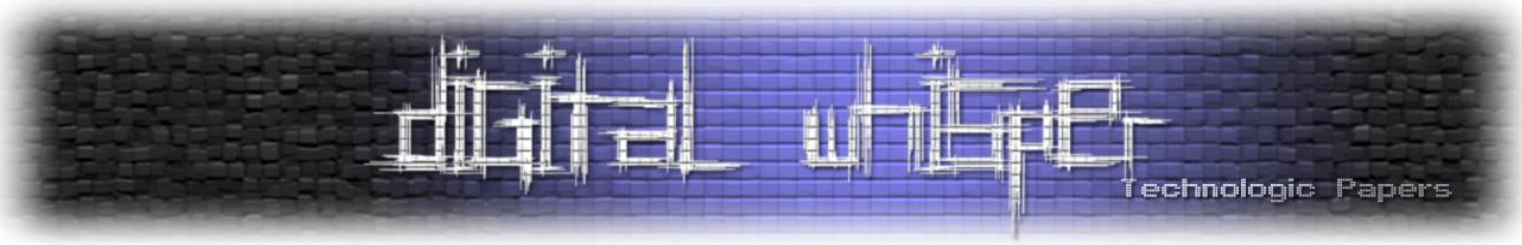


Digital Whisper

גליון 187, יולי 2026

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל וספיר פדרובסקי
כתבים:	שי מרדכי, גיא תותחני, ניר אברהם, סופי ציאדה, דור נאמני וגיא ברנהרט מגן.

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

דבר העורך

ברוכים הבאים לדברי הפתיחה של הגליון ה-187 של DigitalWhisper!

לא מעט נכתב בחודשים האחרונים על הפרסום של חוקרי Google, שבו הציגו שיפור משמעותי בהערכת המשאבים הנדרשים לשבירת הצפנות א-סימטריות, ובין היתר כאלה המבוססות על עקומים אליפטיים (ECC) באמצעות מחשוב קוונטי. הנושא הזה מרתק, ואף יש לנו מאמר עליו בגליון הנוכחי. אבל מה שמשך את תשומת ליבי, דווקא לא היה המחקר עצמו, אלא הדרך שבה בחרו החוקרים לפרסם את תוצאותיו.

למי שמעוניין לעשות הפסקת קריאה קלילה, הינה ה-White Paper ש-Google Quantum AI פרסמו.

לא קראתי את המאמר (ה-"קריאה קלילה" הייתה בדיחה...), אבל כן צפיתי בראיון של שעה ו-40 עם דן בונה (אחד מכותבי המאמר ואחד הקריפטוגרפיים היותר פורים שחי כיום) על המאמר, ובו הוא מספר על הרקע למחקר, על האופטימיזציות לאלגוריתם, ועל המשמעויות שלו. הראיון מרתק, ממליץ לכולם.

הדרך שהחבר'ה מ-Quantum AI בחרו לפרסם את תוצאות המחקר, הייתה (לאחר התייעצות עם גורמים מדיניים) שלא לחשוף את האופטימיזציות שלהם לאלגוריתם Shor (לא לפרסם את המעגל הקוונטי שבו הם השתמשו), אלא לפרסם במקומו הוכחת אפס-ידע (Zero-Knowledge Proof) לעצם קיום המעגל ברשותם. במילים אחרות, הם אפשרו לעולם להשתכנע שברשותם מעגל שמהווה אופטימיזציה לאלגוריתם Shor, שמאפשרת להם לשבור הצפנות מבוססות עקומים אליפטיים בעזרת כמות משאבים קוונטים הנמוכה באופן משמעותי ממה שהאנושות חשבה שצריך עד כה, וזאת מבלי לחשוף שום נתון על המעגל או על הדרך שבה הם הגיעו אליו.

במבט ראשון, זאת נשמעת כמו פשרה מצוינת. מצד אחד, הקהילה המדעית וקהילת אבטחת המידע מקבלת הוכחה לתוצאות המחקר שניתן לסמוך עליה, ובכך לאפשר לכולם לדעת איך להתנהל נכון כלכלית ולהתחיל להקצות משאבים לנושא כבר בעתיד הקרוב. אך מצד שני, פרסום שכזה לא מספק מספיק פרטים שיוכלו לאפשר למעצמות, שבעתיד הקרוב יהיו נגישות למחשבים קוונטים להתכונן לרגע שבו אכן יהיו בידיים מספיק קיוביטים, ולנצל את מרווח הזמן שבין הפרסום ועד עדכון המערכות בפועל.

אבל אז קרה משהו מעניין. תוך זמן קצר מאוד מאז פרסום ה-ZKP של צוות המחקר בגוגל, החלו להופיע באינטרנט מחקרים נוספים ושילובי כוחות שניסו להגיע לאותו המעגל. חלקם אף הציגו פתרונות יעילים יותר, לפחות על פי חלק מהמדדים. ופתאום העולם מצא את עצמו עם פתרונות טובים יותר ממה שגוגל ניסו לא לפרסם, אצל גורמים שאין לאף אחד באמת שליטה עליהם.

וזה גורם לתהות: האם הפרסום שנעשה מתמטית באפס ידע אכן היה באפס ידע? וזה מוביל לשאלה יותר משמעותית: האם באמת אפשר לפרסם היום משהו מבלי לחשוף אותו? ועזבו אתכם מההגדרה

המתמטית, כי הרי גם אם לא פרסמתם את הפתרון, עדיין פרסמתם את העובדה שיש פתרון. ובמקרים רבים, אולי זה כל מה שהקהילה צריכה כדי לרכז משאבים בכיוון ספציפי.

המחשבה הזו הזכירה לי תחום אחר שאנחנו מכירים היטב וגם שם יש דילמה: מה, כמה ומתי לפרסם. ה-Responsible Disclosure על חולשות שנמצאות במוצרים השונים.

במשך שנים התגבשו בתעשייה כללים לא כתובים: יש חברות שמפרסמות רק CVE ותיאור כללי. יש כאלה שמסתירות את פרטי הבאג עד שרוב המשתמשים יעדכנו. אחרות מעכבות את פרסום ה-PoC בעוד מספר שבועות או חודשים. כל אחת בוחרת איזון מעט שונה בין שקיפות לאחריות.

אבל כולן נשענות על אותה הנחת יסוד: היסוד שאומר שקיים פער משמעותי בין רמז לבין פתרון מלא.

הנחת היסוד הזאת אומרת שאם אכתוב שמצאתי חולשת Use-After-Free במנוע JavaScript ספציפי, או שפגיעות במנגנון אימות ספציפי התגלתה ותתוקן בקרוב - עדיין יידרשו שעות, ימים או שבועות של עבודה כדי שחוקרים נוספים יצליחו למצוא אותו ולהבין מה בדיוק מצאתי ואיך ניתן לשחזר את המתקפה.

והשאלה היא האם ההנחה הזו עדיין מחזיקה.

לא בגלל שחוקרים נהיו טובים יותר בן לילה, אלא בגלל שכלי העבודה נהיו עוצמתיים יותר, והעלות של לחקור כיוון מסויים הפכה להיות הרבה יותר זולה מעד לפני שנים בודדות.

גוגל למשל, נוקטת לעיתים בגישה מדורגת: תחילה הודעה כללית כמו זאת:
Note: Access to bug details and links may be kept restricted until a majority of users are updated with a fix. We will also retain restrictions if the bug exists in a third party library that other projects similarly depend on, but haven't yet fixed.

ורק בהמשך, פרסום מלא של פרטי החולשה. זאת מתוך מחשבה שחלון הזמן הזה עדיין מייצר יתרון למגינים. אבל עושה רושם שגם ההבחנה הזו מתחילה להיסדק.

היום כבר לא מופרך לדמיין Agent שמקבל Advisory קצר, מבצע Patch Diff, מתמקד ברכיב שהשתנה, ומפיק כיווני חקירה תוך דקות. הוא לא בהכרח מגיע לפתרון מלא - אבל מספיק כדי לצמצם משמעותית את המרחק בין רמז להבנה. למעשה, יש כבר כמה חברות שטוענות שיש להן מוצר כזה. זאת לדוגמא.

ויש גם כיוון נוסף, כמעט הפוך: פרסום עדכונים מבלי לפרט מה בדיוק תוקן. לא מתוך רצון להסתיר מידע מהקהילה, אלא מתוך ניסיון לא למשוך תשומת לב ממוקדת לנקודת החולשה לפני שהעדכון הופך לנפוץ מספיק. סוג של "עמעום זמני" של משמעות התיקון עצמו.

ופתאום גם הבחירה של Google נראית באור שונה. ה-ZKP באמת הסתיר את הפתרון. אבל האם הוא הסתיר מספיק את כיוון החיפוש? אין לי תשובה חד משמעית. למעשה, אני לא בטוח שלמישהו יש.

אבל ייתכן שבעשור הקרוב נצטרך להגדיר מחדש מושג שליווה את קהילת אבטחת המידע במשך עשרות שנים. אולי "פרסום אחראי" כבר לא יהיה רק השאלה מתי לפרסם או כמה לפרסם.

אולי השאלה תהיה הרבה יותר בסיסית: האם עצם הידיעה שיש מה לחפש - היא כבר סוג של פרסום?

ושיהיה בהצלחה לכולנו!

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

תודה רבה ל**שי מרדכי**, תודה רבה ל**גיא תותחני**, תודה רבה ל**ניר אברהם**, תודה רבה ל**סופי ציאדה**, תודה רבה ל**דור נאמני** ותודה רבה ל**גיא ברנהרט מגן**.

קריאה נעימה,

ספיר פדרובסקי ואפיק קסטיאל

תוכן עניינים

תקציר מנהלים

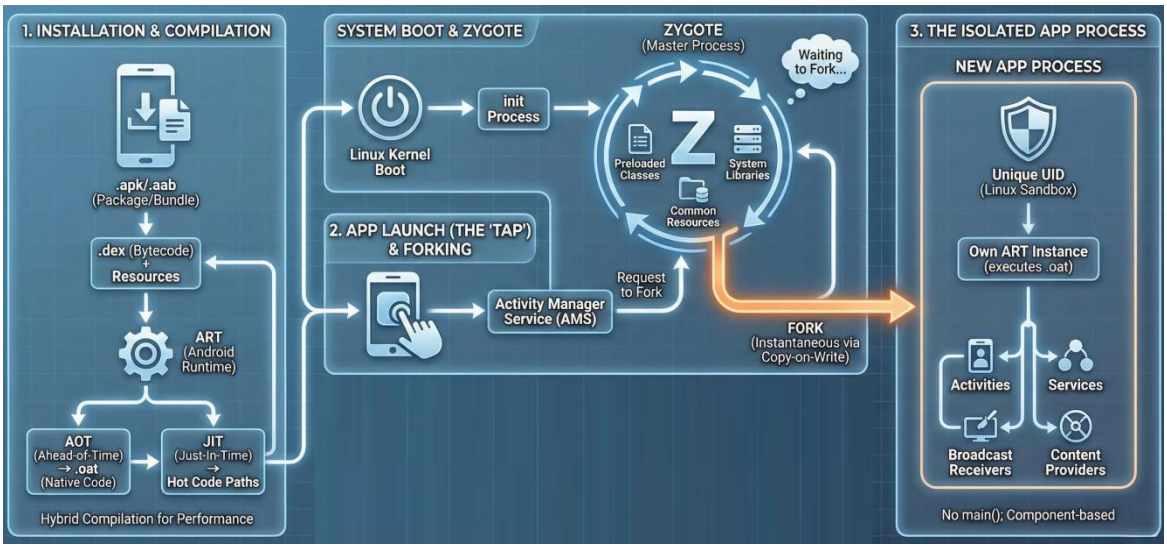
הרבה מאוד משתמשי אנדרואיד (ואפילו מפתחים) לא מודעים לנתון הבא:

"עולם האנדרואיד" כפי שאנחנו מכירים אותו, לא באמת קיים ברגע שהמכשיר נדלק. בשניות הראשונות, הטלפון שלכם פשוט פלטפורמת Linux רגילה לחלוטין.

