...

The Fluhrer-Mantin-Shamir Attack

ההתקפה נקראת בקיצור FMS Attack. היא מנצלת מימוש לקוי של ה-RC4 ב-WEP. נסמן את ה-keystream בתור מערך של ביטים X, למשל X[0] הוא הביט הראשון ב-keystream. תיבת התמורה תסומן באות S והמערך K ייצג את הסיסמא (בדומה להדגמת ה-RC4).

נניח שאנחנו יודעים את ה-l ביטים הראשונים בK, ונרצה לגלות את K[l], הביט ה-l+1 כמו כן, X[0] נתון. (משום שבתוכן המוצפן נכללים גם ה-headers של השכבות הגבוהות יותר, ניתן לגלות יחסית בקלות את הביט הראשון של המידע הלא מוצפן ולבצע XOR).

כמו כן, נדרש ש-l=>2 אך משום שה-IV נתון, שלושת הביטים הראשונים של הסיסמא נתונים.

נסמן Slבתור תיבת התמורה לאחר l פעמים, כלומר כשב- KSA מתקיים ש: i = l - 1. את ההופכית נסמן בתור (S^)-1. אנחנו יודעים את S l משום שניתן לבצע את ההתחלה של KSA, מאותה הסיבה אנחנו יודעים את ערכו של J (ואת ערכו של i) במיקום ה-l.

נדרוש את התנאים הבאים (נשתמש רק בסיסמאות המקיימות את התנאים כי הם ברי בדיקה): (הסיסמאות שמקיימות את התנאים הבאים ב-l נקראות resolved condition).

1. $S_l[1] < l$
2. $S_l[1] + S_l[S_l[1]] = l$
3. $S_l^{-1}[X[0]] \neq 1$
4. $S_l^{-1}[X[0]] \neq S_l[1]$

נסתכל על X[0], השוויון הבא מתקיים (פשוט להסתכל ב-PRGA):

X[0] = S[S[1] + S[S[1]]]

RC4 KSA :	**RC4 PRGA :**	$\text{Swap}(S[x], S[y])$
For $i = 0$ to $N - 1$ do:	$i = j = 0$	$t = S[x]$
$S[i] = i$	**Loop :**	$S[x] = S[y]$
$j = 0$	$i = i + 1$	$S[y] = t$
For $i = 0$ to $N - 1$ do:	$j = j + S[i]$	
$j = j + S[i] + K[i \bmod kLength]$	$\text{Swap}(S[i], S[j])$	
$\text{Swap}(S[i], S[j])$	Return: $S[S[i] + S[j]]$	

[מקור: ויקיפדיה - RC4]

לכן, אם מתקיים ש:

If neither Sl [1], Sl [Sl [1]] nor S(l +1)[l] did participate in any further swaps in the rest of the RC4-KSA.

אזי:

X[0] = S[S[1] + S[S[1]]] = S[l] = S(l +1)[l]

משום ש Sl[l]מוחלף עם SlJ[J(l + 1)], נובע ש:

Sl[J(l + 1)] = S(l +1)[l]

ולפי הנוסחה של J:

[Sl[J(l + 1)] = Sl[Jl + K[l] + Sl [l]

נקבל ש:

X[0] = S(l +1)[l] = Sl[J(l + 1)] = Sl[Jl + K[l] + Sl [l]]

X[0] = Sl[Jl + K[l] + Sl [l]]

ולבסוף:

$$\mathcal{F}_{fms}(\mathsf{K}[0],\ldots,\mathsf{K}[l-1],\mathsf{X}[0]) = \mathsf{S}_l^{-1}[\mathsf{X}[0]] - j_l - \mathsf{S}_l[l]$$

היא פונקציה שמחזירה את K[l], במידה וההנחה מתממשת. אחרת, היא מחזירה מספר שהוא בערך רנדומלי.

מהו הסיכוי שיתקיים התנאי?

הסיכוי שבכל פעם J לא יהיה אחד משלושת הערכים הוא (n-3)/n כאשר n הוא האורך של S (לרוב n=256). לכן, הסיכוי שההנחה מתממשת הוא:

$$\frac{(n-3)^{n-l}}{n^{n-l}}$$

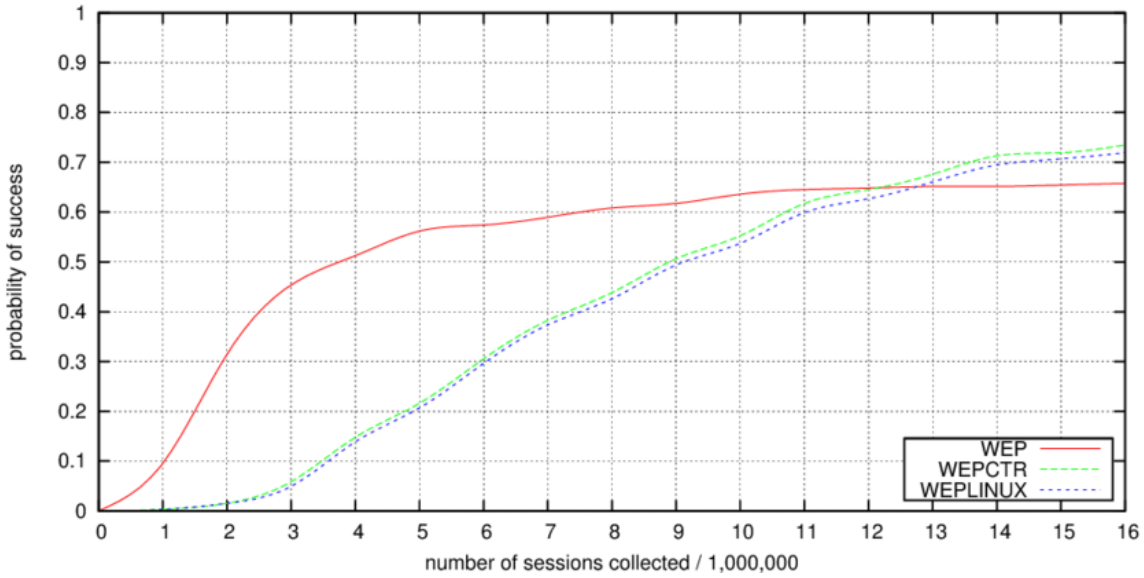
כאשר n=256 ובנוסף l = 3 , הסיכוי הינו בערך 5.07%.

מימוש FMS

תוקף יצבור כמות לא מבוטלת של IV ואת ה-X[0] שמתאים להם, הוא יבדוק אילו סיסמאות מכילות את המצב resolved condition, כאשר שלושת הביטים הראשונים של הסיסמא ידועים (משום שהם ה-IV).

על כל אחד מהם, תתבצע הפונקציה $\mathcal{F}_{fms}$ והתוצאה תישמר. כאשר יש מספיק IV, ניתן להניח שהביט הבא של המפתח התקבל.

התוקף יבצע שוב את אותה הפעולה, אך כשידועים ארבעה ביטים של המפתח... עד שהתוקף קיבל סיסמא שבסבירות יחסית גבוהה היא הסיסמא הנכונה.



[מקור: https://eprint.iacr.org/2007/471.pdf]

הגרף מייצג את סיכויי ההצלחה של FMS attack בתוכנה aircrack-ng, כשזמן העיבוד מוגבל לשלוש דקות. למרות שאחוזי ההצלחה אינם גבוהים במיוחד, הם גבוהים מספיק כדי לאיים על הבטיחות של ה-AP.

התקפות יותר עדכניות כמו KoreK (key recovery attack) משיגות אחוזי הצלחה גבוהים יותר ודורשות פחות מידע מוצפן. ההתקפה שפיתח KoreK למשל, משיגה בערך 97% הצלחה, עם עשירית מכמות ההפעלות. בניגוד להתקפה FMS שמשיגה פחות מ-80% הצלחה, גם כשכמות ההפעלות היא גבוהה מאוד (זמן עיבוד נוסף משפר מאוד את התוצאה של FMS באופן מאוד משמעותי).

סיכום

רשתות אלחוטיות הן דבר שכיח ונמצאות בכמעט כל בית, מסעדה, משרד... אמנם רשתות אלו מאפשרות ניידות ונוחות מירבית, אך יש להן גם חסרונות. המידע מועבר במרחב פתוח, באמצעות גלי מיקרו למשל. לאור זאת, המידע המועבר חשוף להאזנה של כל מי שנמצא בסביבת המשדר.

כדי למנוע חשיפת מידע רגיש יש לנקוט אמצעי הגנה ואבטחה על הרשת האלחוטית. לשם כך נכללים בתקנים השונים אמצעי הצפנה, כמו WEP, שנועדו להבטיח שרק הצד המשדר והקולט יוכלו לפענח את התוכן המשודר. מנגנון ההצפנה WEP הוא מנגנון הצפנה יחסית פשוט, שנועד לאבטח רשתות אלחוטיות כדי להעניק פרטיות שקולה לזאת של חיבור קווי.

כפי שהוצג, המנגנון כולל כמה חולשות וכשלים מבניים, המאפשרים להשיג את המפתח לרשת או לחלופין לפענח את המידע ללא המפתח הצפנה. נכון לעכשיו, מומלץ להשתמש ב- WPA2, אלגוריתם הצפנה חדש יותר ומאובטח יותר. אולם גם ב- WPA2 ישנן בעיות אבטחה ולכן כדאי להשתמש בגרסה עדכנית של המנגנון. זאת ועוד, WPA3 כבר הוכרז וצפוי להחליף את WPA2 בשנים הבאות. כיום, השימוש ב- WEP רק מעניק אשלייה של אבטחה וכלים מודרניים מאפשרים לפרוץ לרשת מבוססת WEP בדקות וחברות שלא עדכנו את האבטחה נחשפו לפריצות, למשל TJ Maxx שנגנבו ממנה פרטי אשראי של 94 מיליון לקוחות.

על עצמנו

יהלי הכט, בן 15 מרעננה, לומד בכיתת מחוננים בחטיבת השרון (רעננה) ומשתתף בתוכנית אודיסאה, מרכז מדעני העתיד, באוניברסיטת תל אביב לנוער. במסגרת תכנית אודיסאה, שבה אנחנו לומדים במסלול סייבר בשנה א', התבקשנו לכתוב מאמרים בנושא הקשור לרשתות תקשורת בהנחייתו של שלומי בוטנרו. בעקבות החיבה שלנו למתמטיקה, בחרנו לכתוב על WEP, מנגנון שאמנם האבטחה שלו ניתנת לפריצה על ידי כלים אוטומטיים כבר לא מעט שנים, אך למרות זאת, מרתק ללמוד על הפרצות, גם אלו שלא הוכנסו למאמר לבסוף .

מתן ונגוש, בן 15 מכפר סבא, לומד בכיתת מחוננים בחטיבת הביניים על שם משה שרת בכפר סבא ועולה לכיתת המחוננים בבית הספר התיכון על שם יצחק רבין בכפר סבא, לומד לתואר ראשון במדעי המחשב בתכנית אודיסיאה, מרכז מדעני העתיד, באוניברסיטת תל אביב לנוער.

במהלך הלימודים בתכנית אודיסאה, נדרשנו לכתוב מאמרים, כאשר הנושא הכללי היה רשתות תקשורת. בחרנו לכתוב את המאמר על פרוטוקול האבטחה WEP בגלל העניין והאתגר המתמטי שהוא עורר אצלנו, ורצינו ללמוד עליו עוד ולשתף את המידע שצברנו.

מקורות

IEEE 802.11 WEP (Wired Equivalent Privacy) Concepts and Vulnerability

- https://pdfs.semanticscholar.org/beb2/90fb1db86715c5ddb256245bff2ca1bb1dec.pdf

What's Wrong With WEP?

- http://www.opus1.com/www/whitepapers/whatswrongwithwep.pdf

WEP Keystream and Plaintext Recovery

- http://etutorials.org/Networking/Wireless+lan+security/Chapter+6.+Wireless+Vulnerabilities/WEP+Keystream+and+Plaintext+Recovery/

Attacks on the WEP protocol

- https://eprint.iacr.org/2007/471.pdf

What is RC4?

- https://paginas.fe.up.pt/~ei10109/ca/rc4.html

Computer Network | RC4 Encryption Algorithm

- https://www.geeksforgeeks.org/computer-network-rc4-encryption-algorithm/

RC4 cipher is no longer supported in Internet Explorer 11 or Microsoft Edge

- https://support.microsoft.com/en-us/help/3151631/rc4-cipher-is-no-longer-supported-in-internet-explorer-11-or-microsoft

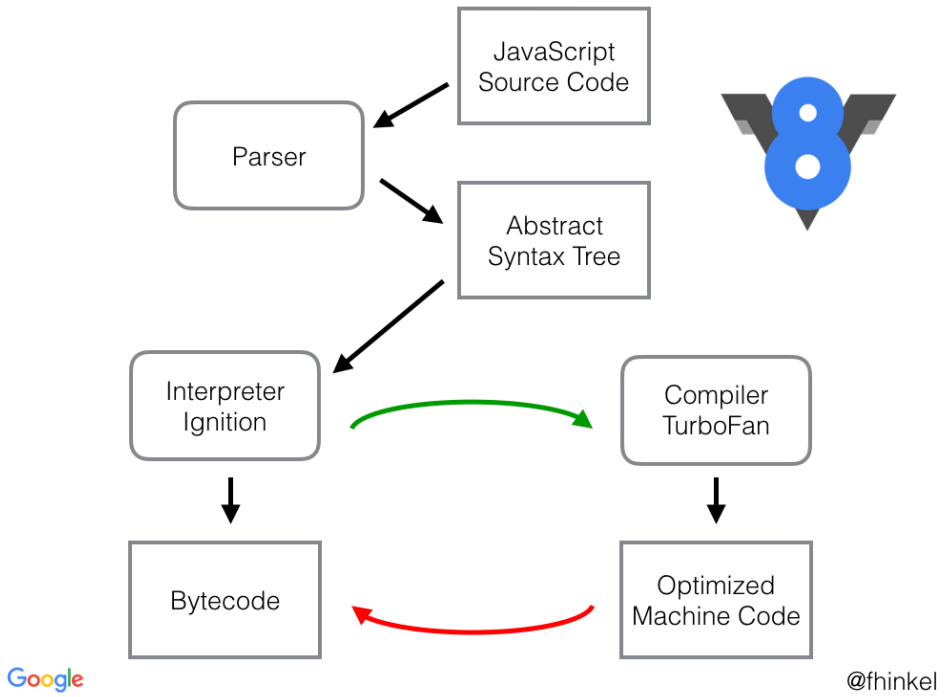
מאת רון דלאל

הקדמה

WebAssembly (או בקיצור - WASM) היא דרך חדשה להריץ קוד על ה-Browser שלכם, בצורה מהירה. כותבים בשפה כמו C/C++/Rust, מקמפלים את הקוד ל-WebAssembly, והוא רץ על הדפדפן שלכם, כחלק מהקוד של האתר. כדי להבין איך היא עובדת, ואת היתרונות שלה, נצלול קודם לסיפור מקדים על איך בכלל Browser Engines עובדים.

מבוא ל-Browser Engines

איך הקוד שלכם רץ על הדפדפן? נדגים את זה על Chrome:



Parser

לפני שנדבר על תהליך הקומפילציה שJavascript עוברת, נדבר מעבר על מאפיין חשוב בשפה, שמשפיע על צורת הקימפול שלה.

כפי שכבר כנראה אתם יודעים, Javascript היא שפה שהיא Dynamically Typed, שזה בדרך כלל כיף מאוד למפתחים, והרבה פחות כיף מאוד לכותבי הקומפיילרים של השפה.

נסתכל על קוד Javascript לדוגמה:

```
const dog = {happy:true};
const my_dog = Object.create(dog);
console.log(my_dog.happy);
```

בכדי להריץ את השורה האחרונה, הParser יבצע קודם כל את הבדיקה הבאה:

```
my_dog.hasOwnProperty("happy");
```

זאת אומרת, הוא יבדוק האם לאובייקט dog יש את השדה happy. אם לא, אז הוא ממשיך בשרשרת הירושה למעלה, ובודק (בדוגמה שלנו הוא יעצר ב-dog). יש עוד המון בדיקות שקורות בדרך, בחרתי להתמקד ספיציפית בזו, כי היא דיי נפוצה.

זכרו את החלק הזה - הבדיקות האלו לוקחות זמן, ובשפות שהן Dynamically Typed, הן חלק לא מבוטל מזמן הריצה.

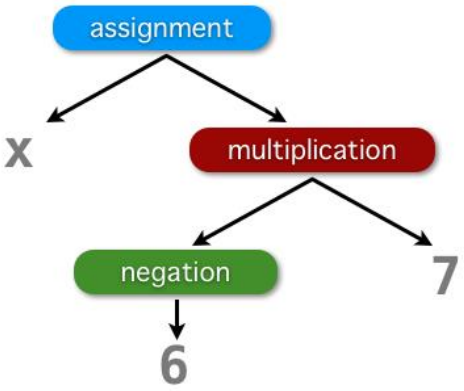
Abstract-Syntax-Tree

אז בדוגמה שלפנינו אפשר לראות את V8 (מנוע הJavascript של Chrome), שלוקח את הקוד שכתבנו ומיצר ממנו Abstract Syntax Tree - בעצם, הצגה כללית מאוד, המתארת את מהלך הקוד.

לדוגמה, עבור קוד ה-Javascript:

```
let x = -6 * 7;
```

נקבל את ה-AST הבא:



מכאן, נוח להעביר את ההצגה הכללית הזו, כמעט לכל שפה.

ByteCode & JIT

בשלב הבא, הקוד יגיע ל-Ignition Compiler, שזה שם יפה (יש המון שמות יפים לדברים בChrome, כמו שלמודול שאחראי על העוגיות קוראים CookieMonster) למוצר שהופך את ה-Abstract Syntax Tree ל-Bytecode, ומשם, ל-Assembly.

- ByteCode - אבסטרקציה של קוד מכונה / סט פקודות מופשט שמוכר למי שאמור להעביר את זה לAssembly (במקרה של V8 זה TurboFan), אבל הוא לא מפורט כמו Assembly אלא בעיקר מהווה שפת ביניים זריזה ופשוטה.
  לדוגמה, עבור הקוד הבא:

```
let result  = 1 + obj.x;
```

נקבל את ה-Bytecode הזה:

```
ldaSmi [1] ; load small integer 1 (Smi == small integer)
Star r0 ; into r0
LdaNamedProperty a0, [0], [4] ; get obj.x
Add r0, [6] ; add 1 to obj.x
```

בונוס - אם תרצו לראות את ה-Bytecode שיוצא מ-v8 - אפשר פשוט לרשום את האופציה print-bytecode בתור flag בשורת הרצה של Chrome או Node.js

הספקנים (או אנשי C\CPP), יכולים לשאול "למה שנרצה בכלל להעביר ל-ByteCode ולא ישר להפוך את כל ה-Javascript ל-Assembly?" שאלה מעולה.

1. לפני שבכלל נתחיל לרוץ, זה יקח לנו המון זמן להעביר את כל הקוד ל-Assembly
2. זוכרים ש-Javascript היא Dynamically Typed? זה אומר שצריך להעביר את כל האופציות האפשריות של הקוד ל-Assembly, וזה פוגע משמעותית בזמן הריצה (יעילות מקום וזמן ריצה)
3. יכול להיות שחלק מהקוד לא יקרא בריצה הנוכחית, או לפחות בזמן הקרוב (לדוגמה - פייסבוק יטענו לכם את הlive של דודה שלכם ברקע, כשאתם רק באתם לראות מה הצעת החברות החדשה שקיבלתם).

אכן בעבר, כשאתרים היו קומפקטיים (הרבה) יותר, ההמרה הזו הייתה קורת, וכל הJavsacript היה הופך לAssembly ישירות. הפסיקו עם זה מאז IE9.

אז אחרי שהשתכנענו בצדקת הBytecode, נרצה להריץ אותו. בכדי לעשות כך נקרא ל-TurboFan, שהוא הJIT Compiler, שזה בעצם אומר שהוא אחראי על ההמרה מ-Bytecode ל-Assembly.

משמע, הקוד שראינו מקודם, יהפוך ל:

```
movl rbx, [rax+0x1b]
REX.W movq r10, 0x100000000
REX.W cmpq r10, rbx
jnc 0x30d119104275
REX.W movq rdx, 0x100000000
call 0x30d1191043e0
Int3laddl rbx, 0x1
....
```

שזה ממש Assembly שמתאים למעבד.

- JIT (Just-In-Time) Compiler - אחראי על העברה של ה-ByteCode ל-Assembly, בדיוק ברגע שבו קוראים לקטע הקוד הנחוץ. לכן הוא נקרא Just-In-Time, ולא Ahead-Of-Time (לדוגמה C היא Ahead-Of-Time, כיוון שהיא מתמקפלת מבעוד מועד ל-Assembly).
  - יכול מאוד להיות שאתם חושבים לעצמכם "רגע...זה לא כזה חכם, אם אני קורא לפונקציה המון פעמים, אז כל פעם להמיר אותה ל-Assembly מ-Byte-Code זה בזבוז זמן רציני".
  - אם חשבתם את זה לעצמכם - קודם כל, כל הכבוד. תחשבו על להגיש משרה לצוות שעוסק בקומפילרים, ועל הדרך, נסו לחשוב על פתרון (מפורט בנקודה הבאה).
  - במקרה של קטע קוד שנקרא המון פעמים - הפתרון היה לסמן אותו כ"קוד חם", ולשמור בצד המרה שלו ל-Assembly, ולהשתמש בה כ-Cache

בואו נסכם את איך ה-JIT עובד, בדוגמה קצרה:

זוכרים את זה ש-JS היא Dynamically Typed? אם מריצים את הקוד הבא:

```
function get_x(obj) {return obj.x;}
for (dog of my_dogs) {get_x(dog)}
```

אז בהתחלה, המנוע יבדוק האם x קיים אצל obj, משמע:

```
obj.hasOwnProperty("x");
```

אם לא, הוא יתחיל לחפש בעץ הירושה שלו (ממי ש-obj יורש ממנו, עד ל-Object, האובייקט שכולם יורשים ממנו ב-JS). ורק אחרי כל מיני בדיקות כאלו ואחרות - הוא ימיר את הקוד ל-Assembly, ויריץ אותו.

בגלל שאנחנו קוראים לget_x המון פעמים, TurboFan (ה-JIT ב-V8), יסמן את הפונקציה כ"פונקציה חמה" וישמור עותק של ה-Assembly מיד, וכל פעם נקרא לאותה פונקציה - מבלי להמיר מחדש.

חשוב לציין - ב-Aseembly של get_x, עבור ארגומנט מסוג dog נקבל אסמבלי שונה מקריאה ל-get_x עם ארגומנט מסוג cat. לכן יש צורך "לקמפל" את הקוד מחדש, כל פעם.

אז אם הקוד שלנו היה מכיל המון קריאות ל-get_x עם המון סוגי אובייקטים שונים?

```
// Define a lot of different classes (dog, cat, duck, etc...)
for (animal of animals) {get_x(animal);}
```

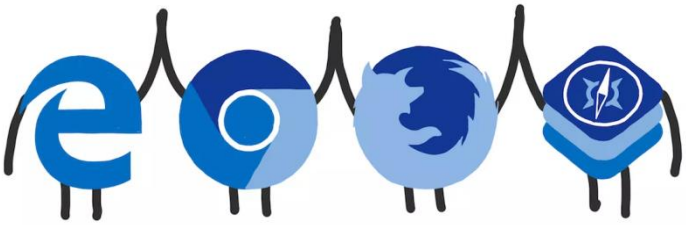
האם TurboFan היה יוצר Assembly עבור כל קריאה של אובייקט שונה?

התשובה היא: לא. בשלב מסויים הוא יפסיק לייצר Assembly חדש לאותה פונקציה. (בTurboFan Default הוא אחרי 4 פעמים שונות).

ומעתה, עבור קריאה ל-get_x - היינו צריכים לעשות את הסדרת בדיקות (שלוקחת זמן בפני עצמה), ואת ההמרה מחדש, מה שהיה יקר מאוד מבחינת זמן ריצה.