המאמר מתמקד בשבריר השניה לאחר ריצת האפליקציה על מכשיר האנדרואיד, וכיצד נוזקות מתקדמות מנסות "לעצור את הזמן" ולהידמות למערכת ההפעלה בעצמה, עוד לפני שהמשתמש רואה מסך כלשהו.

במאמר זה, ננתח חלק מהנוזקה Anatsa - כלי תקיפה ממשפחת ה-Banking Trojans, המיועד לגניבת פרטי גישה לחשבונות פיננסיים. ב-2024 היא הגיעה ל-100,000 הורדות ב-Google Play עד שהסירו אותה. הנוזקה הזו, מבצעת עקיפה לא שיגרתית של הצגת ה-UI של האפליקציה. בחרתי לנתח אותה, כמקרה בוחן ל-Framework שפיתחתי למידול התנהגות נוזקה לפי 5 שכבות, ו-Anatsa פשוט ביצעה כל אחת מהן.

התרשים הבא מתאר את תהליך יצירת אפליקציה בשלבים - החל מהתקנה עד יצירת התהליך הלינוקסי:



[מקור: Krithika Ravishankar - BOOTING OF THE ANDROID PROCESS]

המירוץ מתחיל

כאשר משתמש מריץ אפליקציה לגיטימית, עץ התהליכים נראה כך:

The Processes Map Hierarchy

Plaintext

```
Linux Kernel
   |
   └── init (PID 1)
       |
       ....
       |
       └── Zygote
            |
            ├── System Server (Zygote Fork)
            |      ├── ActivityManagerService (AMS)
            |      └── PackageManagerService (PMS)
            |
            |
            └── App Process (Zygote Fork)
                   |
                   └── ActivityThread
                         └── Main / UI Thread
                                └── Looper
```

ה-Zygote הוא בעצם טמפלט נקי עבור כל אפליקציית אנדרואיד. כדי להכין את הקרקע, תהליך ה-Zygote מאתחל שלושה רכיבים בסיסיים:

- ה-Android Runtime: ה-ART הינו "המכונה הוירטואלית", המסוגלת לקרוא ולהריץ קוד Java או Kotlin (קבצי ה-DEX).
- ה-Android Framework: טעינה מראש של אלפי מחלקות ומשאבי מערכת אל הזיכרון.
- סוקט מקומי: ערוץ תקשורת שמאזין לבקשות ()fork. ה-Zygote יוצר אותו לפני שעדיין ה-System Server נולד.

הפיצול

במהלך ה-Boot, מתבצע ()fork ראשון שיוזם ה-Zygote. נוצר תהליך בן שירש את ה-ART, את ה-Framework, ובלית ברירה, גם את הסוקט.

מתכנתי הקוד של אנדרואיד, הזדרזו לסגור את הסוקט מטעמי אבטחה, כפי שניתן לראות בקטע הקוד הבא:

```
Zygote.closeAllFilesExcept(managedFileDescriptors);
```

ה-Fork "נולד" עם מחרוזת בזיכרון, שתשמש אותו כדי להריץ את ה"ייעוד" שלו בעזרת פונקציות בזמן ריצה (ע"י מנגנון Reflection).

מי שאחראי לכך שה-Fork הראשון ייווצר עם המחרוזת ()SystemServer.main, זהו ה-Init (בכבודו ובעצמו). הוא מעביר אותה כחלק מיצירת ה-Zygote עם דגל start-system-server-- חד-פעמי (לחיסכון בזמן). כך ה-Zygote מחזיק בזיכרון את המחרוזת ומיד משכפל את עצמו.

בהמשך, כאשר ה-System Server ירוץ ויגיד ל-Zygote שצריך Fork חדש, הוא זה שיקבל את המחרוזת מה-Launcher, ויעביר אותה ל-Zygote דרך Local Socket.

מעתה והלאה, ה-Zygote עובר למצב האזנה בסוקט שפתח. מי שאחראי "להעיר" אותו וליזום קריאות של Zygote Forks, זה רק המנהל הראשי - System Server.

חלוקת תפקידים

תהליך ה-System Server

הפונקציה SystemServer.main() היא זו ש"מייצגת" את מערכת ההפעלה, ומנהלת את תהליך האפליקציה. שניים מהשירותים המרכזיים שחשוב להכיר הם:

1. Activity Manager Service - ה-AMS, הוא ה"מבוגר האחראי" שמנהל את הרכיב המנוהל - ActivityThread שבתוך האפליקציה.
2. Package Manager Service - ה-PMS, אחראי לנהל ולשמור את המידע על כל האפליקציות המותקנות במכשיר.

ברגע שמתעורר ה-System Server במהלך ה-Boot, דבר ראשון שהוא עושה זה להריץ את ה-main שלו. ה-PMS סורק את כל קבצי ה-Manifests שיש במכשיר (בפועל, סורק קובץ כללי אחד - כחלק מאופטימיזציה), ומעדכן טבלה פנימית שמגדירה הרשאות ושירותים לכל אפליקציה.

מאוחר יותר, ה-Launcher (מסך הבית) רק "מתשאל" את ה-PMS, כדי לקבל את האייקון והשם (ה-Label) שלה, כדי להציג אותם במסך הבית (כפי שנראה בחלק ג', בקוד ה-XML הראשון).

תהליך אפליקציה

נוצר בכל פעם שהמשתמש לוחץ על ה-Launcher של האפליקציה.

ה-System Server שולח פקודה ל-Zygote דרך ה-Socket: "תשכפל פרוסס חדש". התהליך החדש שנוצר, משתמש ב-Reflection כדי להריץ את ActivityThread.main(), שמפעיל את ה-Looper (נראה בהמשך) ומכין את ה-UI Thread לקבלת הודעות מה-System Server.

ברגע שהאפליקציה "חיה ונושמת", ה-Zygote מסיים את תפקידו וחוזר "לישון". מעתה והלאה, התקשורת השוטפת מתבצעת ישירות בין ה-ActivityThread לבין ה-System Server (ה-AMS), באמצעות מנגנון העברת הודעות מאובטח (IPC) הראשי של אנדרואיד - Binder IPC.

תהליך ה-ActivityThread

זהו "סוכן" שיושב בתהליך האפליקציה, ומקבל את הפקודות מ-AMS. למרות שמו, הוא אינו Thread, אלא מחלקת Java שמנהלת את האפליקציה מבפנים. על מנת לתקשר עם החוץ, הוא מחזיק בתוכו מחלקה פנימית שמאזינה לבקשות מ-AMS. בקשות אלו מועברות לתור של ה-UI Thread.

מנגנוני ה-Socket וה-Binder

נמצאנו למדים, שיש ל-System Server כ-3 תפקידים עיקריים:

- לבקש בעצמו Zygote Forks עבור אפליקציות חדשות.
- לתקשר עם האפליקציה שנוצרה באמצעות ה-AMS.
- לסרוק את קובץ ה-AndroidManifest.xml במהלך ה-Boot או התקנת האפליקציה באמצעות ה-PMS.

אך כאן עולה שאלה מעניינת: למה המערכת צריכה שני מנגנוני תקשורת (IPC) שונים? בדומה למודלים של IPC בין דפדפן ליישומים בלינוקס, גם כאן התקשורת לא מתבצעת ישירות.

- בשלב ההקמה: שלב שבו עוברת התקשורת דרך ה-Local Socket, בין ה-System Server לבין ה-Zygote. זהו צינור מהיר ופשוט שנועד למטרה אחת בלבד - להעיר את ה-Zygote ולהגיד לו לבצע שכפול.
- בשלב הריצה: שלב שבו עוברת התקשורת דרך ה-Binder, בין ה-AMS לבין ה-ActivityThread.

מנוע החיים של התהליך: ה-Looper של ה-UI Thread

(חשוב להבהיר: ארכיטקטורת אנדרואיד רוויה בלולאות Looper שונות, ולמעשה כמעט לכל Service).

בהקשר של "לידת" אפליקציה, לאחר ה-Fork והרצת הפונקציה הייעודית מ-Framework, יש רק Thread אחד, ה-UI Thread. בפועל, כל מה שאנחנו קוראים לו 'האפליקציה', יושב על לולאה אחת (Looper) שחיה בתוך ה-UI Thread, שמנוהלת על ידי ה-ActivityThread. היא משהה את ה-UI Thread מלהסתיים, וממתינה לפקודות ה-System Server כדי להריץ את קוד האפליקציה.

כבר הזכרנו שה-System Server כבר מחזיק בזיכרון שלו הרשאות, וביניהן את ה-Entry Point של האפליקציה. בזמן הריצה, ה-System Server שולח פקודות הרצה (כמו onCreate, onStart וכו'), כדי להניע את ה-Entry Point. הפקודות האלו נשלחות מה-System Server דרך ה-Binder IPC, ומתורגמות להודעות בתוך תור ההודעות של האפליקציה.

ה-UI Thread מצידו לא "נחנק" - הוא יושב בתוך לולאת ה-Looper האינסופית, שולף את הפקודות האלו מהתור אחת אחת בעזרת המתווך שלו, ומריץ את פונקציות האפליקציה בזו אחר זו.

נקודת המחטף

עד כאן, זה מה שהבנתי מחקירת OWASP Uncrackable Level 2. זה כבר הספיק כדי "לשרוף לי את המוח" בקטע טוב. אך מסתבר שזו הייתה רק ההתחלה. שם, קובץ ה-AndroidManifest.xml עבד "ממש לפי הספר". כשהגעתי לחקור את הנוזקה Anatsa, ראיתי שה-Manifest שלה נראה אחרת לגמרי.

שתי נקודות הכניסה - Application vs. Activity

באנדרואיד אין רק "נקודת כניסה" אחת. יש למעשה שתי רמות שונות של הרצת קוד שרצות זו אחר זו האחת ברמת ה"תהליך", והשניה ברמת ה"חלון."

הזכרנו שה-System Server אחראי לשלוח פקודות ל-Looper. יש פקודה אחת שמתרחשת בתחילת הרצת האפליקציה, והיא מושפעת ממה שכתוב (או לא כתוב) ב-Manifest. הפקודה הראשונה של ה-AMS נקראת bindApplication. היא הטריגר שגורם למערכת ההפעלה ליצור את אובייקט ה-Application (כפי שנראה בהמשך), והיא מתרחשת עוד לפני שהאפליקציה מריצה UI כלשהו.

ניתוח AndroidManifest.xml של Level 2

```
<application
    android:theme="@style/AppTheme"
    android:label="@string/app_name"
    android:icon="@mipmap/ic_launcher"
    android:allowBackup="true"
    android:supportsRtl="true">

    <activity android:name="sg.vantagepoint.uncrackable2.MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
</application>
```

המאפיין android:name היא נתון שמשמש ליצירת Entry Point. הפעם הוא קיים רק ברמת ה"חלון". אין כאן מאפיין ששייך לשלב ה-bindApplication, לכן ה-Application יהיה ברירת מחדל, ואליו יתווספו מאפיינים גלובליים לאפליקציה.

ניתוח ה-AndroidManifest.xml של הנוזקה

```
<application
    android:label="PDF Reader: Update"
    android:icon="@vista6/vista_res_0x7f060000"
    android:name="ihp.opjhprs.fsthjpoyr.rhhkkff"
    android:allowBackup="true"
    android:usesCleartextTraffic="true">
```

כאן יש תכונה שמשפיעה מאוד על שלב ה-bindApplication. התכונהandroid:name גורמת למערכת ההפעלה ליצור אובייקט Application מותאם אישית, ולא את אובייקט ברירת המחדל שראינו ב-Uncrackable. בעצם, הוגדר כאן Entry Point מוקדם מאוד ברמת ה-Process, עוד לפני ה-Entry Point של ה-UI.

אם הקוד בשלב זה יכיל פעולות ארוכות, ה-UI Thread ייתקע, והמשתמש יראה מסך שחור וימחק את האפליקציה. לכן, הקוד ב-Entry Point הזה חייב להיות קצר וקטלני.