זה אחד מהאתגרים שהמנוע מתמודד איתם בשפה שהיא Dynamically Typed, וזה אחד מהאתגרים, שנראה בהמשך - שלא קיימים ל-Webassembly

הקשר ל-WebAssembly

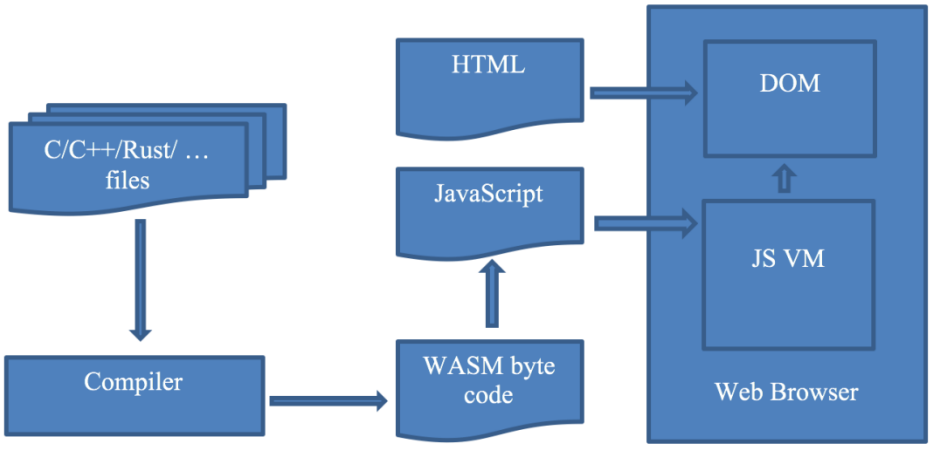


לפני כמה שנים, החליטו כל יצרניות הדפדפנים להתאחד - במטרה לגרום לאפליקציות Web לרוץ מהר יותר, ובכך - לפתוח את עולם ה-Web לאפשרויות רבות הרבה יותר (עיבוד גרפי, משחקים מורכבים, וכו').

המוצר שיצא הוא ה-Web Assembly, או WASM בקיצור.

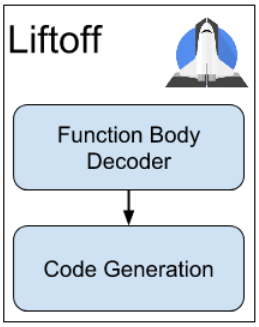
Web Assembly הוא קוד שאפשר לכתוב בשפה כמו C/C++/Rust, "לקמפל" אל ByteCode*, ופשוט להכניס אל תוך הדפדפן, והוא ירוץ כחלק מהאפליקציה. הן חשבו על זה שזה יהיה הרבה יותר מהיר, אם הם פשוט יקבלו את הקוד "מקומפל" אל הJIT שלהם, ויריצו אותו כשהם צריכים.

וזה בערך נראה ככה:

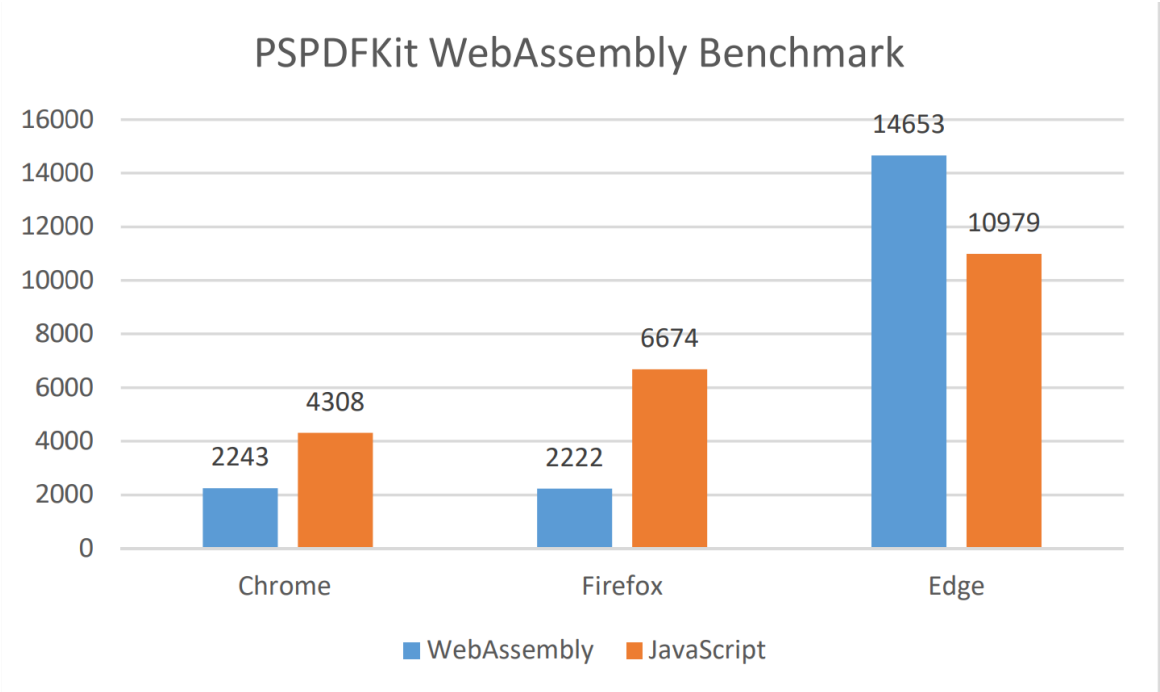


- קוד ה-WASM שלנו, לאחר הקימפול, נכנס עם ה-Javascript אל V8, לחלק משלו, ולא צריך לעבור Parsing מעמיק (אין צורך ב-Parsing, פשוט כי הקוד הוא Statically Typed). הוא מתקבל בתור Bytecode*, ומומר מאוד מהר לשפת מכונה.
- Bytecode* - ה-Bytecode המתקבל מה-Compiler של WASM שונה מזה שראינו עד כה, מהמון סיבות (לדוגמה, הוא בכלל נוצר משפות שהן Staticly Typed).
  - רגע... למה בכלל Bytecode?
  - כל הכבוד, שוב שאלה טובה.
  - הדפדפנים צריכים לרוץ על כל מיני ארכיטקטורות (ARM בטלפונים, x86 ב-PC).
  - כל הדפדפנים צריכים לתמוך בתקן, ולכן יש ליצור שפה אחידה ביניהן

מבחינת יצירת ה-Assembly, אפשר לראות את Liftoff (עוד רכיב בתוך v8), שממיר את ה-bytecode של WASM בצורה פשוטה וזריזה:



יש בו משמעותית פחות חלקים, מה שתורם מאוד גם ביעילות מקום, וגם יעילות זמן ריצה.



[חוץ מבEdge- שעומד בכל מקרה להתמזג לתוך Chrome]

אתם יכולים להריץ את ה-Becnhmark בעצמכם, שמריץ ממש את המוצר עצמו, ולראות את השיפור. נכון לכתיבת שורות אלו, ב-Chrome 75 על לינוקס, ראיתי שיפור של 25%, נתון מעודד מאוד, במיוחד בהינתן שהBenchmark בוחן את המוצר עצמו, ולא סט פקודות תיאורתי.

בכל מקרה, השיפור שנראה בגרף הוא ראשוני יחסית, וישתפר עם הזמן .בעיקר בגלל שהטכנולוגיה עדיין לא הבשילה, והמלאכה עוד מרובה במנועי ה-WebAssembly, בטח בהשוואה למנועי ה- JSהוותיקים.

השיפורים שנראה הם גם באזורי ה-Compilers (אלו שממירים את ה-C/C++/Rust ל-Byte-Code של WebAssembly), וגם באזורי הדפדפנים (לדוגמה, המעבר מ-TurboFan אל Liffoff) שאמורים להאיץ את הקוד WebAssembly לרוץ מהר כמעט כמו המקביל ה-Native-י שלו.

מימושים של WebAssembly

כמו תמיד, עם טכנולוגיות חדשות יש כבר דוגמאות יצירתיות שמסתובבות ברשת - החל מ-VM של Windows2000 שרץ על הדפדפן, ועד למשחקים או עריכת סרטים.

אם נסתכל קצת על העתיד:

- תחשבו על שרתים ב-NodeJS שהחלקים שדורשים פחות חישובים יהיו ב-Javascript, ומה שהכרחי יהיה ב-WASM
- Progressive Web Apps - טכנולוגיה קיימת (נתמכת ב-iOS ו-Android) שמאפשרת לאפליקציות Web לרדת כאפליקציות למכשיר, ולהתנהג בצורה דיי דומה לאפליקציות רגילות.
  - ביצועי האפליקציות האלו יכולים להיות דומים (אם לא טובים יותר) מאפליקציות Native, ולהכתב רק פעם אחת לכל שלושת הפלטורמות הגדולות.

Wasm Hello World

שימו לב לדוגמא הבאה:

כפי שניתן לראות בה, אנו יוצרים פונקציה חדשה בשם doubler, תחילה היא הופכת להיות ה-WebAssembly ByteCode (שנקרא גם Wat), ולאחר מכן (ממש בדפדפן עצמו) מתקמפלת עבור Firefox ועוברת להיות x86 Assembly מוכר וידוע.

יש כמה מדריכים פשוטים באינטרנט שאיתם אפשר להתחיל לשחק עם WASM, שווה לנסות.

סיכום

במאמר זה, פתחנו בהסבר קצר על מנועי דפדפנים, על איך שפות שהן Dynamically Typed (ובתוכן Javascript) רצות, מהו מנוע ה-JIT, ואיך כל זה נראה ב-Chrome.

לאחר מכן, התעמקנו בחסרונות שלהם, וביתרונות של שפות שמתקמפלות Ahead Of Time. משם עברנו ישירות לפתרון המוצע בעולם ה-Web, שהוא ה-WebAssembly. הראנו את השיתוף פעולה (הדיי יחודי) של יצרניות הדפדפנים השונות, בכדי ליצור אפשרות לאפליקציות Web לרוץ מהר יותר.

התעמקנו ביתרונות שלו במצבים מסוימים על פני Javascript, ועל איך הוא עובד. קינחנו בהסתכלות קצרה על הביצועים של WebAssembly כיום, ועל המימושים שלו.

אני מאמין שנמשיך לראות את הטכנולוגיה הזו מאפשר לעוד אפליקציות לרוץ בדפדפן בצורה חלקה וללא הבדלים בין ה-Web ל-Native, ועד מהרה, נמצא את עצמנו בעולם שבו הסטנדרט הוא Write-Once-Run-Everywhere, לפחות בשלושת הפלטפורמות הגדולות (Web, iOS, Android).

מוזמנים לפנות אלי במייל לכל שאלה: rondalal54@gmail.com

תקני Token Ring ו-FDDI, ולמה כבר לא משתמשים בהם

מאת אירי מור ועידו פרידמן

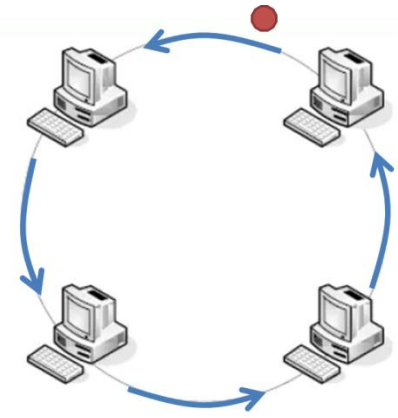
הקדמה

מאמר זה מתמקד בשני התקנים לתקשורת ברשת מקומית Fiber Distributed Data Interface (מבוסס על תקן token bus, IEEE 802.4) שפועל על רשת טבעת כפולה, וב-Token Ring (תקן IEEE 802.5) שפועל עם טבעת לוגית על רשת בטופולוגיית כוכב פיזית. נבצע סקירה של שני התקנים האלו, ושל הסיבות בגללן השימוש בהם כיום הוא מועט עד אפסי.

שכבת הקו (Data Link Layer) היא השכבה השנייה במודל שבע השכבות OSI. שכבת הקו מתחלקת לשתי תת-שכבות: Logical Link Control (LLC) ו-Media Access Control (MAC):

- LLC מספקת את הכתובות ויכולות השליטה בשכבת הקו, נמצאות בה השיטות לשליחת מידע מעל תווך, ולשליטה בו.
- MAC היא השכבה התחתונה יותר בשכבת הקו, ומפרטת את הדרכים לקבוע למי מותר לגשת לתווך ומתי.

Token Ring



Token Ring הוא פרוטוקול ברשתות מקומיות (LANs) אשר עובדות בטופולוגית כוכב (בדרך כלל. ישנם תקנים של Token Ring התומכים ברשת בטופולוגיית טבעת פיזית, בה כל רכיב מחובר לזה שלפניו ולזה שאחריו בלבד) ומשתמשת בהעברת Token לוגי בין הרכיבים ברשת - רק רכיב אשר מחזיק ב-Token יכול לשדר מידע. הפרוטוקולים הנפוצים ביותר של Token Ring הם של IBM, שפורסמו באמצע שנות ה-80. גרסאות של Token Ring יכלו להעביר מידע במהירות של 4Mbps ושל 16Mbps. בכך שרק צומת אשר מחזיק ב-Token יכול להעביר מידע בכל רגע נתון, ומספר ה-Tokens ברשת קבוע ונשלט, Token Ring מונע לגמרי התנגשויות בין סגמנטים של מידע.

טופולוגיות רשת Token Ring

רשת Token Ring היא בדרך כלל רשת בטופולוגית כוכב פיזית (כל הצמתים מחוברים לרכיב אחד המעביר את ה-frames ביניהם), ו-Token Ring יוצר טבעת לוגית מהרשת. טופולוגיית Token Ring משתמשת ב-Multistation Access Units (או Media Access Units, בקיצור MAUs), שמאפשרות חיבור כוכב של הרשת. ה-MAU מחוברת לשאר הצמתים ברשת, כך שאין ביניהם חיבור פיזי. לכן, חיבור הטבעת מתממש רק ברמה הלוגית - כשמידע מועבר ל-MAU, הוא לא נשלח ליעד מסוים, אלא לצומת הבא ברצף הקבוע של הטבעת.

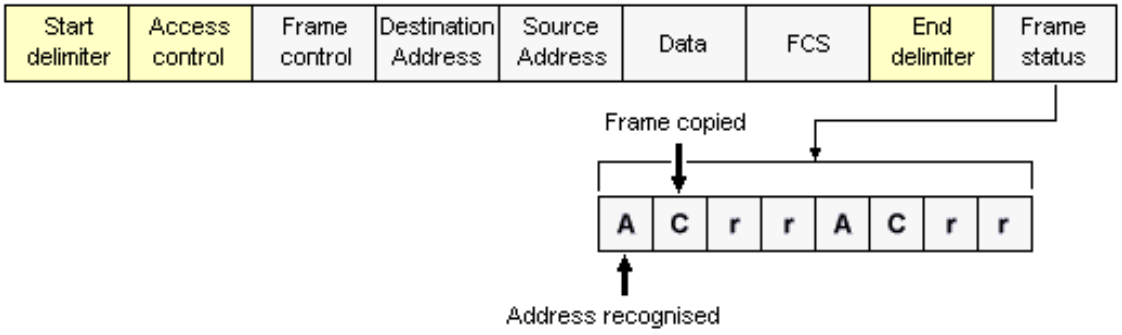
רוב ה-MAUs שייצרה IBM פעלו ללא חיבור לחשמל - היו בהם סדרת ממסרים שהעבירו את המידע לצומת הבא ברשת.

עמידות בפני תקלות

המאפיין המרכזי של Token Ring שמאפשר עמידות בפני תקלות נמצא ב-MAUs. כשיש כשל בצומת כלשהו ברשת (כבל תקשורת פגום, הפסקת פעולת הצומת וכו'), ממסרי המעקף (bypass relays) ב-MAUs מתוכננים להוציא את הצומת שכשל מהטבעת, ולהמשיך את התקשורת בטבעת ישירות בין הצמתים שנמצאו לפני ואחרי הצומת שכשל, והרשת עדיין מחוברת ופעילה.

מבנה ה-Token ודרך העברתו

שיטת העברת Token מבטיחה שלא כל הצמתים ישלחו מידע לרשת באותו זמן - רק מחשבים שמחזיקים ברגע נתון ב-Token יכולים לשלוח חבילות מידע לרשת, וגם אם אין מידע שצריך להעביר ברשת, ה-Token ימשיך "להסתובב" בין הצמתים. Token הוא חבילת מידע של שלושה בתים, ומורכב מהשדות SD, Access Control, ED (בצהוב בתמונה).



- **Start Delimiter** - או SD, מתריע על ההגעה של ה-Token או ה-frame.
- **Access Control** - מכיל את שדה העדיפות (שלושה ביטים שמבטאים את רמת העדיפות של ה-frame), את שדה ה-Token (ביט יחיד, מודיע אם הועבר frame או Token), שדה monitor (ביט יחיד, מונע סיבובים אינסופיים של frames שלא הגיעו ליעדם) ושדה reservation (שלושה ביטים).
- **Frame Control** - שדה של byte יחיד, מודיע אם ה-frame מכיל מידע לצומת אחר או נתוני בקרה (control information). ב-frame שליטה (command frame), שדה זה מפרט את סוג מידע הבקרה שמועבר.
- **Dest. Address, Source Address** - שני שדות של שישה בתים כל אחד, מזהים את כתובות ה-MAC של של צומת המקור וצומת היעד.
- **Data** - המידע המועבר ב-frame. האורך המקסימלי מוגבל על ידי זמן החזקת ה-Token של הטבעת (מאפיין של הרשת שקובע את הזמן המקסימלי שתחנה יכולה להחזיק בו את ה-Token מבלי להעבירו הלאה).
- **End Delimiter** - מתריע על סוף ה-Token או ה-frame, מכיל ביטים שיכולים לשמש לבדיקה האם ה-frame נפגע, ולזיהוי ה-frame האחרון מתוך רצף.
- **Frame Status** - שדה של בית אחד שמסיים את ה-frame, כולל את השדות Frame-copied ו-Address-recognized, שמעידים האם המידע הגיע לתחנת היעד ונקרא. מכיוון ושדה Frame Status לא נכלל בחישוב הערך לבדיקת יתירות מחזורית (CRC), כדי לוודא שהמידע בשדה זה לא נפגע, השדות Frame-copied ו-Address-recognized מופיעים פעמיים ב-Frame Status.

כשלאף תחנה אין מידע להעביר, ה-Token מועבר בין התחנות השונות בטבעת. כאשר לתחנה יש מידע להעביר, היא תסמן את שדה ה-Token ב-Access Control, תוסיף את המידע או את נתוני הבקרה, מה שייצור frame מידע או שליטה, ותשלח את ה-frame לתחנה הבאה. אם התחנה הזו אינה תחנת היעד, היא תעתיק את ה-frame ותשלח אותו מחדש (כך נשמרת עוצמת האות, ומידע לא נאבד) לתחנה שאחריה עד שיגיע לתחנת היעד.

כשיגיע ה-frame לתחנת היעד, היא תקרא את המידע, תסמן את הביטים address-recognized ו- frame copied בשדה Frame Status, ותשדר את ה-frame לתחנה הבאה. כשיגיע בחזרה לתחנת המקור שלו, היא תיצור Token חדש ותעבירו לתחנה הבאה בטבעת (גם אם יש לה מידע נוסף לשלוח). כך, לכל תחנה ברשת יש מספר הזדמנויות שווה לשליחת מידע, ואין התנגשויות.

FDDI - Fiber Distributed Data Interface

FDDI הוא תקן להגדרת תקשורת בשכבה הפיזית, שפותח ע"י מכון התקנים האמריקאי ANSI בועדת התקנים X3T9.5 שהתקיימה באמצע שנות ה-80 של המאה הקודמת. באותה תקופה, מהירויות גבוהות של מחשבים התחילו להופיע ברשתות מקומיות שונות שהיו מבוססות על Ethernet ו-Token Ring, מה שהוביל לצורך בתקן חדש לרשת מקומית שהיה מסוגל לתמוך בקלות במחשבים אלה.

בנוסף, הצורך ברשת אמינה הלך וגבר כשמנהלי מערכות החלו להעביר תוכנות חיוניות ממחשבים חזקים אל הרשת המקומית. FDDI פותח כדי למלא את הצרכים האלו.

התקשורת בין הרכיבים מתבצעת באמצעות סיבים אופטיים, ומהירות ההעברה יכולה להגיע לעד 100Mbps (לשם השוואה, ה-Ethernet בצורתו המקורית תמך במהירות של עד 2.94Mbps, והשתמש בכבל קואקסיאלי לתקשורת).

אבטחת הרשת, אמינותה וביצועיה, כולם משתפרים בעזרת השימוש בסיבים אופטים, מכיוון שסיב אופטי אינו פולט אותות חשמליים - לתווך פיזי שכן פולט אותות חשמליים ניתן לצוטט, מה שמאפשר גישה לא מאושרת למידע שעובר בתווך.

בנוסף, סיבים אופטים חסינים להפרעה של אותות רדיו ואותות אלקטרומגנטים שפוגעים באמינות הרשת. באופן "היסטורי", סיבים אופטים יכולים לתמוך ברוחב פס גדול בהרבה מאשר חוטי נחושת (למעט התקדמויות טכנולוגיות שקרו לאחרונה והצליחו להגדיל את רוחב הפס המקסימלי של חוטי נחושת).

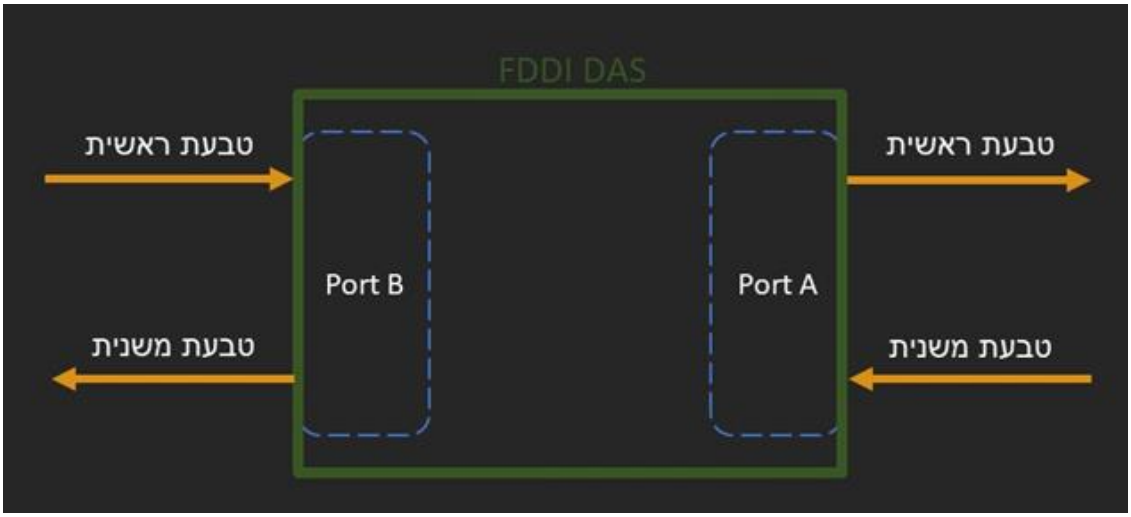
FDDI מבוסס על טופולוגית טבעת, ומשתמש בטבעת כפולה - הטבעת הראשית משמשת להעברת ה-frames, בעוד הטבעת המשנית משמשת כגיבוי, redundancy, במקרה של כשל בטבעת הראשית (למעט בשימוש ב-FDDI Full Duplex Technology או FFDT, בה הטבעת המשנית מעבירה את חבילות המידע במקביל לטבעת הראשית, מה שמגדיל את ביצועי הרשת).

חיבורים פיזיים

כאמור, FDDI משתמש בטבעת כפולה. התעבורה בטבעות מתבצעת בכיוונים נגדיים (הסבר איך זה תורם לאמינות הרשת בהמשך). תחנת FDDI היא צומת ברשת שניתן לפנות אליה (addressable node), המסוגלת לשלוח, להעביר ולקבל מידע. FDDI מגדיר ארבעה סוגים של תחנות:

- **Single-Attachment Station** (SAS)
- **Dual-Attachment Station** (DAS)
- **Single-Attachment Concentrator** (SAC)
- **Dual-Attachment Concentrator** (DAC)

DAS היא תחנה ברשת המחוברת ישירות לשתי הטבעות - הראשית והמשנית - דרך שני חיבורים (port A and port B) שכל אחד מהם מחובר לשתי הטבעות. לכן, כשל (למשל התנתקות או כיבוי) בתחנה המשתמשת בחיבור DAS ישפיע על שתי הטבעות.



[חיבורים חיצוניים של Dual-Attachment Station ב-FDDI]

SAS היא תחנה שמחוברת לרשת רק דרך Concentrator, שיכול לספק חיבורים למספר SASs.