להלן חלק מקוד ה-Entry Point הידני של הנוזקה:

```
/* JADX INFO: compiled from: StubApp.java */
/* JADX INFO: loaded from: classes.dex */
public class rhhkkff extends Application {
    @Override // android.content.ContextWrapper
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);
        new eomqmgsskufu(base).oqwrkywwlykpu();
        new tttlkqttg(base).mopksfpffv();
        kjyylrrmxujekkg.wqlx(this, qhpsx.noxgiqtlgxejjlkguiqsl,
qhpsx.pmoerwxrnviwturjgmsrs);
    }
}
```

הניתוח הסטטי כאן מוגבל, כי הקוד עבר Obfuscation רציני. מה שכן אפשר לראות כאן מיד, שהקלאס הזה הוא מסוג Stub Unpacking (לפי השם בשורה הראשונה של ה-Info).

ה-Stub Unpacking עובד ב-3 שלבים מרכזיים:

1. טעינת קובץ המקור לזיכרון.
2. שילוב המחלקות הזדוניות אל תוך סביבת הריצה של האפליקציה.
3. העברת השליטה ל-Entry Point של התוכנית המקורית.

הקוד שלעיל מבצע בדיוק את השלבים האלה, אותם נשאיר לשלב ה-Dynamic Analysis עם Frida.

וידוי

ברגע הראשון שהסתכלתי על הקוד, לא הבנתי מי מריץ אותו. הוא נראה כמו 4 פונקציות "באוויר", בלי שאף פונקציית main קוראת להן!

גיליתי שבאנדרואיד, ה-ART עובדת הפוך משאר סביבות הרצת הקוד שאנחנו מכירים. אפילו מה-JVM בעצמו, אני חייב לקרוא בעצמי לפונקציה שכתבתי כדי שהיא תתחיל לרוץ. באנדרואיד לעומת זאת, כבר יש מי שקורא ל-Entry Point - אובייקט ה-Application או ה-Activity, וזה מתבצע בשלב שכבר ראינו - יצירת אובייקט ה-Application בעזרת פקודת bindApplication ששולח ה-AMS.

הדוגמה הבאה מקוד ה-Android Framework:

```
public Application newApplication(ClassLoader cl, String className, Context
context) {
    return (Application) cl.loadClass(className).newInstance();
}
// ------------------------------------------------------------------
public Activity newActivity(ClassLoader cl, String className, Intent intent) {
    return (Activity) cl.loadClass(className).newInstance();
}
```

הנוזקה הגדירה קלאס מותאם אישית בתור Entry Point ידנית, ולכן הוא יתלבש ישירות על אובייקט ה-Application.

כעת אפשר להבין שהקלאס שהוגדר כ-Process Entry Point באובייקט ה-Application, בעצם מורץ ברקע, וכל זה באמצעות קוד מערכת ההפעלה שמשתמש במנגנון Reflection.

הדגמת "הלבשה" ידנית של פונקציות על גבי אובייקט ה-Application בתהליך אתחול ה-Process:

- Application.attachBaseContext(...)
- Application.onCreate()
- Application.createPackageContext(...)
- Application.getPackageName()

אתגר הניתוח הדינמי

לאחר שהבנו שהנוזקה חוטפת את שלב ה-Bootstrap ורצה בתוך attachBaseContext(), עולה שאלה קריטית: איך בכלל אפשר לנתח אותה בזמן אמת? כאן נכנס לתמונה כלי ה-Dynamic Binary Instrumentation המוביל בתעשייה - Frida.

Frida הוא כלי המאפשר לחוקרים להזרים סקריפטים מותאמים אישית של JavaScript ישירות לתוך הזיכרון של תהליכים חיים, ובכך לבצע הזרקת סקריפט (Hooking) לפונקציות ולשנות את התנהגותן בזמן ריצה. הבעיה היא הארכיטקטורה של הנוזקה, שדורשת מאיתנו לשנות את הדרך שבה אנחנו משתמשים ב-Frida.

1. **הגישה הרגילה והמכשול שלה: חיבור מאוחר (Attach)**

במצב הרגיל (Attach), אנחנו מנסים להתחבר לאפליקציה אחרי שהיא כבר הופיעה על המסך (בשלב ה-UI). ההתחברות מתבצעת הלכה למעשה רק אחרי שהתחיל ה-Entry Point עם אובייקט ה-Activity.

בשלב זה, Frida שולחת פקודה לקרנל: "עצור את ה-Threads של האפליקציה", מזריקה את סוכן הניטור שלה (ה-Frida Agent), ומשחררת את הריצה. לרוע המזל, זה כבר מאוחר מדי. מתודת ה-attachBaseContext() של הנוזקה כבר רצה והסתיימה מזמן, יחד עם אובייקט ה-Application.

2. **הדרך לנצח את המירוץ: מצב Spawn**

מכיוון שאין לנו שום יכולת אנושית (או טכנית במצב Attach) להיות מהירים יותר מפקודות ה-Bootstrap של ה-System Server, הדרך לנצח היא להקדים את טעינת האפליקציה בעזרת מצב Spawn.

במצב Spawn, האפליקציה מופעלת כרגיל (מה שגורם ל-System Server לבקש Fork מתוך ה-Zygote), אך Frida מתערבת מיד לאחר ה-Fork. היא תופסת את התהליך החדש ומעבירה אותו למצב השהייה, עוד לפני שהפקודה הראשונה של ה-ActivityThread מתחילה לרוץ (לפני ה-Process Entry Point). כך, כשהנוזקה תגיע לבסוף אל ה-attachBaseContext(), אנחנו כבר נהיה שם בפנים כדי לתפוס אותה בזמן אמת.

3. **שבירת חוקי המשחק: Zygote Hook**

במקרים קיצוניים של נוזקות מתקדמות (או כאשר קיימות הגנות Anti-Analysis חזקות במיוחד), ניתן להשתמש בשיטה האגרסיבית ביותר: ה-Zygote Hook. במקום להמתין ליצירת תהליך האפליקציה הספציפי, ה-Frida מזריקה את עצמה ישירות לתוך תהליך ה-Zygote המקורי עצמו. המשמעות היא שכל אפליקציה עתידית שתרוץ במכשיר, תשתכפל ותיוולד כאשר ה-Frida Agent כבר נמצא ב-"DNA" שלה, עוד לפני שלב ה-Fork.

ה-Hook במילים פשוטות:

נניח שההוק שלי מתחיל לרוץ מיד אחרי ה-Fork, מה הוא רואה?

- שעוד מעט מתחילה סריקת ה-Manifest.
- הולך להיווצר אובייקט Application - ושם בדיוק נעצור אותו. נתפוס את מי שקרא ליצירת Application חדש, נכריח אותו ליצור אובייקט נקי, ונחזיר אותו ללא תלות בקוד הזדוני של האפליקציה.

כדי להבין בדיוק מי יוצרת אובייקט חדש, חקרתי את הדרך שבה ה-ActivityThread בונה את האפליקציה, וגיליתי שהוא משתמש במחלקה פנימית בשם LoadedApk. זיהיתי שהפונקציה makeApplicationInner (בתוך מחלקה זו) היא זו שקוראת פיזית ליצירת האובייקט (newApplication).

היא מבצעת בדיקה פנימית על הפרמטר הראשון הבוליאני:

```
if (forceDefaultAppClass || (appClass == null)) {
    appClass = "android.app.Application";
}
```

אם הפרמטר הוא false ואין appClass מוגדר (כלומר, אף אחד לא הכניס מאפיין android:name ל-Application בתוך ה-Manifest), ייווצר אובייקט Application ברירת מחדל. אך בגלל שהנוזקה כן קיימה והגדירה appClass זדוני, אנחנו נכריח את המשתנה forceDefaultAppClass להיות true בזמן ריצה, ומיד ייווצר אובייקט appClass חדש ונקי לחלוטין:

```
Java.perform(function() {
    var reset_entry_point = Java.use("android.app.LoadedApk");

    reset_entry_point.makeApplicationInner.implementation =
function(forceDefaultAppClass, instrumentation, allowDuplicateInstances) {
        // Force creation of a clean, default Application class.
        forceDefaultAppClass = true;
                return this.makeApplicationInner(forceDefaultAppClass, instrumentation,
allowDuplicateInstances);
    };
});
```

לאחר תיקון קל לתמיכה ב-overload כשיש מספר רב של פונקציות עם אותו השם, ה-Hook הנקודתי הצליח, (אך הוא צפוי להיכשל בהמשך, כשירוצו הבדיקות של הנוזקה בשלב ה-Activity Bootstrap ואז נעדיף אובייקט מותאם אישית):

```
[shay0129@mobile-lab Anatsa_2024]$ frida -U -f com.clnvyokcv.bgvpowkic -l my_hook.js
     ____
    / _  |   Frida 17.9.1 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Android Emulator 5554 (id=emulator-5554)
Spawned `com.clnvyokcv.bgvpowkic`. Resuming main thread!
[Android Emulator 5554::com.clnvyokcv.bgvpowkic ]->
[+] ==========================================
[+] [Bootstrap Hijack Detected]
[*] Original forceDefaultAppClass value: false
[!] Modifying runtime environment: forcing forceDefaultAppClass = true
[+] [PASS] Bypassing malicious entry point successfully!
[+] ==========================================
```

עד שגילינו ש-Application Bootstrap זה ה-Entry Point המוקדם ביותר של אפליקציית אנדרואיד - גיליתי שיש עוד Bootstrap נסתר, אז אל תלכו לשום מקום. להלן השגיאה מ-Frida:

```
Process crashed: java.lang.ClassNotFoundException: Didn't find class
"com.clnvyokcv.bgvpowkic.util.SomeProvider
```

השגיאה הזו הזכירה לי מידע שראיתי ב-VirusTotal, שה-Provider של הנוזקה הוא SomeProvider (למרות שהשם כבר מרמז לבד). הוא מופיע גם ב-Manifest:

```
<provider
        android:name="com.clnvyokcv.bgvpowkic.util.SomeProvider"
        android:exported="true"
        android:authorities="com.transportation.ukltd"
        android:grantUriPermissions="true"/>
```

המשמעות היא, שיש קוד שרץ קודם שנוצר ה-Default Application!

Provider

זהו פיצ'ר שאנדרואיד הכניסה, כדי לייעל את העבודה לספריות צד שלישי - לאפשר טעינה אוטומטית של קוד שקודם לקוד האפליקציה. נוזקות (ובעיקר Packers), משתמשות בו כדי לטעון את ה-DEX המוצפן, עוד לפני יצירת אובייקט ה-Application:

- **שלב א'** - נוצר Zygote Fork.
- **שלב ב'** - הפונקציה ActivityThread.main מורצת (ע"י Reflection).
- **שלב ג'** - הפקודה הראשונה (bindApplication) של AMS נקראת אצל ה-Looper של ActivityThread ובה הנתונים מה-Manifest (שנאספו ע"י PMS).
- **שלב ד'** - נוצר אובייקט Application בזכרון, אבל UI Thread עדיין לא מריץ אותו!
- **שלב ה'** - כתגובה להודעה מה-AMS, הפונקציה handleBindApplication מורצת ע"י ActivityThread, ה"סוכן" של System Server בתוך האפליקציה.

הוא מבצע 2 פעולות:

- נוצר אובייקט Application אך עדיין לא מורץ.
- ביצוע ContentProvider.onCreate עבור כל ה-Providers שב-Manifest.
- ביצוע Application.onCreate עבור הרצת אובייקט ה-Application.

פקודת ContentProvider.onCreate

אתחול ה-Providers ע"י פונקציית עזר בשם installContentProviders:

```
if (!data.restrictedBackupMode) {
    if (!ArrayUtils.isEmpty(data.providers)) {
        installContentProviders(app, data.providers); // <---
    }
}
```

כאן נטען SomeProvider שמריץ את onCreate (כל Provider בודד, חייב לרשת מ-ContentProvider ולממש את onCreate).