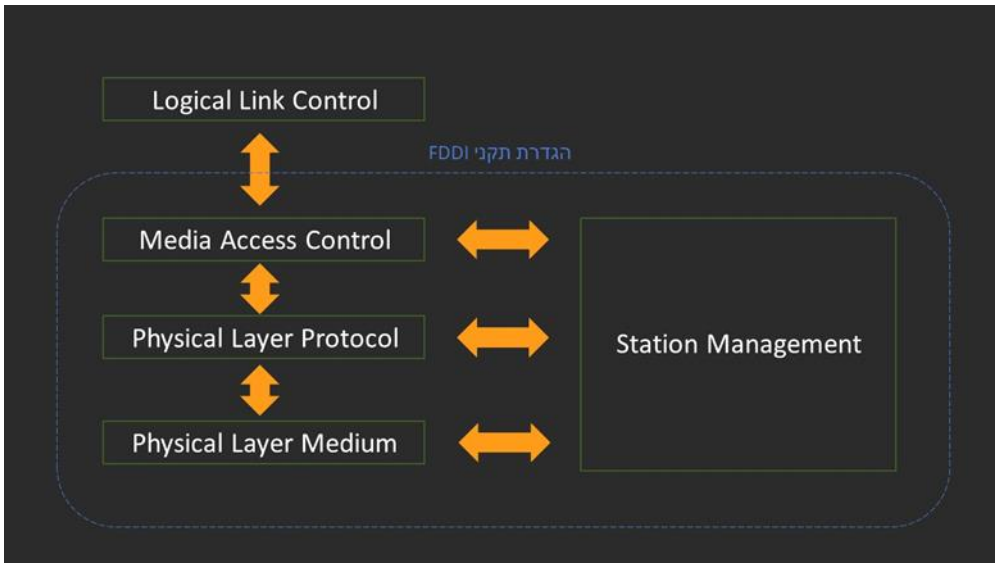
Concentrator יכול להיות Dual-Attachment לשתי הטבעות, או Single-Attachment ל-Concentrator אחר. חיבור SAS ל-concentrator מבטיח שכאשר יש כשל כלשהו ב-SAS (למשל כיבוי של מחשב אישי), או שמכניסים SAS חדש לרשת, לא תהיה השפעה על אף אחת מהטבעות, מכיוון שה-concentrator נשאר מחובר לרשת ומעביר את כל המידע בין הטבעות ל-SASs.

רמות הגדרת תקן FDDI

FDDI מגדיר תקנים בארבע רמות הקשורות זו לזו:

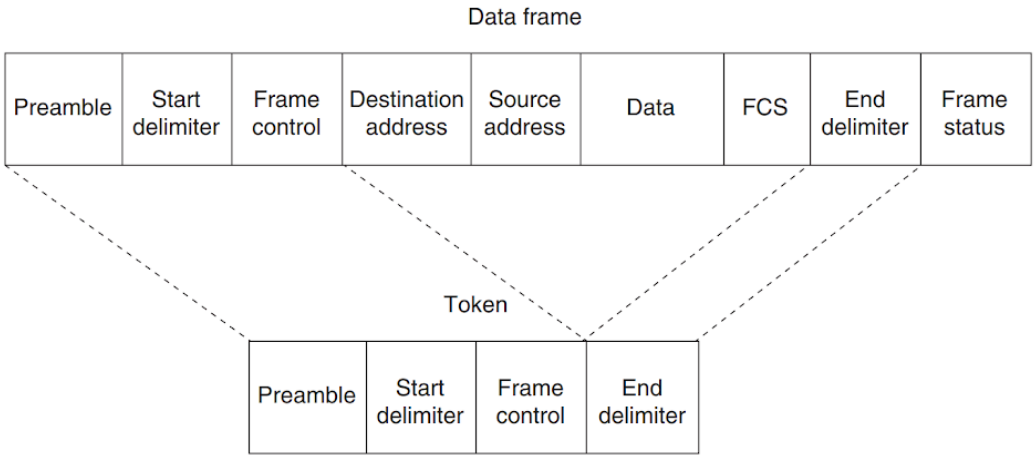
- **Media Access Control** מגדיר את דרך הגישה לתווך הפיזי של העברת המידע (Medium) - כולל פורמט ה-frame, דרך הטיפול ב-Token, האלגוריתם לחישוב ערך לבדיקת יתירות מחזורית (CRC), ודרכי טיפול בשגיאות.
- **Physical Layer Protocol** מגדיר את הליכי ההצפנה והפענוח, דרישות שעון, וכו'.
- **Physical Layer Medium** (נקרא גם **Physical-Medium Dependent**) מגדיר מאפייני תווך השידור - כולל את תכונות הכבל האופטי, כוח השידור, הרכיבים האופטים והמחברים.
- **Station Management** מגדיר את אופן העברת המידע, הטופולוגיה (תפקיד הטבעת הראשית והמשנית) ומאפייני שליטה בטבעות - כולל הכנסת תחנה לרשת, הוצאת תחנה, אתחול הרשת, בידוד וטיפול בתקלות, ואיסוף סטטיסטיקות.

התרשים הבא ממחיש את הקשרים בין ארבע רמות הגדרת התקנים של FDDI ביניהן, ובין ה-LLC בשכבת הקו:



[הקשרים בין רמות הגדרת תקן FDDI]

מבנה frame של FDDI

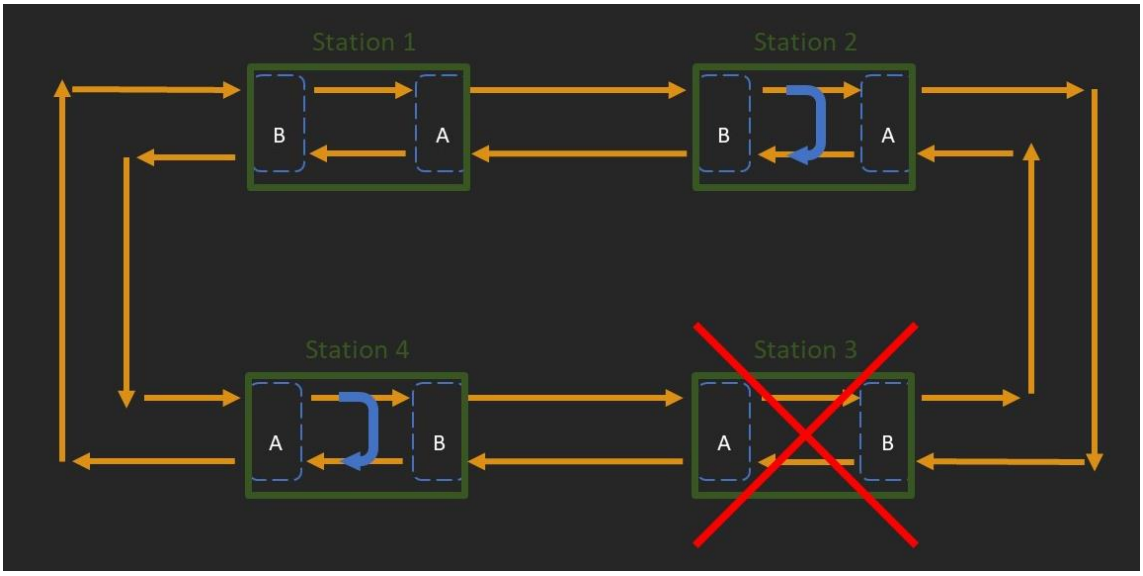


[מקור: https://www.cisco.com/en/US/docs/internetworking/troubleshooting/guide/tr1905.pdf]

- **Preamble** - רצף ייחודי שנועד להכין תחנה לקבלת frame.
- **Start Delimiter** - מתריע על תחילת ה-frame.
- **Frame Control** - גודל השדות בהן מופיעות הכתובות, ועוד מידע בקרה (למשל האם המידע ב-frame הוא סינכרוני או א-סינכרוני).
- **Destination Address** - כתובת היעד של הפריים, יכול להכיל כתובת אחת (Unicast), כמה כתובות ספציפיות (Multicast), או כתובת Broadcast. גודל כל כתובת 6 בתים.
- **Source Address** - כתובת רכיב המקור של הפריים, גודל השדה 6 בתים.
- **Data** - יכול להכיל מידע המיועד לשכבה גבוהה יותר, או מידע בקרה של הרשת המקומית.
- **Frame Check Sequence** - תחנת המקור ממלאת בערך לבדיקת יתירות מחזורית (CRC) שתלוי בתוכן ה-frame (כמו ב-Ethernet), אותו תחנת היעד מחשבת מחדש כדי לבדוק האם המידע ב-frame נפגם בדרך. אם המידע פגום, ה-frame יזרק.
- **End Delimiter** - ערכים המציינים את סיום ה-frame.
- **Frame Status** - מאפשר לתחנת היעד לקבוע אם הייתה בעיה בשליחת ה-frame או אם תחנה בדרך קיבלה ועיבדה את המידע.

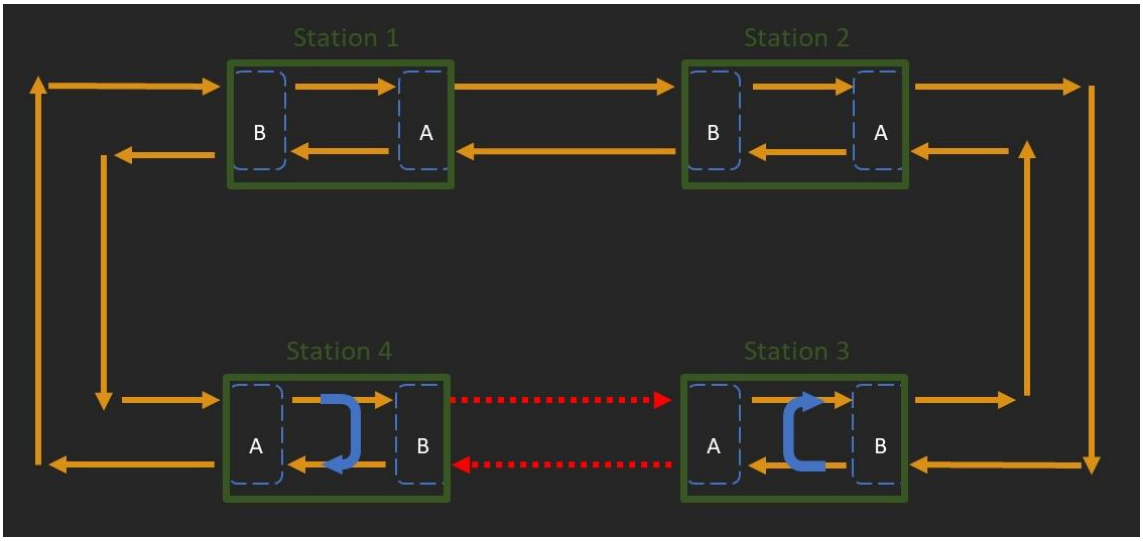
עמידות בפני תקלות

ל-FDDI יש כמה תכונות ההופכות את התקן לעמיד בפני תקלות מגוונות. התכונה החשובה ביותר היא השימוש בטבעת כפולה וב-Ring Wrap. אם יש כשל בתחנה מסוימת על הטבעת הכפולה (למשל נכבית, החיבור הפיזי אליה פגום), הטבעת הכפולה תהפוך באופן אוטומטי לטבעת יחידה (Ring Wrapping), שמנותקת מהתחנה שנכשלה. בתרשים, כשתחנה מספר 3 נכשלת, מתבצע Ring Wrapping בתחנות 2 ו-4 (חץ כחול), מה שיוצר טבעת יחידה, ולמרות שתחנה 3 כבר לא על הטבעת, התקשורת ממשיכה בין שאר התחנות:



[Ring Wrapping כאשר יש כשל בתחנה 3]

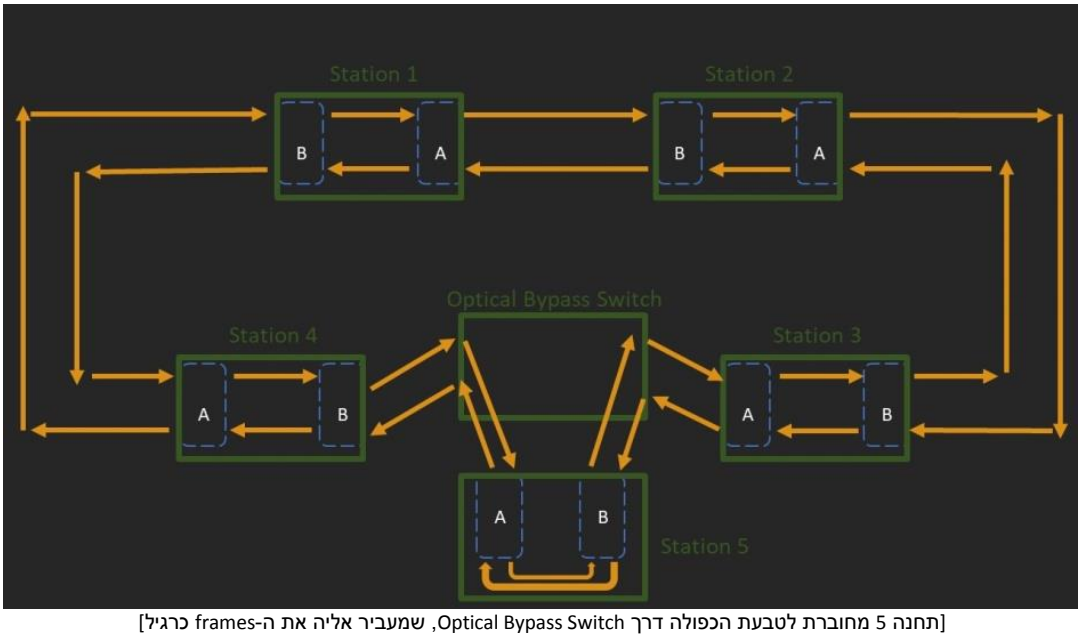
FDDI יכול לפצות גם על כשלים בחיבורים בין התחנות (למשל התנתקות כבל). בתרשים הבא, החיבור בין תחנות 3 ו-4 נכשל, ומתבצע ring wrapping בתחנות אלו על מנת לפצות על הכשל (שוב, נוצרת טבעת יחידה במקום טבעת כפולה):



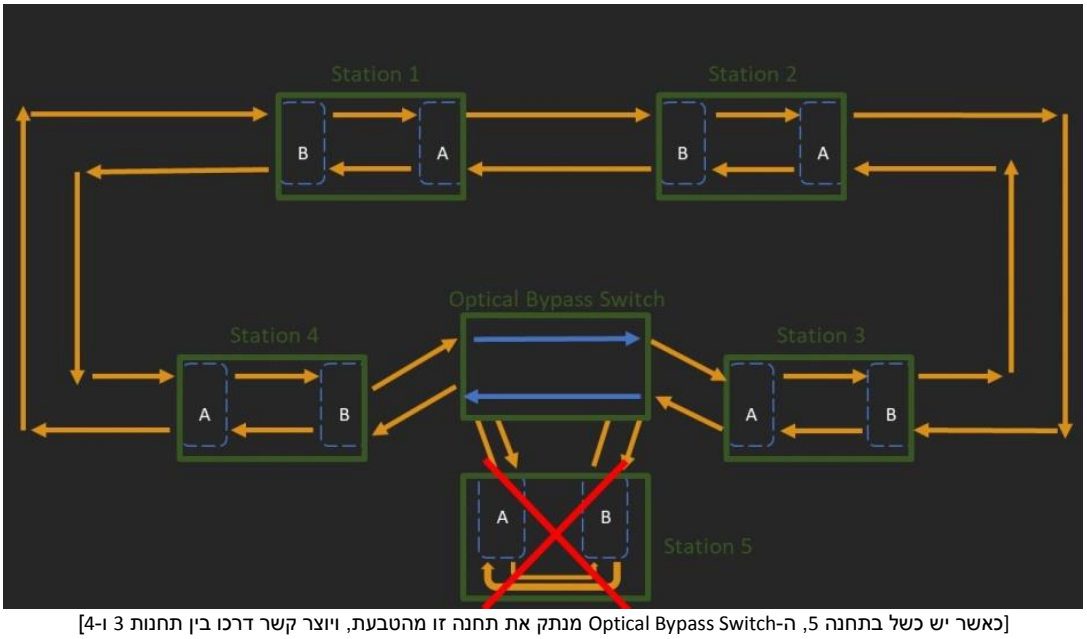
[Ring Wrapping כאשר יש כשל בחיבור בין תחנה 3 ותחנה 4]

נשים לב כי Ring Wrapping יעזור רק כאשר יש כשל יחיד ברשת (בחיבור או בתחנה), וככל שרשת FDDI גדלה, הסיכוי לכשלים מרובים גדל. לכן כשיש שני כשלים או יותר, FDDI יבצע "פיצול" של הטבעת (segmentation) לשתי טבעות או יותר, שיהיו עצמאיות וללא תקשורת ביניהן. כשלים נוספים יגרמו לפיצולים נוספים של הטבעות.

Optical Bypass Switches משמשים למניעה של פיצולים כאלו, ע"י הוצאת תחנות שנכשלו מהטבעת. כל המידע שנשלח לתחנה שמחוברת ל-optical bypass switch מהטבעות עובר דרכו - אם התחנה עובדת באופן תקין, המידע ימשיך מה-switch לתחנה, שתוכל לשלוח תגובה והודעות נוספות. אם יש כשל כלשהו בתחנה, ה-switch יעביר את האותות בסיב האופטי דרך סדרת מראות, כך שימשיכו אל התחנה הבאה "וידלגו" על התחנה שכשלה.



[תחנה 5 מחוברת לטבעת הכפולה דרך Optical Bypass Switch, שמעביר אליה את ה-frames כרגיל]



[כאשר יש כשל בתחנה 5, ה-Optical Bypass Switch מנתק את תחנה זו מהטבעת, ויוצר קשר דרכו בין תחנות 3 ו-4]

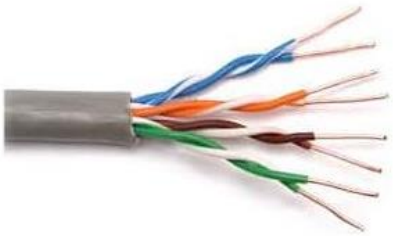
CDDI - Copper Distributed Data Interface

העלות הגבוהה של כבלים אופטים היוותה המכשול העיקרי של תקן FDDI, שמבוסס על העברת המידע דרך כבלים אופטים. לעומת זאת, העלות של כבלי נחושת מסוג Unshielded Twisted Pair (בקיצור UTP) ו-shielded twisted pair (או STP) נמוכה בהרבה ביחס לזו של כבלים אופטים, והם כבר נמצאו בשימוש נרחב. לכן ניתן היה ליישם את FDDI באופן מסוים על גבי כבלי נחושת, ויישום זה נקרא Copper Distributed Data Interface.

התקן הרשמי של ANSI מכיר בדיוק שני סוגים של תווכים עבור CDDI: כבלי UTP ו-STP. הסבר מפורט ניתן למצוא כאן.

הגדרות: זוג שזור הוא כבל חשמלי המורכב מזוג גידים מוליכים (במקרה של CDDI, נחושת), כל אחד מהם מורכב ממוליך פנימי ושכבת בידוד, והם שזורים זה על זה.

- Unshielded twisted pair הוא כבל של ארבעה זוגות שזורים, והוא הסוג הפשוט והנפוץ ביותר; נותן הגנה לקויה מפני הפרעות.



- Shielded twisted pair הוא כבל של ארבעה זוגות שזורים, בו לכל זוג יש סיכוך - שכבת בידוד מתכתית שנועדה להקטין את הרגישות להפרעות.

[כבל UTP ו-STP, מתוך http://www.fiber-optic-components.com/shielded-vs-unshielded-ethernet-cable-use.html]

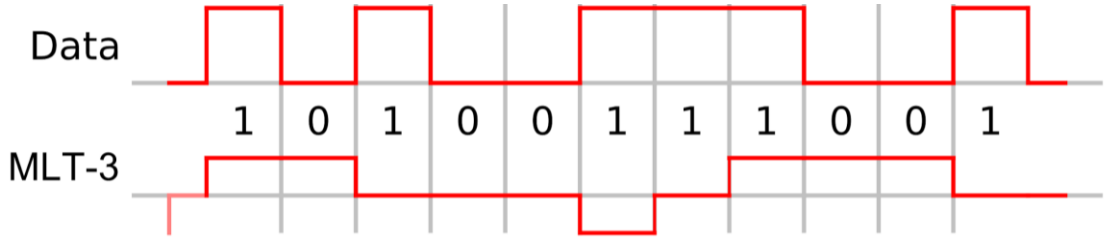
הבעיה העיקרית ביישום של FDDI על גבי כבלי נחושת מסוג UTP או STP היא הקרינה הנפלטת מכבלים אלה בהעברת המידע. נסביר בקצרה על הטכניקות השונות להנמכת הקרינה האלקטרומגנטית הנפלטת מחוטי הנחושת:

- **Scrambling** - כשלא מועבר מידע בתווך, FDDI ישדר דפוס סרק המורכב מתווים בינארים של "1". כשהדפוס הזה נשלח ב twisted-pair wires, ההפרעה האלקטרומגנטית מתרכזת בתדר שבו נשלח דפוס הסרק, מה שגורם לעלייה ברמת המתח בתדר של ההפרעה. כאשר FDDI מערבל את המידע בעזרת רצף פסאודו-רנדומלי לפני השידור, מבוטלים דפוסים חוזרים, וכתוצאה מכך יש עליה במתח המחולקת באופן יותר מאוזן על פני הספקטרום של האות שמשודר.

- **Encoding** - עוצמת האות חזקה יותר וההפרעה האלקטרומגנטית נמוכה יותר כשהשידור מתבצע על גבי twisted-pair wires בתדרים נמוכים. השימוש בסכמת MLT-3 מנמיך את התדר של האות המשודר, כך שהגבוה ביותר מועבר לפחות מ-20Mhz.

  MLT-3 מעביר בין שלוש רמות מתח בסדר הבא: -1, 0, +1, 0. בשביל להעביר "1" בינארי, MLT-3 יעבור לרמת המתח הבאה, ועבור "0" ישאר באותה רמת מתח.

- **Equalization** - הגברת האותות בתדרים הגבוהים בשידור על גבי UTP.



[מקור: https://en.wikipedia.org/wiki/MLT-3_encoding]

בדומה ל-FDDI, תקן CDDI מספק מהירות העברת מידע של 100Mbps ומשתמש בטופולוגית טבעת כפולה. CDDI תומך במרחקים של עד 100 מטרים בין תחנות. שם התקן הרשמי הוא Twisted-Pair Physical Medium-Dependent או TP-PMD, ונקרא גם Twisted-Pair Distributed Data Interface או TP-DDI (עקבי עם השם Fiber Distributed Data Interface).

מסקנות - השוואה עם Ethernet, למה השיטות הוחלפו

היתרון העיקרי של Token Ring היה QoS (או Quality of Service) - רשת Token Ring תישאר יציבה ותפעל תחת כל עומס, בניגוד לרשת Ethernet המשתמשת ב-hubs, שבד"כ יגרמו לתקלות (התנגשויות של מידע) ב-60% עומס, ויקרסו ב-90% עומס. לעומת זאת, כאשר תחנה מחוברת ל-Ethernet Switch, יש רק שני רכיבים שיכולים לגרום להתנגשות - התחנה וה-Switch, וקורות הרבה פחות התנגשויות. בנוסף, ברשתות מקומיות קריטיות (בהן נדרשת אמינות ומהירות גבוהה של הרשת) מתבצע שימוש בטופולוגיית Full Duplex על Ethernet (כל רכיב מחובר לכל הרכיבים האחרים באופן ישיר), ולא קורות התנגשויות של מידע, מכיוון ושני הצדדים יכולים לשדר זה לזה בו-זמנית.

בנוסף, החומרה הנדרשת ברשת Ethernet זולה מאוד, ביחס לסיבים האופטים הנדרשים ברשת FDDI. עבור רשת CDDI המשתמשת בכבלי נחושת הדומים לכבלי Ethernet, מהירות הרשת נמוכה מאוד (עד 100Mbps) ביחס לזו של גרסת ה-Ethernet המהירה ביותר כיום (יותר מ-1Gbps). בתחילת דרכן, Token Ring ו-Ethernet שתיהן השתמשו בכבלים יקרים, אבל כאשר Ethernet תוקננה עבור כבלי Unshielded

Twisted Pair, והומצאו Ethernet Switches, עלות ההקמה של רשת Ethernet ירדה בהרבה. גם העלות של כרטיסי רשת ושאר הרכיבים לרשת Token Ring היו יקרים.

באמצע שנות ה-90, המהירות המקסימלית של Ethernet הייתה 10Mbps ו-Token RIng הציעה 4 או 16Mbps. לכן באותה תקופה, FDDI נחשבה לפתרון טוב עבור רשתות Backbone (רשת מחשבים המחברת בין מספר LANs או Subnets, מהווה נתיב להעברת מידע ביניהן), מכיוון שהציעה מהירות גבוהה - עבור אותה תקופה - של 100Mbps, והעלויות של רשת FDDI היו דומות לאלו של Ethernet. בהמשך נוצרו אלטרנטיבות טובות יותר - Fast Ethernet שהציע 100Mbps ותמך בכבלי נחושת ובסיבים אופטים, ואחריו Gigabit Ethernet עם מהירות של 1Gbps ועלות יותר נמוכה מזו של Token Ring ושל FDDI. שיקולים כספיים אלו נתנו ל-Ethernet יתרון מוחלט על פני Token Ring, והמכירות של רכיבי Ethernet עלו, בעוד השימוש ב-Token Ring וב-FDDI הפך לאפסי.