- **שלב ו'** - AMS שולח פקודה ל-ActivityThread, שמפעילה את ה-Activity.onCreate וגורמת להעלאת ה-UI של האפליקציה.

השלכות למחקר הדינמי

ברגע שהרצתי את ה-Hook, נוצר אובייקט Application בזכרון. אמנם כעת חסרה פונקציית הפענוח של קבצי ה-Dex שישבה בראש אובייקט ה-Application של הנוזקה (שדרסנו). זה הוביל לתלונת ה-ClassLoader אודות קובץ ה-DEX האבוד.

בשלב המימוש, רציתי לזייף MySomeProvider תוך כדי ריצה ב-Frida, ולשלוח אותו ל-ClassLoader, כדי שלא יבכה שלא מצא אותו.

עם קצת ניסוי וטעיה, למדתי על הדינמיקה עם ה-ClassLoader. לא מספיק להזריק את ה-Class שלי ל-ART, כי ה-ClassLoader מחפש Path של קובץ בדיסק הקשיח (ולא ב-RAM), ברגע טעינת ה-ActivityThread.

תיאור ה-Hook במילים:

- ליצור SomeProvider.java ולקמפל ידנית ל-DEX
- לאתחל מופע חדש של DexClassLoader, שמירת ה-Path, הכנת MySomeProvider להזרקה ועדכון האב של ClassLoader.
- עצירת שרשרת האספקה והזרקת ה-MySomeProvider שיצרתי, לתוך תהליך קבלת ה-DEX ל-ART (לא הרחבתי, כדי לא לגלוש מגבולות המאמר).
- כפיית אובייקט Default Application (כמו שכבר הכרנו).

הקוד:

```
Java.perform(function() {
    var dexPath = "/sdcard/someprovider.dex";

    var ParentClassLoader = Java.use("java.lang.ClassLoader").getSystemClassLoader();
    var ClassLoader = Java.use("dalvik.system.DexClassLoader");

    var myLoader = ClassLoader.$new(dexPath, "/sdcard/", null, ParentClassLoader);
    var MySomeProvider =
myLoader.loadClass("com.clnvyokcv.bgvpowkic.util.SomeProvider");
// ------------------------------------------------------------
    var BaseDexClassLoader = Java.use("dalvik.system.BaseDexClassLoader");

    BaseDexClassLoader.findClass.implementation = function(name) {
        if (name === "com.clnvyokcv.bgvpowkic.util.SomeProvider") {
            return MySomeProvider;
        }
        return this.findClass(name);
    };
// ------------------------------------------------------------
    var reset_entry_point = Java.use("android.app.LoadedApk");

    reset_entry_point.makeApplicationInner
    .overload('boolean', 'android.app.Instrumentation', 'boolean')
    .implementation = function(forceDefaultAppClass, instrumentation,
allowDuplicateInstances) {
        return this.makeApplicationInner(true, instrumentation,
allowDuplicateInstances);
    };
});
```

הפלט:

```
📦[shay0129@mobile-lab Anatsa_2024]$ frida -U -f com.clnvyokcv.bgvpowkic -l eight_hook.js
     ____
    / _  |   Frida 17.9.1 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Android Emulator 5554 (id=emulator-5554)
Spawned `com.clnvyokcv.bgvpowkic`. Resuming main thread!
[Android Emulator 5554::com.clnvyokcv.bgvpowkic ]-> [*] Initializing Entry Point Bypass...
[+] Mock Stub Class loaded successfully: java.lang.Class
[*] Bypass instrumentation active. Waiting for application to load...
[+] [SUCCESS] Application Entry Point bypassed successfully.
[+] Intercepting resolution request for: com.clnvyokcv.bgvpowkic.util.SomeProvider
```

סיכום

ראינו איך מכשיר האנדרואיד שלנו, כבר לא כ"כ בטוח כמו שחשבנו. המורכבות שלו יוצרת "פתחים", שנוזקות מתקדמות יודעות לנצל. אותי זה הביא למסקנה - להשתמש בו לצרכי מחקר ופחות לצרכים אישיים ושימוש יומיומי (ולהעדיף פלאפון מקשים).

מערכת מורכבת כמו אנדרואיד, מלאה ב-APIs שמקשרים לעוד APIs, והמון דברים קורים "מתחת לפני השטח".

המחקר הוביל למסקנה, שבאנדרואיד אין "נקודת כניסה" אחת לאפליקציה, אלא שרשרת Bootstrap מרובת שלבים. כל אחד מהשלבים מהווה שטח תקיפה בפני עצמו עבור נוזקות.

מה שריגש אותי במחקר, הוא לראות איך שבריר שנייה בודד בזמן הריצה, יכול להרגיש כמו "אינסוף", של שעות ארוכות מול המסך. כמו שכתוב בתורה ,"יום לשנה" - יש דברים שקרו ביום אחד, ולוקח שנה לספר אותם.

על המחבר

שי מרדכי, סטודנט לקראת סיום תואר ראשון במדעי המחשב במכון לב, משרת מילואים, נשוי ואב לשניים. משלב לימודי רבנות בכולל, עם תשוקה עמוקה לחקר חולשות, Reverse Engineering ו-Android Internals. אוהב לחקור לעומק מערכות הפעלה ולפרק בעיות מורכבות עד לשורשן.

תודה גדולה לברק גונן - מרצה, חוקר, ובוגר 8200, שהצית בי את התשוקה לחקר מערכות הפעלה ו-RE.

מקורות מידע

- android/app/LoadedApk.java - Implementation of makeApplicationInner()
- android/app/Instrumentation.java - Implementation of newApplication() and newActivity()
- android/app/ActivityThread.java - Implementation of handleBindApplication() and installContentProviders()
- dalvik/system/BaseDexClassLoader.java - Implementation of findClass() used for Dex loading.
- Android Developers: DexClassLoader
- Android Developers: Processes and Threads Overview.
- Android Developers: App Manifest Overview (<application> element).
- Android Developers: ContentProvider

- Android Developers: Create a content provider
- Stefano Santilli: Deep Dive into Android Boot Process (Part 1).
- Stefano Santilli: Android Boot Process (Part 2: From Zygote to SystemServer).
- Cleafy Labs: A Stealthy Threat Uncovered - TeaBot on Google Play Store.
- Frida Documentation: Modes of Operation (Spawn vs. Attach).
- Daniel Bergers: Security Notes 002: Getting Started with Frida.
- Frida Releases: Frida v17.6.0 updates.
- OWASP MASTG: UnCrackable Mobile Apps (Level 2).
- Axelle Apvrille: Reverse Android Malware Like a Jedi Master
- Axelle Apvrille: Live Reverse Engineering of a Trojanized Medical App.
- Malware Sample Hash (SHA-256): 262e1c40fb33ac2cfb766cb277505f6ebdf0851252a33a83859a3bfb3e54af8c (Available on VirusTotal / InQuest).

מהגנה למודיעין: בניית Honeypot מבוסס Cloudflare לזיהוי והטעיית תוקפים

מאת גיא תותחני

תקציר מנהלים

במהלך מספר ימים של ניטור שגרתי החלו להופיע בלוגים בקשות חוזרות לנתיבים כגון /wp-login.php, /.env ו-xmlrpc.php/. במבט ראשון הן נראו כמו רעש רקע רגיל של האינטרנט, אך ניתוח מעמיק יותר חשף פעילות אוטומטית רחבת היקף שבוצעה באמצעות תשתיות ענן ציבוריות ברחבי העולם. במקום לחסום את התוקפים מיד, הוחלט לנצל את ההזדמנות כדי להבין כיצד הם פועלים, אילו חולשות הם מחפשים ואילו כלים הם מפעילים. ההחלטה הזו הפכה בסופו של דבר לפרויקט שקיבל את השם **Operation GhostEdge.**

מאמר זה מתעד את פרויקט Operation GhostEdge. הקמת מערכת הגנה, זיהוי והטעיה (Deception) עבור אתר אינטרנט ציבורי, אשר נבנתה כולה על גבי תשתית הקצה של Cloudflare. הפרויקט נולד מתוך תצפית פשוטה אך נפוצה: כל שירות Web החשוף לאינטרנט, ללא קשר לגודלו או לחשיבותו, הופך כמעט מיד ליעד של תעבורה עוינת אוטומטית. בקשות חשודות מגיעות מסורקים, בוטים ותשתיות ענן ציבוריות, ומחפשות בשיטתיות חולשות מוכרות, ממשקי ניהול חשופים וקבצים רגישים.

מטרת המאמר היא לתאר את מלוא מחזור החיים של הפרויקט: החל מזיהוי הפעילות העוינת וניתוחה, דרך מידולה במונחי איום (Threat Model) ומיפויה ל-MITRE ATT&CK, וכלה בתכנון והטמעה של ארכיטקטורת הגנה רב-שכבתית. הארכיטקטורה משלבת בין סינון ב-Edge, מנגנון Threat Scoring דינמי, שכבת Honeypot המדמה שירותים פגיעים ומנגנון Tarpitting להאטת סורקים. הדגש המרכזי של הפרויקט אינו חסימה בלבד, אלא הפיכת התקיפה עצמה למקור Threat Intelligence מבצעי.

המאמר נכתב עבור קהל יעד מגוון. חשוב להדגיש כבר בפתח הדברים כי המערכת נבנתה למטרות הגנה, מחקר ולימוד בלבד. לא נעשה בה שימוש התקפי, לא בוצע ניסיון לפגוע במערכות צד שלישי, וכל המידע שנאסף שימש לצורכי Detection, Threat Intelligence ושיפור יכולות ההגנה בלבד.

רקע

עבור מפעילי אתרים רבים, המפגש הראשון עם תוקפים אינו מתרחש בעקבות אירוע אבטחה משמעותי או ניסיון פריצה מתוחכם, אלא כבר בשעות הראשונות לאחר העלאת השירות לאוויר. מערכות סריקה

אוטומטיות פועלות באופן רציף ברחבי האינטרנט, מחפשות שירותים חדשים, תתי-דומיינים שנחשפו זה עתה ויישומים אשר טרם הוקשחו. במקרים רבים ניתן לראות ניסיונות גישה לנתיבים מוכרים זמן קצר לאחר פרסום האתר, גם כאשר הוא עדיין אינו מופיע במנועי חיפוש ואינו מוכר לציבור הרחב.

תופעה זו אינה חריגה למעשה, היא הפכה לחלק בלתי נפרד מהמציאות של כל שירות Web ציבורי. האינטרנט המודרני מלא בבוטים, סורקים אוטומטיים ותשתיות תקיפה מבוזרות המבצעים ללא הפסקה פעולות Reconnaissance, כלומר איסוף מידע ומיפוי מטרות פוטנציאליות. חלק מהפעילות מבוצע על ידי מנועי חיפוש ושירותי אינדוקס לגיטימיים, אך חלק משמעותי ממנה מגיע ממערכות שמטרתן לאתר חולשות, שירותים חשופים וטעויות תצורה שניתן לנצל.

בניגוד לדימוי המקובל של תוקף היושב מול מחשב ובוחר יעד ספציפי, רוב הפעילות העוינת המופנית כיום כלפי שירותי Web מבוצעת באופן אוטומטי לחלוטין. התוקפים אינם מכירים את הארגון, אינם יודעים מי מפעיל את האתר ולעיתים אף אינם יודעים מהו תוכן השירות. במקום זאת הם מפעילים מערכות סריקה רחבות היקף אשר בודקות מיליוני כתובות אינטרנט ביום, בניסיון לאתר מטרות פגיעות.

גישה זו מכונה Mass Scanning. מטרתה פשוטה: לבצע מספר עצום של ניסיונות זיהוי בעלות נמוכה ככל האפשר, מתוך הנחה שגם אם רק אחוז קטן מאוד מהיעדים יתגלה כפגיע, ההשקעה תחזיר את עצמה. מבחינת התוקף, אין משמעות מיוחדת לאתר מסוים כל יעד הוא עוד כתובת ברשימת הסריקה.

הסריקות האוטומטיות מכוונות בדרך כלל למטרות נפוצות אשר קיימת סבירות גבוהה למצוא אותן ברשת. בין היעדים השכיחים ניתן למצוא התקנות WordPress, ממשקי ניהול חשופים, קבצי Environment המכילים סודות מערכת, חולשות PHP ידועות, תוספים פגיעים, קבצי גיבוי שנשכחו על השרת, ממשקי מסדי נתונים וכלי פיתוח אשר נותרו זמינים בסביבת הייצור.

דפוס פעולה זה מסביר תופעה שעלולה להיראות מבלבלת במבט ראשון. במסגרת הפרויקט זוהו ניסיונות גישה חוזרים ונשנים לנתיבים כגון:

```
/wp-login.php
/.env
/xmlrpc.php
/vendor/phpunit/eval-stdin.php
/shell.php
/adminer.php
```

למרות שהאתר שנבדק כלל לא התבסס על WordPress, לא השתמש ב-PHP ולא הכיל את הרכיבים הללו, הסורקים המשיכו לבדוק אותם שוב ושוב. הסיבה לכך פשוטה: הסורק אינו "יודע" באיזו טכנולוגיה האתר משתמש. הוא פועל לפי רשימת מטרות קבועה מראש. אם אחד הנתיבים קיים ומחזיר תגובה מעניינת, הוא יסומן להמשך בדיקה או לניסיון ניצול. אם הנתיב אינו קיים, הסורק ימשיך מיד אל היעד הבא.

כל אחד מהנתיבים הללו מייצג יעד תקיפה מוכר. חלקם משמשים כממשקי התחברות, חלקם עלולים לחשוף מידע רגיש, וחלקם קשורים לחולשות אשר נוצלו בעבר בקמפיינים רחבי היקף. לכן עצם הניסיון לגשת אליהם מהווה לעיתים אינדיקציה ראשונית לפעילות סריקה או לניסיון Exploitation.

מאחורי רוב קמפייני הסריקה האלו עומד היגיון כלכלי פשוט. עלות הסריקה נמוכה במיוחד. תוקף יכול לשכור שרת וירטואלי (VPS) בעלות של דולרים בודדים לחודש, להריץ עליו כלי סריקה אוטומטיים ולכסות בתוך זמן קצר מיליוני יעדים. גם אם רק חלק קטן מאוד מהמערכות שייסרקו יימצא פגיע, התשואה הפוטנציאלית עדיין מצדיקה את המאמץ. מאפיין נוסף של פעילות זו הוא השימוש הנרחב בתשתיות ענן ציבוריות. ספקי ענן מודרניים מאפשרים הקמה מהירה של שרתים, החלפת כתובות IP בתוך דקות בודדות והרחבת פעילות בהיקפים גדולים מאוד.

במהלך החקירה המתוארת במאמר זה זוהה כי חלק משמעותי מהתעבורה החשודה הגיע מתשתיות ענן ציבוריות מוכרות, ובהן Oracle Cloud, Microsoft Azure, DigitalOcean וספקים נוספים. עובדה זו אינה מעידה בהכרח על מעורבותם של הספקים עצמם, אלא על השימוש שעשו התוקפים במשאבי הענן הזמינים להם. מעבר לניסיונות מיפוי ואיתור נתיבים, פעילות אוטומטית מסוג זה כוללת לעיתים קרובות גם ניסיונות Credential Stuffing ו-Brute Force. מתקפות אלו מתבססות על מאגרי סיסמאות וחשבונות שדלפו בעבר ממערכות אחרות, מתוך הנחה שמשתמשים רבים עושים שימוש חוזר באותם פרטי התחברות במספר שירותים שונים. כאשר ניסיון כזה מצליח, התוקף עשוי לקבל גישה לחשבון תקין מבלי לנצל חולשה טכנית כלשהי.

במהלך הניטור השוטף של אתר הלקוח זוהו שלושה דפוסי פעילות מרכזיים:

- **Reconnaissance** - ניסיונות מיפוי ואיתור נתיבים ושירותים
- **Credential Stuffing** - ניסיונות התחברות באמצעות פרטי הזדהות גנובים
- **Exploit Scanning** - חיפוש פעיל אחר חולשות הניתנות לניצול

השילוב בין שלושת הדפוסים הללו הצביע על כך שלא מדובר ברעש רקע שגרתי בלבד, אלא בפעילות אוטומטית בעלת מאפיינים התקפיים ברורים. בשלב זה עלתה השאלה המרכזית שהובילה להקמת הפרויקט: האם נכון פשוט לחסום את הבקשות ולהמשיך הלאה, או שניתן לנצל את הפעילות כדי ללמוד על התוקפים, לאסוף מודיעין ולשפר את יכולות ההגנה של המערכת?

מטרות

לאחר שהתגבשה ההבנה כי האתר אינו מתמודד עם אירוע נקודתי אלא עם זרם קבוע של סריקות אוטומטיות, עלתה השאלה כיצד נכון להגיב לפעילות זו. האפשרות הפשוטה ביותר הייתה להסתפק בחסימה באמצעות חוקי Firewall ו-WAF, אך גישה זו הייתה פותרת רק חלק מהבעיה. היא אמנם הייתה

מונעת חלק מהבקשות החשודות, אך לא הייתה מספקת תשובות לשאלות החשובות באמת: מי עומד מאחורי הפעילות? אילו חולשות מחפשים התוקפים? האם מדובר בקמפיין רחב היקף? ומה ניתן ללמוד ממנו כדי לשפר את ההגנה בעתיד? מתוך שאלות אלו גובשו מטרות הפרויקט, אשר השפיעו על כל החלטה ארכיטקטונית שהתקבלה בהמשך הדרך.

המטרה הראשונה הייתה **זיהוי וניתוח של פעילות עוינת**. במקום לראות בכל בקשה חשודה עוד אירוע בלוגים, המערכת נדרשה לזהות דפוסים חוזרים, להבחין בין פעילות לגיטימית לפעילות עוינת ולספק תמונה ברורה של סוגי האיומים המופנים כלפי האתר

במקביל, היה צורך **להגן על שרת המקור** ולצמצם ככל האפשר את החשיפה שלו לאינטרנט הציבורי. אחת ההחלטות המרכזיות בפרויקט הייתה לעצור חלק משמעותי מהתנועה החשודה כבר בשכבת ה-Edge של Cloudflare, עוד לפני שהיא מגיעה לתשתית האמיתית.

מטרה נוספת הייתה **למנוע ניסיונות Exploitation** של חולשות Web מוכרות. גם כאשר האתר אינו משתמש בטכנולוגיות מסוימות, תוקפים ממשיכים לחפש נתיבים, קבצים ורכיבים פגיעים. המערכת נדרשה לזהות ניסיונות אלו מוקדם ככל האפשר ולמנוע מהם להגיע לסביבת הייצור.

אולם בשונה ממערכות הגנה מסורתיות, מטרת הפרויקט לא הייתה חסימה בלבד. אחד המרכיבים המרכזיים בתכנון היה הקמת **סביבת Honeypot מבוקרת**, אשר תדמה שירותים פגיעים ותאפשר לתוקפים להמשיך בפעילותם בתוך סביבה מבוקרת. גישה זו נועדה להפוך את ניסיון התקיפה עצמו למקור מידע בעל ערך.

כתוצאה מכך הוגדרה גם מטרה נוספת: **איסוף Threat Intelligence בזמן אמת**. המערכת תוכננה לתעד מידע על כתובות IP, User-Agents, Payloads, דפוסי תקיפה, תדירות פעילות ומאפיינים נוספים אשר יכולים לסייע בהבנת האיום ובשיפור מנגנוני ההגנה.

מטרה נוספת הייתה **צמצום שטח התקיפה (Attack Surface)** של האתר. ככל שפחות שירותים, נתיבים וממשקים חשופים לתוקפים, כך קטן הסיכון לפגיעה במערכת. לכן הארכיטקטורה תוכננה כך שחלק ניכר מהאינטראקציה עם התוקפים יתבצע בשכבת ה-Edge ולא מול שרת המקור.

בסופו של דבר, כל המטרות הללו התכנסו לרעיון מרכזי אחד: להפוך את שכבת ההגנה ממנגנון תגובתי אשר רק חוסם תוקפים, לפלטפורמה המסוגלת לזהות, לנתח, ללמוד ולאסוף מודיעין מתוך הפעילות העוינת עצמה.

Threat Model

לפני שניתן להחליט כיצד להגן על מערכת, יש להבין מפני מה היא נדרשת להתגונן. מודל איום (Threat Model) הוא כלי המשמש לזיהוי הנכסים הדורשים הגנה, סוגי התוקפים הפוטנציאליים ומטרות התקיפה האפשריות. הגדרה מסודרת של מודל האיום מאפשרת להתאים את מנגנוני ההגנה לסיכונים הממשיים ולא לאיומים תאורטיים בלבד.

נכסים מוגנים

המערכת נועדה להגן על מספר נכסים מרכזיים:

- אתר הלקוח
- שרת המקור (Origin Server)
- פרטי הזדהות (Credentials)
- מידע רגיש
- זמינות השירות

חשוב להבחין בין נכסים גלויים לנכסים סמויים. שרת המקור הוא יעד ברור שניתן לנסות לתקוף באופן ישיר, אך גם מידע כגון מבנה הנתיבים הפנימי, גרסאות תוכנה, מפתחות גישה וקבצי תצורה עשוי להיות בעל ערך רב עבור תוקף.

אחת ממטרות שכבת ה-Edge היא לצמצם ככל האפשר את כמות המידע שהתוקף מסוגל ללמוד על נכסים אלו, ובכך להקטין את יכולתו לבצע Reconnaissance ולתכנן מתקפה ממוקדת.

סוגי תוקפים

הפעילות שנצפתה במהלך החקירה הצביעה בעיקר על תוקפים אוטומטיים והזדמנותיים, ולא על גורם אשר מיקד את פעילותו בארגון מסוים.

בין סוגי הפעילות שזוהו:

- Automated Scanners
- Opportunistic Attackers
- Credential Stuffing Bots
- Distributed Reconnaissance Infrastructure

ההבחנה בין תוקף ממוקד (Targeted) לבין תוקף הזדמנותי (Opportunistic) משמעותית במיוחד. תוקף ממוקד עשוי להשקיע זמן, משאבים ויצירתיות כדי לעקוף מנגנוני הגנה. לעומתו, תוקף הזדמנותי מחפש מטרות קלות ונוטה לעבור ליעד הבא ברגע שהעלות או המאמץ הנדרשים ממנו עולים.

מרבית הפעילות שנצפתה בפרויקט תאמה לפרופיל ההזדמנותי. מסיבה זו, מנגנונים כגון Tarpitting ו-Deception צפויים להיות אפקטיביים במיוחד, שכן הם מגדילים את עלות התקיפה ומפחיתים את הכדאיות שלה.

מטרות התקיפה

ניתוח הפעילות העלה מספר מטרות אפשריות של התוקפים:

- Account Takeover
- Credential Harvesting
- Environment Disclosure
- Remote Code Execution
- Web Shell Deployment

ניתן לראות מטרות אלו כשלבים שונים לאורך שרשרת תקיפה טיפוסית. בשלב הראשון התוקף מנסה לאסוף מידע על המערכת ולזהות חולשות אפשריות. לאחר מכן עשויים להגיע ניסיונות השגת גישה באמצעות Credentials גנובים או חשיפת מידע רגיש. במקרים חמורים יותר, המטרה עשויה להיות הרצת קוד מרחוק או השתלת Web Shell לצורך יצירת אחיזה מתמשכת במערכת.

שרשרת תקיפה זו מתחילה בדרך כלל ב-Reconnaissance, ממשיכה לניסיון גישה או ניצול חולשה ומסתיימת בהשגת שליטה או התמדה (Persistence). כפי שיוצג בהמשך המאמר, מיפוי הפעילות ל- MITRE ATT&CK משקף באופן ברור את המעבר בין שלבים אלו.