קצת על עצמנו

- **אירי מור** - בן 14, גר בגבעתיים, לומד בתיכון אהל שם ברמת גן ובתכנית אודיסיאה, מרכז מדעני העתיד, באוניברסיטת תל אביב לנוער במסלול סייבר שנה א'.
- **עידו פרידמן** - גר ולומד ברחובות, בשנה א' בתכנית אודיסיאה באוניברסיטת תל אביב, מסלול סייבר.

המאמר נכתב כחלק מהלימודים בקורס רשתות בתכנית אודיסיאה, מרכז מדעני העתיד, באוניברסיטת תל אביב לנוער במסלול סייבר שנה א', בהנחיית שלומי בוטנרו. בחרנו בנושא זה מכיוון שעניין אותנו מדוע Ethernet הוא כמעט הפרוטוקול היחיד בשימוש ברשתות מקומיות כיום ומה הן האלטרנטיבות הקיימות כיום ובעבר.

ביבליוגרפיה

- Fiber Distributed Data Interface (FDDI) Volume 34 of Fiber Optics Reprint Series, ISBN 1568510837, 9781568510835
- https://www.cisco.com/en/US/docs/internetworking/troubleshooting/guide/tr1905.pdf
- https://www.sciencedirect.com/topics/computer-science/fiber-distributed-data-interface
- https://phdessay.com/fddi-and-cddi/
- https://www.ionos.com/digitalguide/server/know-how/token-ring
- http://www.technologyuk.net/telecommunications/networks/fibre-distributed-data-interface.shtml

iptables unleashed

מאת עדי מליאנקר (Magen_Gide0n)

הקדמה

במאמר הזה, אני מקווה לסגור, או לפחות להמשיך את היריעה ממאמר מצוין שכתב אפיק קסטיאל בדצמבר 2009, לפני כמעט 10 שנים באשר ל-iptables ולהבין את התהליכים שקורים בטבלאות השונות. הרעיון להמשיך את המאמר הגיע מתוך מאמר אחר שיצא לי לכתוב למגזין (אך עוד לא פורסם) באשר להתקדמות ברשת ע"י port forwarding ומתקשר במעט ל-iptables.

קודם כל ולפני כל, נביא כהקדמה קטנה סקירה של המאמר של אפיק, כי 10 שנים, לך תזכור... (למרות שממליץ בחום לקרוא לפני את המאמר הנ"ל).

סקירה מהעבר

אם כן, iptables הינו firewall מבוסס אירועים המכיל תכונת stateful firewalling, כלומר היכולת לבדוק חבילה בפני עצמה וגם בהקשר לשיחה אליה היא שייכת. הכלי נכתב על ידי Rusty Russel עבור מערכת ההפעלה לינוקס ומגיע מובנה בה. הכלי מכיל טבלה ענקית הנקראת filtering table אשר מחולקת לכמה טבלאות בסיסיות לניהול תעבורת החבילות. הטבלאות הן filter (שהיא הטבלה הדיפולטיבית), טבלת NAT האחראית על פעולות לניתוב הפקטות וטבלת mangle לטיפול מתקדם בחבילות.

כל טבלה מכילה כמה שרשראות, אוסף חוקים. לדוגמא, טבלת filter מכילה 3 סוגי שרשראות: input, output ו-forward. מערכת iptables מנתבת את החבילות לטבלאות המתאימות. כאשר input הן החבילות הנכנסות, output הן החבילות היוצאות ו-forward הן החבילות אשר מיועדות לעבור הלאה... כל טבלה מכילה אוסף של חוקים, כמו כל firewall בסיסי ומתבצעת בדיקה האם ישנו חוק כלשהוא בטבלה המדבר על החבילה, **כאשר סדר החוקים הוא חשוב** ואם נמצא חוק שמתאים לפקטה, יתבצע מה שרשום בחוק ויופסק החיפוש (גם אם יש בהמשך חוקים שעשויים גם להתאים לחבילה).

נניח שנתונה לנו הטבלה הבאה:

No.	Protocol	Source IP	Destination IP	Dest. Port	Action
1	TCP	10.1.1.1	20.1.1.1	80	Accept
2	TCP	10.1.1.2	20.1.1.1	80	Deny
3	TCP	10.1.1.0/24	20.1.1.1	80	Deny
4	TCP	10.1.1.3	20.1.1.1	80	Accept
5	TCP	10.2.2.0/24	20.2.2.5	80	Deny
6	TCP	10.2.2.5	20.2.2.0/24	80	Deny
7	TCP	10.3.3.0/24	20.3.3.9	80	Accept
8	TCP	10.3.3.9	20.3.3.0/24	80	Deny
9	IP	0.0.0.0/0	0.0.0.0/0	0-65535	Deny

[מתוך research gate]

במקרה זה, אם תגיע ל-firewall הודעת tcp מ-10.1.1.1 אל 20.1.1.1 בפורט 80, ההודעה מיד תתקבל בחוק מספר 1 (למרות שחוק מספר 3 גם היה עשוי לעניין אותנו, אבל כמו שאמרנו, הסדר חשוב... ולכן לא נגיע בכלל לבדיקת חוק מספר 3).

קיימות 5 אפשרויות להתנהלות אל מול חבילה:

- accept - העברת חבילה
- drop - זריקת חבילה
- reject - כמו drop, אבל תשלח הודעת שגיאה לשולח
- queue - העברת החבילה לשימוש user space
- return - החזרת הקריאה של החוקים לחוק שממנו הופעל החוק הנוכחי).

2 חוקי ברזל חשובים:

- כאשר נוסיף חוקים לטבלה, הם דיפולטיבית, אלא אם ציינו אחרת, יהיו בתחתית הטבלה.
- בדיקת החוקים מתחילה מראש הטבלה את תחתיתה.

עד כאן הסקירה, בואו ניגש לעבודה.

אם כן, כיצד iptables עובד?

הכלי iptables מדבר עם hook-ים של packet filteting ברמת הקרנל, hook-ים השייכים לסביבת netfilter.

מושגים רבותיי, מושגים

מה זה hook?

המושג hook מתייחס להטמנת קוד בסביבת מערכת ההפעלה כך שהקוד יקרא על ידה כאשר תקרה פעולה מסוימת ברמת מערכת ההפעלה. hooking משמש לביצוע debugging, הרחבת פונקציונליות של תוכנית מסוימת וכן כתיבת תוכנות המסוות את עצמן כמו rootkits למיניהן. באחת מבדיקות החדירות שיצא לי לבצע בעבר, יצא לי להשתמש בתוכנית המשתמשת ב-hook-ים על מערכת ההפעלה כדי ליירט הקשות מקשים והקלקות עכבר, keylogger.

NAT - על מה ולמה?

בואו נדבר על NAT , ראשי תיבות של network address translation. כמה פעמים יצא לכם לבצעipconfig ולראות subnet מוזר שדי תמיד חוזר על עצמו בכל בית הישראל- 10.0.0.0/24 או כתובת שמתחילה 192.168, צירוף מקרים? לא ממש. התפקיד של NAT נולד מהצורך ההולך וגובר לאבטחה עבור עמדות הקצה בסביבות פרטיות וכן העודף ההולך וגדל של כתובות ip בעולם. לשם כך, נולד מושג NAT שבו אנו לוקחים סביבות פרטיות-ביתיות או תעשייתיות, מקיפים אותן ב"שומר", שמבצע את הקישור בינן לבין החוץ.

לכל מחשב ברשת הפנימית, ניתנת כתובת פנימית. הכתובת הפנימית רלוונטית רק לרשת הפנימית ולכן ברשתות פנימיות שונות נוכל למצוא כתובות ip זהות. הקישור ממיר בין הכתובת הפנימית אל הרשת החיצונית, האינטרנט. כך, מחשב באינטרנט לא יכול לפנות ישירות אל מחשב ברשת הפנימית (היות והוא לא מכיר ישירות את כתובת ה-ip של המחשב הפנימי והיות ויש הרבה מאוד מחשבים עם הכתובת הזו). מה שהוא כן יכול לעשות, הוא לדבר עם רכיב ה-NAT, השומר, על מנת שיקשר אותו למחשב הפנימי (בדרך כלל הוא גם צריך לחכות שהמחשב הפנימי יפנה אליו כתוצאה מרכיבים כמו firewall הנמצאים בעסק).

אופן הפעולה של iptables

חבילות המגיעות ללינוקס, יעוררו את ה-hook-ים, כאשר הם נכנסים ל-stack החבילות הדורשות פעולה. בפעולה זו, נגרום לכלל התוכנות, kernel modules, הרשומות ל-hook-ים להתעורר ולשמש בתפקיד מפתח בעבודה עם החבילות. ישנם 5 hook-ים ב-netfilter שניתן להירשם אליהם. ה-hook-ים שחבילות יעוררו תלויים באופי החבילה, האם היא נכנסת או יוצאת, מה יעדה, האם היא נפלה או נדחתה לפני כן ועוד.

ה-hook-ים הם:

- NF_IP_PRE_ROUTING המתעורר כאשר נכנסת חבילה כלשהיא למחסנית הרשת ועדיין לא הוחלט לאן לנתב אותה.
- NF_IP_LOCAL_IN מתעורר כאשר מגיעה חבילה המיועדת למערכת המקומית.
- NF_IP_FORWARD המתעורר כאשר מגיעה חבילה המיועדת למארח אחר.
- NF_IP_LOCAL_OUT המתעורר כאשר חבילה כלשהיא יוצאת מהמכונה המקומית החוצה.
- NF_IP_POST_ROUTING המתעורר כאשר חבילה אמורה להיות מועברת הלאה או לצאת מהמכונה והיא קיבלה כבר החלטת ניתוב מhookים אחרים. ה-hook קורה קצת לפני יציאת החבילה החוצה.

kernel modules המעוניינים להירשם ל-hook-ים, חייבים לספק priority number בכדי לקבוע את סדר התוכנות שירוצו כאשר hook מסוים נקרא. כל תוכנה תרוץ כאשר יגיע תורה ותחזיר את החלטה אל ה-netfilter framework לאחר שהחליטה מה לבצע בחבילה.

כמו שאמרנו, כל טבלה ב-iptables מסודרת בעזרת שרשראות. כל טבלה מייצגת את סוג החוקים שהיא מחזיקה (לדוגמא טבלת NAT תחזיק חבילות המתעסקות עם חבילות NAT, לתרגום כתובת פנימית לחיצונית ולהיפך), שרשרת , לעומת זאת, מייצגת hook מסביבת netfilter framework ומחליטה מתי חוק מסוים יקרא (לדוגמא קיים בטבלת FILTER שרשרת מסוג INPUT המתייחס לחבילות הנכנסות פנימה ול-hook של NF_IP_LOCAL_IN וכן הלאה...). השרשרת היא שמחזיקה בפועל את החוקים. נשים לב לפרט חשוב-לכל אחת משלושת הטבלאות, אין שרשראות עבור כל חמשת ה-hook-ים, אלא רק עבור אלה הרלוונטים בשבילם. עוד פרט חשוב הוא שיש חפיפה בין השרשראות בטבלאות השונות (לדוגמא גם בטבלת FILTER וגם בטבלת NAT ישנה שרשרת של output). כיצד זה עובד, אתם שואלים את עצמכם?

כמו שאמרתי קודם, תוכנות המעוניינות להירשם לhook מסוים שולחות אליו את הpriority number שלהן וכך, לדוגמא כאשר ה-hook של NF_IP_LOCAL_OUT יתעורר, אם ה-priority number של output בטבלת FILTER גבוה מהמספר של output בטבלת NAT, יקראו קודם כל החוקים בטבלת output ב-FILTER לפי הסדר ורק לאחר מכן יקראו החוקים בטבלת output ב-NAT.

הטבלאות הינן:

- טבלת FILTER, הטבלה הדיפולטיבית והנפוצה ביותר, הטבלה שנותנת ל-iptables את הכינוי firewall, אחראית על חבילות היוצאות, נכנסות ומועברות.
- טבלת NAT, המממשת NAT ומבצעת המרה של כתובת ip אחת ופורט אחד לאחרים בכדי להיכנס לרשתות שאין גישה ישירה אליהן.
- טבלת MANGLE משמשת לצורך שינוי ה-header-ים של חבילות של ה-ip בדרכים שונות (שינוי ה-type of service, ttl והגדרות סימון).
- טבלת RAW מתייחסת לחבילות בהקשר של מיקומן ב-session או קשר מסוים.

- טבלת security מאפשרת לסמן חבילות מסוימות עבור SElinux (נגנון בלינוקס המאפשר להגביל כניסה של אובייקטים, כמו לדוגמא תהליכים אל קבצים מסוימים) וטיפולה בהן.