סביבת העבודה והטכנולוגיות

הפרויקט נבנה כולו על גבי שכבת הקצה של Cloudflare, ללא צורך בשרתים ייעודיים נוספים. פרק זה מתאר בקצרה את הרכיבים המרכזיים ואת תפקידם, ומפנה לתיעוד הרשמי עבור הרחבה.

רכיב	תיאור התפקיד
Client Server (Origin)	שרת המקור של האתר הנכס שעליו מגנים
Cloudflare	Reverse Proxy ושכבת הגנת Edge מול האתר
Cloudflare Workers	מנוע ה-Honeypot, הזיהוי וה-Scoring
Cloudflare WAF	סינון וחסימת תקיפות לפי חוקים
Rate Limiting	מניעת Brute Force ו-Flooding
Firewall / Security Events	מקור ה-Telemetry לניתוח

[רכיבי הסביבה ותפקידם]

Cloudflare ושכבת ה-Edge

Cloudflare היא רשת תשתית עולמית הפועלת כ-Reverse Proxy בין המבקרים לבין שרת המקור. כל בקשה עוברת תחילה דרך נקודת קצה של Cloudflare הקרובה גאוגרפית למבקר, ושם ניתן לבדוק, לסנן או להסיט אותה עוד לפני שהיא מגיעה לשרת. ארכיטקטורה זו היא הבסיס לכל יכולות ההגנה בפרויקט, משום שהיא מאפשרת לעצור פעילות עוינת ב-Edge ולחשוף את שרת המקור באופן מינימלי. מידע נוסף בדף המוצר: cloudflare.com/products/workers.

Cloudflare Workers

Cloudflare Workers היא פלטפורמת Serverless המריצה קוד JavaScript ישירות ב-Edge, בתוך מנגנון Isolates קל-משקל. לכל בקשה נכנסת מופעל מטפל **fetch()** המקבל את הבקשה ומחזיר תגובה. בפרויקט שלנו, ה-Worker הוא הלב של המערכת: הוא מבצע את הזיהוי, מחשב את ה-Threat Score, מנתב בקשות חשודות ל-Honeypot ואוסף Telemetry. תיעוד רשמי: developers.cloudflare.com/workers.

היתרון של מודל ה-Isolates על פני מודל מסורתי מבוסס Containers או מכונות וירטואליות הוא שהקוד רץ קרוב מאוד למבקר, ללא "Cold Start" משמעותי וללא צורך בניהול תשתית. עבור מערכת אבטחה זהו יתרון כפול: ראשית, ההחלטה (לחסום, לאתגר, להטעות) מתקבלת במהירות וב-Edge, לפני שהבקשה מגיעה לשרת שנית, הלוגיקה ניתנת לעדכון מהיר ומופצת גלובלית בתוך שניות. כך ניתן להגיב לדפוס תקיפה חדש כמעט בזמן אמת, מבלי לגעת בשרת המקור.

Cloudflare WAF

Cloudflare WAF בודק בקשות Web ו-API נכנסות ומסנן תעבורה לא רצויה לפי קבוצות חוקים (Rulesets). ה-WAF מבוסס על שפת ביטויים גמישה המאפשרת לסנן לפי מאפייני בקשה כגון כתובת IP, נתיב URL, Headers ותוכן ה-Body. בפרויקט שולב ה-WAF עם מנגנון ה-Scoring כדי לאפשר חסימה, Managed Challenge או ניתוב ל-Honeypot. תיעוד: developers.cloudflare.com/waf.

Rate Limiting

מנגנון ה-Rate Limiting של Cloudflare מגדיר תקרה לכמות הבקשות העומדות בביטוי מסוים בתוך חלון זמן נתון, ופועל כאשר התקרה נחצית. הוא חיוני לבלימת Brute Force, Credential Stuffing ו-POST Flooding, משום שהוא פוגע ישירות בקצב הפעולה של כלי Automation. תיעוד:
developers.cloudflare.com/waf/rate-limiting-rules

ניתוח המידע

מקורות הנתונים

הנתונים שעליהם התבסס הניתוח נאספו ממספר מקורות בתוך סביבת Cloudflare, אשר יחד סיפקו תמונה רחבה של התעבורה החשודה:

מקור נתונים	תיאור
Firewall Events	בקשות שנחסמו, אותגרו או סווגו כחשודות
Security Events	אירועי אבטחה ממערכת הניטור של Cloudflare
Bot Protection Logs	מידע על בוטים, Crawlers ותעבורה אוטומטית
Rate Limiting Logs	חסימות והגבלות קצב בעקבות פעילות חריגה
Worker Logs	Telemetry מתוך מערכת ה-Honeypot

[מקורות הנתונים לחקירה]

מתודולוגיית הניתוח

בשלב הראשוני בוצע ניתוח של Firewall Events ו-Worker Logs במטרה לזהות פעילות עוינת ודפוסים חוזרים. הניתוח כלל בחינה של כמות הבקשות והיקף הפעילות (Requests Count), זיהוי מדינות ותשתיות המקור (Source Countries), ניתוח כתובות IP חשודות ותשתיות ענן (Source IP Analysis), ניתוח כלי

תקיפה ובוטים לפי User-Agent, מיפוי הנתיבים שנבדקו (Path Enumeration), וסיווג שמפריד בין Crawlers לגיטימיים לבין פעילות עוינת (Bot Classification).

עיקרון מנחה בניתוח היה לבחון אינדיקציות בהקשר, ולא במנותק. בקשה בודדת לנתיב .php אינה מעידה בהכרח על תקיפה אך כאשר אותה כתובת IP פונה לעשרות נתיבי WordPress ברצף, בתדירות גבוהה ועם User-Agent של כלי Automation - התמונה המצטברת ברורה. לכן הניתוח התמקד ב-Correlation: קישור בין מאפייני בקשה שונים (User-AgentIP, נתיב, שיטה, תזמון) לכדי פרופיל פעילות אחד, שאותו ניתן לסווג ולנתח.

ממצאים מרכזיים

החקירה העלתה כי האתר היה יעד לסריקות אוטומטיות רחבות היקף, אשר בוצעו ברובן באמצעות תשתיות ענן ציבוריות. הטבלה הבאה מסכמת את הממצאים העיקריים:

פרמטר	ממצא
סוגי בקשות	GET / POST
מדינות שזוהו	ארה"ב ותשתיות ענן בינלאומיות
סוגי תנועה	Suspicious Reconnaissance לצד SEO Crawlers
נתיבים שנבדקו	wp-login.php / .env / shell.php
תשתיות שזוהו	AWS / Azure / Oracle Cloud / OVH

[ממצאים מרכזיים]

ניתוח IOC

במהלך החקירה זוהו מספר Indicators of Compromise (אינדיקטורים לפעילות עוינת) אשר הצביעו על Automation. הטבלה הבאה מציגה מבחר כתובות IP שנצפו, יחד עם ה-ASN והספק שמאחוריהן. ה-ASN הוא מזהה ייחודי לרשת המנוהלת על-ידי גורם אחד, ומאפשר לקשר כתובת IP לספק התשתית:

IP	מדינה	ASN	ספק	סוג פעילות
161.153.99.197	US	AS31898	Oracle Cloud	Credential Stuffing
161.118.254.192	SG	AS31898	Oracle Cloud	Automated Recon
150.136.244.33	US	AS31898	Oracle Cloud	Brute Force
174.138.54.2	US	AS14061	DigitalOcean	XMLRPC Recon
20.226.103.253	BR	AS8075	Microsoft Azure	PHP Enumeration
140.245.121.218	SG	AS31898	Oracle Cloud	XMLRPC Enumeration
193.143.1.112	RU	AS198953	Proton66 OOO	ENV File Recon

IP	מדינה	ASN	ספק	סוג פעילות
173.239.214.211	US	AS62240	Clouvider	WP Login Probe
103.230.178.122	IN	AS136634	Navkar Netsol	XMLRPC POST
172.202.0.52	US	AS8075	Microsoft Azure	PHP Path Enum
88.151.33.24	NL	AS41608	NextGenWebs	Git Repo Enum
176.65.139.232	NL	AS214472	Offshore LC	ENV Enumeration
195.178.110.159	NL	AS48090	TECHOFF SRV	Git Enumeration

[מבחר כתובות IP חשודות שזוהו (IOCs)]

רוב הכתובות החשודות השתמשו בתשתיות ענן ציבוריות דבר המאפיין קמפיינים אוטומטיים של Reconnaissance ו-Credential Stuffing. בנוסף זוהו דפוסים של User-Agent Rotation, שימוש ב-VPS ציבוריים, וניסיונות הסתרת Fingerprint.

ניתוח הפיזור הגאוגרפי וה-ASN מלמד שתי תובנות. ראשית, ריכוז ניכר של כתובות שייך ל-AS31898 Oracle Cloud דבר המעיד על קמפיין שעושה שימוש אינטנסיבי בספק ענן יחיד, ככל הנראה באמצעות מספר מכונות שהוקמו במקביל.

שנית, הפיזור על פני מדינות רבות (ארה"ב, סינגפור, ברזיל, רוסיה, הודו, הולנד) אינו מעיד בהכרח על מיקומו האמיתי של התוקף, אלא על מיקום מרכזי הנתונים של ספקי הענן. במילים אחרות, ה"מדינה" היא מאפיין של התשתית, לא של היריב. תובנה זו חשובה למדיניות החסימה: חסימה לפי מדינה (Geo-Blocking) תהיה גסה מדי ועלולה לחסום משתמשים לגיטימיים, בעוד שחסימה או ניתוב מבוססי ASN, יחד עם Threat Scoring ברמת הבקשה, מספקים איזון טוב בהרבה בין הגנה לבין שמירה על זמינות לתעבורה אמיתית.

מודיעין User-Agent

שדה ה-User-Agent מאפשר להבדיל בין פעילות לגיטימית לבין פעילות עוינת. הטבלה מציגה דוגמאות לסיווג שבוצע:

User-Agent	סיווג
Googlebot	Legit Search Engine
AhrefsBot	SEO Crawler
meta-webindexer	AI / Meta Indexer
curl	Suspicious Automation
sqlmap	Offensive Security Tool
Go-http-client	Automation Framework

[סיווג User-Agents]

השימוש ב-curl, ב-sqlmap ובכלי Automation נוספים הצביע על סריקה אוטומטית ולא על גלישה אנושית. בהמשך מובאים תיאורים קצרים של הכלים הפומביים הרלוונטיים.

על הכלים הפומביים שזוהו

curl - כלי שורת פקודה להעברת נתונים בפרוטוקולים שונים (בהם HTTP/S). לגיטימי לחלוטין בשימוש יומיומי, אך נפוץ מאוד גם בסקריפטים אוטומטיים של סריקה. אתר הכלי: curl.se.

wget - כלי GNU להורדת קבצים מהאינטרנט דרך שורת הפקודה, אף הוא לגיטימי אך נפוץ ב-Automation. אתר: gnu.org/software/wget.

sqlmap - כלי בדיקות חדירה (Offensive Security) קוד פתוח לאוטומציה של זיהוי וניצול חולשות SQL Injection. הופעתו ב-User-Agent מעידה כמעט תמיד על סריקה התקפית. אתר: sqlmap.org.

nikto - סורק שרתי Web בקוד פתוח המחפש קבצים מסוכנים, גרסאות מיושנות וקונפיגורציות בעייתיות. מאגר הפרויקט: github.com/sullo/nikto.

masscan - סורק פורטים מהיר במיוחד המסוגל לסרוק טווחי כתובות עצומים בזמן קצר כלי מרכזי בקמפיינים של Mass Scanning. מאגר: github.com/robertdavidgraham/masscan.

Googlebot / AhrefsBot - בוטים לגיטימיים: הראשון הוא ה-Crawler של מנוע החיפוש Google, והשני סורק SEO מסחרי. אלו זוהו בנפרד והוכנסו ל-Allowlist כדי לצמצם False Positives.

חשוב לזכור ששדה ה-User-Agent ניתן לזיוף בקלות תוקף יכול להציג כל מחרוזת שירצה, כולל התחזות ל-Googlebot. לכן ה-User-Agent לבדו אינו מספיק לסיווג, והוא משמש כאינדיקציה אחת מני רבות במנגנון ה-Scoring. אימות "בוטים מאומתים" (Verified Bots) מתבצע באמצעות בדיקות נוספות, כגון התאמה בין כתובת ה-IP לבין טווחי הכתובות הרשמיים של הספק, ולא על סמך מחרוזת ה-User-Agent בלבד.

דוגמאות מייצגות מתוך הלוגים

להמחשת אופי הפעילות, מובאות כאן מספר דוגמאות מייצגות של רשומות לוג, לאחר Sanitization..

ניסיון Credential Stuffing

```
POST /wp-login.php HTTP/1.1
Host: <target>
User-Agent: Mozilla/5.0 (compatible)
Content-Type: application/x-www-form-urlencoded
log=admin&pwd=********&wp-submit=Log+In
-> classification: HIGH (credential_stuffing)
```

הדוגמה ממחישה ניסיון התחברות אוטומטי לנתיב WordPress. שדה הסיסמה הוצג ממוסך (Masked) המערכת שמרה Hash בלבד לצורכי Correlation, בהתאם לעקרונות ה-Data Minimization.

WordPress של Fingerprinting

```
GET /wp-includes/js/wp-emoji-release.min.js HTTP/1.1
User-Agent: Go-http-client/1.1
-> classification: MEDIUM (wp_fingerprinting)
```

האירוע הצביע על ניסיון Fingerprinting של סביבת WordPress באמצעות גישה לנתיב סטטי מוכר. השימוש ב-Go-http-client הצביע על Automation ולא על גלישה אנושית.

XMLRPC Abuse עם החלפת User-Agent

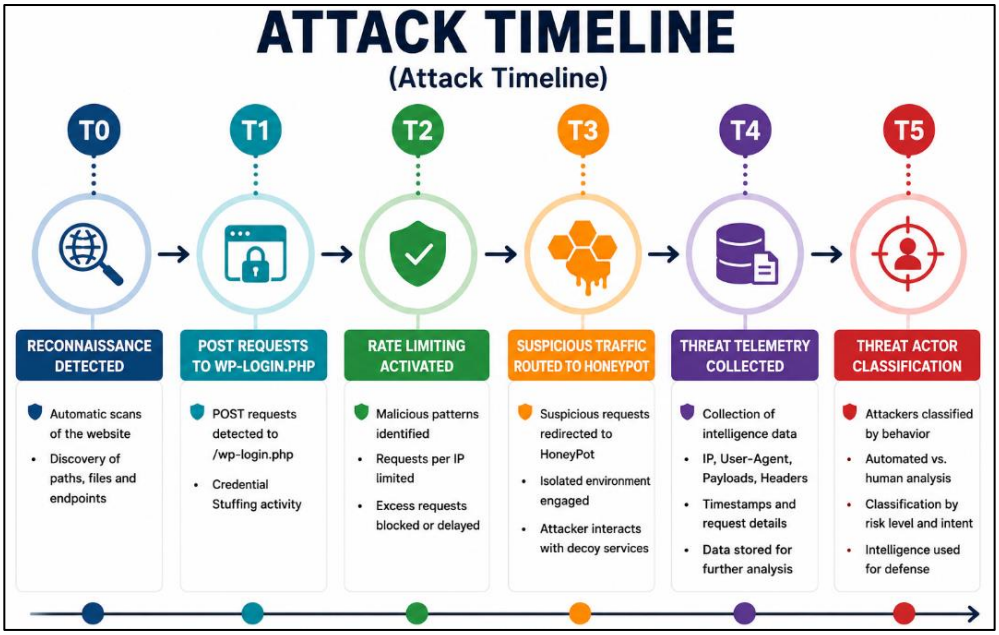
```
POST /xmlrpc.php HTTP/1.1
User-Agent: Go-http-client/1.1
<methodCall><methodName>metaWeblog.newPost</methodName> ...
-> classification: CRITICAL (xmlrpc_abuse)
   (repeated from same IP with rotated User-Agent)
```

המערכת זיהתה ניסיון XMLRPC Abuse באמצעות קריאה ל-metaWeblog.newPost, עם Payload בפורמט XML מלא. כאשר אותו מקור חזר על הניסיון תוך החלפת ה-User-Agent, סומן הדבר כ-User-Agent Rotation וכ-Bot Obfuscation ניסיון לעקוף זיהוי מבוסס Fingerprinting. שילוב של POST, XMLRPC Abuse Credential Attempt ומבנה Payload חשוד הוביל לסיווג Critical.

מסקנות Threat Intelligence ראשוניות

ניתוח האירועים העלה כי לא מדובר בתוקף בודד, אלא בפעילות אוטומטית רחבת היקף שבוצעה באמצעות תשתיות ענן ציבוריות. הממצאים הצביעו על קמפיינים של Credential Stuffing, על Reconnaissance מבוזר, על Enumeration של WordPress, על תשתית תקיפה מבוססת ענן, ועל ניסיונות Exploitation אוטומטיים. דפוסי הפעילות תאמו קמפיינים של Mass Scanning המבוצעים כנגד שירותי Web ציבוריים.

ציר זמן של האירועים



מיפוי MITRE ATT&CK

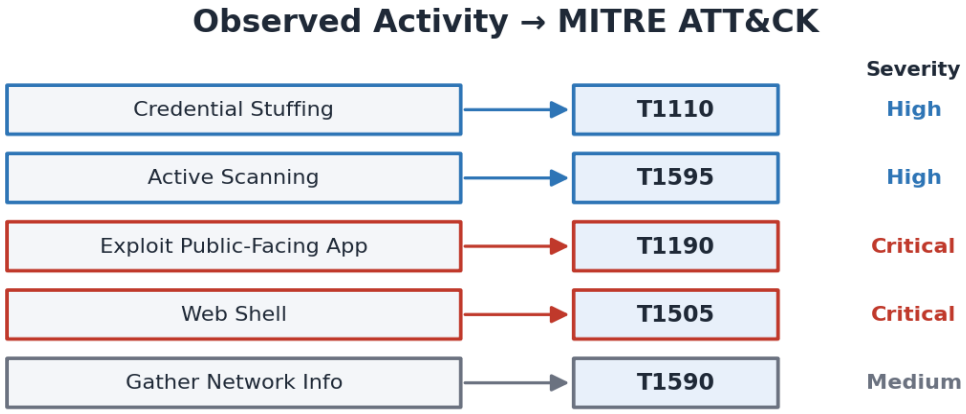
כדי לתאר את הפעילות בשפה אחידה ומקובלת בתעשייה, מופתה הפעילות שנצפתה למסגרת MITRE ATT&CK בסיס ידע פתוח של טכניקות תקיפה. המיפוי מסייע להבין את שלבי התקיפה ולתעדף תגובה:

Technique	MITRE ID	Evidence	Severity
Credential Stuffing	T1110	POST ל-/wp-login.php	High
Active Scanning	T1595	XMLRPC / ENV Enum	High
Exploit Public-Facing App	T1190	phpunit/eval-stdin probes	Critical
Web Shell	T1505	בקשות shell.php	Critical
Gather Network Info	T1590	בקשות .git/config	Medium

[מקור: MITRE ATT&CK. attack.mitre.org/techniques/]

הפניות ישירות לעמודי הטכניקות: T1110 , T1595 , T1190, T1505, T1590

המיפוי משקף את שלבי שרשרת התקיפה כפי שנצפו בפועל. TGather Network Information (1590) ו-TActive Scanning (1595) הם שלבי איסוף המידע הראשוניים התוקף ממפה את היעד ומחפש נקודות כניסה. TExploit Public-Facing Application (1190) ו-TWeb Shell (1505) מייצגים את שלב הניצול וההשתלטות, שבו התוקף מנסה להריץ קוד או להשתיל אחיזה מתמשכת. TBrute Force (1110), הכולל Credential Stuffing, מכוון להשגת גישה דרך פרטי הזדהות. הצגת הפעילות בשפה המשותפת של ATT&CK מאפשרת לתקשר את הסיכון לגורמים שונים בארגון ולתעדף תגובה לפי שלב התקיפה וחומרתו.



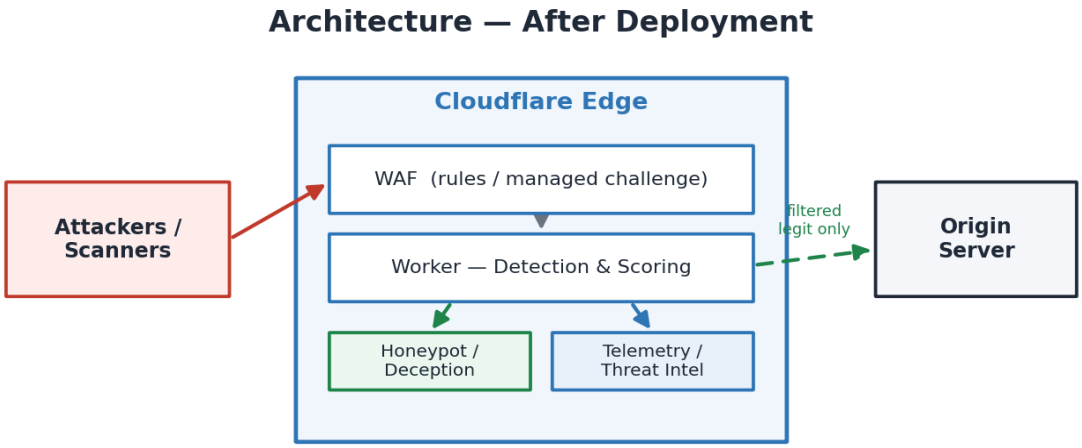
ארכיטקטורת ההגנה

לפני ההטמעה

במצב ההתחלתי, בקשות חשודות הגיעו ישירות לשרת המקור. לא נאסף מודיעין מתקדם, לא בוצעה Deception, ולא התקיים Threat Classification. שטח התקיפה היה חשוף, וכל בקשה לגיטימית או עוינת הגיעה אל השרת באופן זהה:

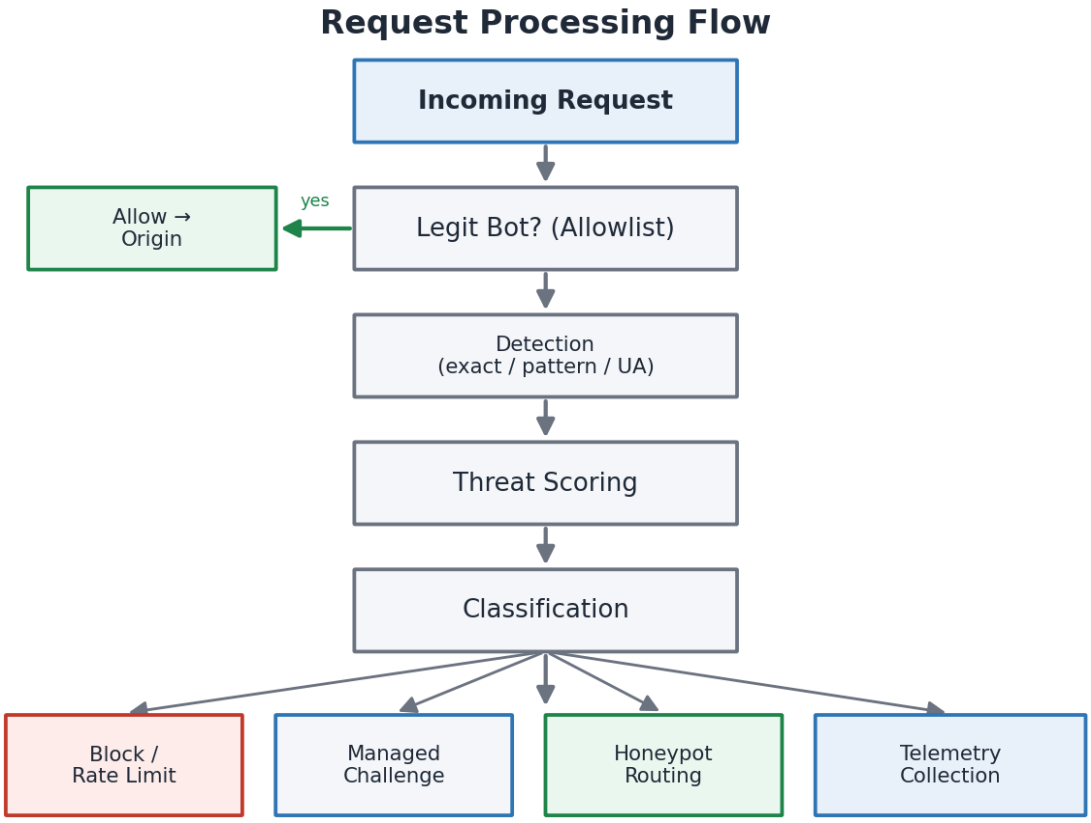
לאחר ההטמעה

הארכיטקטורה החדשה ממקמת את Cloudflare כשכבת Edge בין התוקפים לבין שרת המקור. בקשות עוברות תחילה דרך ה-WAF, משם ל-Worker שמבצע זיהוי ו-Scoring, ובהתאם לסיווג מנותבות לחסימה, ל-Managed Challenge, ל-Honeypot או ל-Telemetry. רק תעבורה לגיטימית מסוננת ממשיכה אל שרת המקור. כך מתקבלת עצירת איומים ב-Edge, הפחתת חשיפת השרת, איסוף מודיעין בזמן אמת, שכבת Deception ו-Threat Classification מובנה.