בטבלה הבאה ניתן לראות את השרשראות הנמצאות בכל טבלה.

Tables↓/Chains→	PREROUTING	INPUT	FORWARD	OUTPUT	POSTROUTING
(routing decision)				✓	
raw	✓			✓	
(connection tracking enabled)	✓			✓	
mangle	✓	✓	✓	✓	✓
nat (DNAT)	✓			✓	
(routing decision)	✓			✓	
filter		✓	✓	✓	
security		✓	✓	✓	
nat (SNAT)		✓			✓

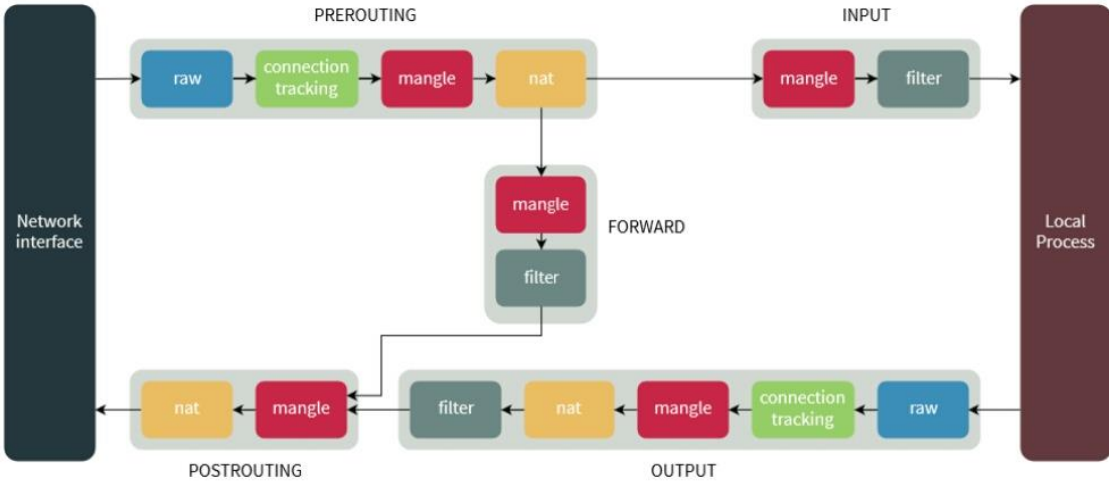
[מתוך: https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture]

**נשים לב לשוני בין DNAT (מתבצע שינוי כתובת היעד) ל-SNAT (מתבצע שינוי כתובת מקור).

כמו כן, ישנם אירועים מסוימים שיביאו לכך שחלק מהטבלאות לא יקראו, לדוגמא רק עבור החבילה הראשונה בקשר, תתבצע ניתוח של חבילה בטבלת NAT, עם שאר החבילות תתבצע אותה ההחלטה שהתקבלה בראשונה.

מתוך שיקול של גמישות, החליט יוצר iptables לאפשר למשתמש ליצור טבלאות משלו. אבל אנחנו נתקעים פה בבעיה, הלא כן? איך נגיע לטבלאות הללו בזמן ריצת המערכת? כאן מגיע פעולת jump לעזרנו המאפשרת קפיצה מהטבלאות המסודרות המוגדרות ב-iptables לטבלאות שיצר המשתמש.לאחר שקפצנו לטבלאות הללו, נוכל לחזור לטבלה הקודם לאחר שהסתיימה קריאת כל החוקים או אם נתקלנו בפקודת RETURN.

נוכל לראות שסדר ההרצה הוא כך:



[מתוך: https://www.google.com/amp/s/www.booleanworld.com/depth-guide-iptables-linux-firewall/amp]

תכונת Connection tracking

כאמור, iptables היא firewall הבודק חבילות גם מתוך המיקום שלהם בחיבור, לפי ה-contex שלהם ויחסן לחבילות הקודמות בקשר. תכונת connection tracking הנמצאת ב-netfilter framework מאפשרת ל-iptables לסווג חבילות לפי קשרים ולא רק להסתכל על החבילות כאינדיבידואליות. כאשר חבילה רק נכנסת למחסנית הרשת, מתבצע עליה בדיקת ה-connection tracking (למרות שטבלת RAW בודקת את החבילה לפני כן) והחבילה נבדקת אל מול רשימה של חיבורים שתועדו בכדי למצוא שייכות של החבילה. לאחר מכן, מעודכן סטטוס החיבור. כמובן, חבילות שסומנו כ-NO TRACK בלוגיקת טבלת RAW, ידלגו על הבדיקה של connection tracking.

מצב קשר יכול להיות אחד מהבאים:

- NEW - כאשר מגיע חבילה שלא קשורה לשום חיבור, נוצר חיבור חדש. הדבר רלוונטי הן ל-tcp והן ל-udp.
- established - כאשר עובר בחיבור מסוג NEW, חבילה בכיוון תשובה. עבור tcp מדובר בsyn+ack, עבור udp ו-icmp מדובר בחבילה שבה כתובת המקור והיעד הפוכות מאלו שבחבילה הראשונה בקשר.
- RELATED - כאשר מגיעה חבילה שאינה חלק מקשר קיים, אבל קשורה אליו (לדוגמה תשובת icmp על פרוטוקול אחר).
- INVALID - כאשר חבילה אינה חלק מקשר קיים ואינה תקינה לפתוח קשר חדש / אינה ניתנת לניתוב / אינה ניתנת לזיהוי.
- UNTRACKED - דיברנו על זה קודם, כאשר טבלת RAW יכולה לסמן חבילה במצב זה ולדלג על בדיקת ה-connection tracking.

- SNAT- סטטוס המציין כי בחבילה שהגיעה אלינו שונתה כתובת המקור. מיידעת את
  - ה-connection tracking שבהעברת החבילה מהמכונה המקומית יש לשנות כתובת המקור.
- DNAT -סטטוס המציין כי בחבילה שהגיעה אלינו שונתה כתובת היעד. מיידעת את
  - ה-connection tracking שבהעברת החבילה מהמכונה המקומית יש לשנות טת כתובת היעד בחבילה.

היכולת של ה-connection tracking מאפשרת למשתמש לייצר חוקים המתייחסים לסטטוס החיבור.

קיימים מודולים שונים היכולים להצטרף ל-iptables, אחד מהם הוא connection tracking, אחרים נוספים הם tcp המשמש לבדיקת דגלי tcp בחבילות, limit לצורך הגבלת מספר החבילות העוברות, owner המשמש לחסימת חבילות פר משתמש ומארח ועוד. טעינת המודול בעזרת דגל m, לדוגמא:

```
iptables -A INPUT -p tcp -m tcp --tcp-flags ALL FIN,PSH,URG -j DROP
```

כאן, נזרקות חבילות tcp המכילות את כלל הדגלים.

קמצוץ של אבטחה לעסק

כעת, אחרי שהבנו כיצד להשתמש ב-iptables (מאמר הקודם) ומה עומד מאחורי iptables, אני מעוניין לתת כמה עצות להגן על המחשב שלנו בבית על בסיס iptables המבוססות על best practices. חשוב לציין שיכול להיות שחלק מהדרישות לא יתאימו לשימושים מיוחדים שאתם עושים במחשב, אבל הדרישות עצמן, נועדו להיות גנריות ולהתאים לרוב המוחלט של המשתמשים. כמו כן, חשוב לי לציין שהגנה על מחשב היא מורכבת מאוד ואינה בנויה על רכיב אחד בלבד ולעיתים, גם הממשקים שבין הכלים השונים יכולים להיות קריטיים למערכת. אז בואו נתחיל:

- לכל טבלה ישנה מדיניות ברירת מחדל, המדיניות **אינה** החוק האחרון שיש בטבלה אלא היא כלל חיצוני. ההמלצה היא שלא לשים מדיניות דיפולטיבית כלהיא ב-DROP. מדוע? אם בטעות מנהל רשת אחר ימחק את כלל חוקי הטבלה, או אם לא שמרתם את החוקים, אתם עשויים להביא לכך שתמיד תזרקו את כלל החבילות המגיעות. לכן, ההמלצה היא לשים את מדיניות ברירת המחדל ב-ACCEPT ובחוק הכי אחרון בטבלה, לשים באופן מפורש חוק של DROP אם לא נמצאו התאמות.

- אם יצרתם באחת הטבלאות קבוצת חוקים שעשויה להיות מסובכת, שיקלו להעביר את החוקים לשרשרת חדשה משלכם ושימו JUMP אל מיקום זה. הדבר עשוי לעזור לכם לשמור על סדר ולנהל טת הטבלה בקלות.

- נסו להיות כמה שיותר ספציפיים בחוקים שלכם בכדי לא לאפשר חבילה שלא רציתם לאפשר. לדוגמא, בכדי לאפשר כניסת ping-ים למכונה, אני אעדיף את החוק הבא, שמאפשר icmp מסוג 8 בלבד (פינג):

```
iptables -A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
```

על פני החוק הבא שמאפשר כל icmp:

```
iptables -A INPUT -p icmp -m icmp -j ACCEPT
```

- זכרו תמיד לשמור את החוקים שלכם, אין פה שמירה אוטומטית לאחר כיבוי המחשב!
- כמו שחשוב לדעת ולתעד מי נכנס למדינה בכל רגע, ככה חשוב לתעד את החבילות המתאימות לחוק מסוים, זה יעזור לתחקורים מתקדמים יותר במחשב...
- מדיניות DROP היא תמיד מומלצת בסיום טבלאות.
- שימו חוקים ספציפיים יותר בראש הטבלה וחוקים גנרים יותר בראשה.
- אפשרו למשתמשים את מינימום השירותים הנדרשים לעבודתם (שירותים כמו http ו-dns הם הכרחיים לעבודת המערכת), אבל האם לעבודת המערכת, אתם רוצים לאפשר לדוגמא שירותים כמו telenet ו-ssh? ומה לגבי smb פנימה והחוצה?
- חסמו את מספר הקישורים שמחשב כלשהו יכול לבצע אליכם כדי למנוע DOS, לדוגמא בעזרת מודול של multiports. מראה את החוק הבא המאפשר להגביל את מספר החיבורים החדשים ל-20 עבור https ו-http:

```
iptables -A INPUT -p tcp -syn -m multiport --dport http,https -m
connlimit --connlimit-above 20 -j REJECT
```

- הפקטה לא טובה להעברה הלאה? העדיפו לעשות לה DROP ולא REJECT. בעזרת REJECT, אנו מדווחים לצד השני שהחבילה נפלה. נעדיף להשתמש ב-DROP בכדי להימנע מעיבוד נוסף למערכת, אשר עשוי לעדכן גם גורם שעשוי להיות תוקף למערכת.

סיכום

במהלך המאמר הצגתי כיצד עובד הכלי של iptables מאחורי הקלעים, הצגתי את המנגנונים שבהם iptables משתמש אל מול מערכת ההפעלה ומדוע. דיברנו קצת על hook-ים ו-NAT והקשר שלהם

ל-iptables. לסיום, הבאתי כמה best practices לעבודה עם iptables כמנהלים רשתות וכאנשים פרטיים. מקווה שעכשיו הכלי הקטן הזה שנמצא על כל מחשב, מובן לכם יותר. אז לסיום,

Be ethnical, be safe and dont be a script kiddies

על הכותב

עדי מליאנקר, 22 חוקר אבטחה עצמאי, מתעסק ב-bug bounty וייעוץ לחברות. כותב חיצוני בבלוג אבטחה בעניינים הפנימיים של מערכת ההפעלה windows (תבוא לבקר!).

זמין לשאלות, הערות, ויותר מאשמח לתוספות לדברים ששכחתי במאמר או סתם memes במייל (temporarynothing0gmail.com).

ביבליוגרפיה

- https://www.jensd.be/343/linux/forward-a-tcp-port-to-another-ip-or-port-using-nat-with-iptables
- https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture
- https://www.booleanworld.com/depth-guide-iptables-linux-firewall/amp/
- https://major.io/2010/04/12/best-practices-iptables/
- https://opensource.com/article/18/10/iptables-tips-and-tricks
- https://www.lammertbies.nl/comm/info/iptables.html&ved=2ahUKEwiSj9bhzOviAhVOZJoKHcU4Ad0QFjADegQIARAB&usg=AOvVaw3xy_wCJ2mEIxoKTUU1OJC5

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-108 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין - Digital Whisper צרו קשר!

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il.

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא בסוף חודש יולי 2019.

אפיק קסטיאל,

ניר אדר,

30.06.2019