מנוע ה-Worker וה-Honeypot

ה-Cloudflare Worker מהווה את מנוע ה-Detection וה-Deception של המערכת. עבור כל בקשה הוא מבצע רצף פעולות: זיהוי (Detection), חישוב סיכון (Threat Scoring), ניתוב ל-Honeypot במידת הצורך, איסוף Telemetry, החזרת תגובות מזויפות (Fake Responses) וסיווג (Classification). מושג ה-Honeypot מערכת מלכודת המדמה יעד פגיע כדי למשוך ולנתח תוקפים מתואר בהרחבה בערך Honeypot computing בוויקיפדיה.



סדר הפעולות בצינור העיבוד (Pipeline) אינו מקרי. תחילה נבדק האם מדובר בבוט לגיטימי מוכר אם כן, הבקשה מועברת ישירות לשרת המקור ללא עיכוב. בקשות שאינן ב-Allowlist עוברות לשלב הזיהוי, ומשם ל-Scoring ולסיווג. רק לאחר שהבקשה סווגה נבחרת הפעולה המתאימה: חסימה, Managed Challenge, ניתוב ל-Honeypot, או רישום Telemetry. סדר זה מבטיח שתעבורה לגיטימית כמעט אינה מושפעת מהמערכת, בעוד שתעבורה חשודה מטופלת באופן מדורג ופרופורציונלי לרמת הסיכון שלה.

מנגנוני זיהוי

Exact Match Detection

המערכת בדקה נתיבים חשודים בהתאמה מדויקת, ובהם /, /adminer.php, /.env, wp-login.php, /shell.php ו-/server-status. נתיבים אלו אינם קיימים באתר הלגיטימי, ולכן גישה אליהם היא אינדיקציה חזקה לסריקה. היתרון בשיטה זו הוא דיוק גבוה ומיעוט False Positives: מבקר אנושי לגיטימי כמעט לעולם לא יפנה ישירות אל wp-login.php/ באתר שאינו מבוסס WordPress. החיסרון הוא נוקשות הרשימה סטטית, ותוקף שמשתמש בנתיב חדש שאינו ברשימה לא ייתפס. לכן Exact Match הוא רק שכבה אחת, והוא משולב עם זיהוי גמיש יותר מבוסס דפוסים.

Pattern Matching

מעבר להתאמות מדויקות, בוצע זיהוי לפי דפוסים (Patterns) המופיעים בנתיב או בבקשה, כגון php., phpunit, shell, git.wordpress ו-obfuscate. גישה זו תופסת גם וריאציות חדשות של נתיבים שלא נצפו קודם.

זיהוי מבוסס-דפוסים מרחיב משמעותית את הכיסוי, אך מחייב כיול זהיר. דפוס רחב מדי עלול לתפוס תעבורה לגיטימית (False Positive), בעוד שדפוס צר מדי יחמיץ וריאציות (False Negative). האיזון בין השניים הוא אחת ההחלטות המרכזיות בעיצוב מערכת זיהוי, ובפרויקט זה הוא נתמך על-ידי מנגנון ה-Scoring שמשקלל את הדפוס יחד עם אינדיקציות נוספות במקום לחסום אוטומטית על כל התאמה.

User-Agent Analysis

נותחו User-Agents המעידים על Automation, כגון curl, wget, sqlmap, masscan ו-nikto, וכן בקשות חסרות מאפייני דפדפן (Headless). שילוב של נתיב חשוד עם User-Agent חשוד מעלה משמעותית את רמת הסיכון המחושבת.

כפי שצוין, ה-User-Agent ניתן לזיוף, ולכן הוא נמדד תמיד בהקשר. בקשה עם User-Agent ריק או חריג, היעדר Headers מקובלים של דפדפן (כגון Accept-Language), ותזמון מכני המדויק מדי כל אלה מצטרפים לפרופיל שמבדיל בין כלי Automation לבין משתמש אנושי. עיקרון ה-Correlation הזה הוא שמאפשר זיהוי אמין גם כאשר התוקף מנסה להתחזות.

סינון בוטים לגיטימיים (AllowList)

ניתוח הלוגים חשף תנועה לגיטימית של Crawlers כגון Googlebot, AhrefsBot, Bingbot ו- Meta Web Indexer. כדי שמנגנוני הזיהוי לא יסווגו אותם בטעות כעוינים, הוטמע מנגנון Allowlist המאפשר את מעברם התקין. מנגנון זה מצמצם False Positives, משפר את איכות ה-Telemetry ומונע "זיהום" של

הלוגים בתעבורה לגיטימית. ההפרדה בין Crawlers לגיטימיים לבין פעילות עוינת היא תנאי הכרחי לאיכות ה-Threat Intelligence שמופק מהמערכת.

עם זאת, ה-Allowlist עצמו מהווה משטח תקיפה פוטנציאלי: תוקף שמתחזה ל-Googlebot מקווה לקבל מעבר חופשי. לכן ההכללה ב-Allowlist אינה מבוססת על מחרוזת ה-User-Agent בלבד, אלא על אימות נוסף למשל בדיקה שכתובת ה-IP אכן שייכת לטווחים הרשמיים של הספק. כך נשמר האיזון בין הימנעות מחסימת בוטים מועילים (כמו מנועי חיפוש, שחשובים ל-SEO של האתר) לבין מניעת ניצול ה-Allowlist להתחמקות.

Threat Scoring

מנוע ה-Scoring

מנגנון ה-Threat Scoring נבנה כדי להעריך את רמת הסיכון של כל בקשה. לכל Request מחושב Score בהתאם למספר פרמטרים: סוג הנתיב שנבדק, ה-User-Agent, שיטת ה-HTTP, אינדיקציות ב-Payload, דפוסי תקיפה מוכרים, אינדיקציות Reconnaissance וניסיונות Exploitation. הניקוד מאפשר להבדיל בין סריקה בסיסית, פעילות Automation חשודה, ניסיון Exploitation, ומתקפה בסיכון גבוה.

לוגיקת הניקוד ו-Correlation

הניקוד מחושב לפי חומרת האינדיקציה ופוטנציאל הסיכון שלה, אך הכוח האמיתי טמון ב-Correlation בין אינדיקציות. לדוגמה: בקשת GET בודדת לנתיב .php מקבלת ניקוד נמוך יחסית אולם שילוב של גישה ל-.env, יחד עם User-Agent חשוד, בקשת POST ותדירות גבוהה מוביל לעלייה חדה בניקוד ולסיווג מחמיר. גישה זו מאפשרת לזהות Credential Stuffing, Reconnaissance אוטומטי, Exploit Scanning ו-Web Shell Enumeration גם כאשר כל אינדיקציה בנפרד אינה חד-משמעית.

כדי להמחיש את העיקרון, נבחן דוגמה (Illustrative) של חישוב Score עבור בקשת POST ל-/wp-login.php המגיעה מתשתית ענן עם User-Agent של כלי Automation.

כל אינדיקציה בנפרד אינה מספיקה לחסימה ודאית אולם הצטברותן יחד מובילה ל-Score גבוה ולסיווג High או Critical. זהו ההבדל המהותי בין חוק חסימה בודד לבין מנוע Scoring: המנוע מודד את התמונה המצטברת, ולכן הוא עמיד יותר בפני ניסיונות התחמקות שבהם התוקף משנה מאפיין אחד בכל פעם.

מנוע הסיווג (Classification)

לאחר חישוב הניקוד, כל בקשה סווגה לרמת סיכון. בקשות בסיכון נמוך נרשמו ל-Log בלבד בקשות בסיכון בינוני הופנו ל-Managed Challenge ובקשות בסיכון גבוה נחסמו או נותבו ל-Honeypot. סיווג זה הוא שמתרגם את ה-Score המספרי לפעולה מבצעית:

SCORE RANGE	CLASSIFICATION	SEVERITY	DESCRIPTION
50+	low_confidence_probe	LOW	Basic reconnaissance Low-confidence activities or weak indicators. Minimal risk to the environment.
100+	suspicious_probe	MEDIUM	Suspicious activity Multiple indicators detected. Potential reconnaissance or probing behavior.
180+	active_scanner	HIGH	Active scanning Aggressive scanning or automation detected. High probability of attack attempts.
250+	critical_exploitation_attempt	CRITICAL	Critical exploitation attempt High-risk exploitation behavior detected. Immediate threat to the system.

NOTE: This classification helps prioritize alerts, automate response actions, and improve detection accuracy by identifying the most dangerous threats in real time.

דוגמאות אמיתיות מתוך הלוגים

Active Scanner Detection

```
{
  "path": "/obfuscate.php",
  "score": 180,
  "classification": "active_scanner",
  "matchedRules": [
    "pattern:obfuscate",
    "pattern:.php"
  ]
}
```

Critical Exploitation Attempt

```
{
  "path": "/vendor/phpunit/eval-stdin.php",
  "classification": "critical_exploitation_attempt",
  "score": 290,
  "matchedRules": [
    "pattern:phpunit",
    "pattern:eval-stdin"
  ]
}
```

מערכת ה-Honeypot

מטרות המערכת

מערכת ה-Honeypot נבנתה כחלק משכבת ה-Deception וה-Threat Intelligence. מטרתה אינה רק לחסום תוקפים, אלא ליצור סביבת Interaction מבוקרת המאפשרת לאסוף מודיעין בזמן אמת. המערכת נועדה להטעות תוקפים באמצעות שירותים ונתיבים מזויפים, לאסוף Payloads ו-User-Agents, לזהות מתקפות אוטומטיות, לנתח ניסיונות Credential Stuffing, ולהאריך את זמן הפעילות של התוקף בסביבה מבוקרת לצורך Telemetry. כך הופכת התקיפה ממקור סיכון למקור מודיעין.

תגובות מזויפות ושכבת Deception

במקום להחזיר שגיאות רגילות, המערכת החזירה תגובות מזויפות המדמות שירותים פגיעים, כדי לגרום לתוקף להמשיך באינטראקציה:

שירות מזויף	מטרה
Fake WordPress Login	זיהוי Credential Stuffing ו-Brute Force
Fake ENV File	זיהוי ניסיונות חשיפת Secrets
Fake Adminer Panel	זיהוי Database Enumeration ו-Admin Probing
Fake phpinfo	זיהוי PHP Reconnaissance ו-Exploit Scanning

שכבת ה-Deception אפשרה להאריך את זמן החשיפה של התוקף, לאסוף Payloads ודפוסי תקיפה, לזהות כלי Automation, לשפר את איכות ה-Threat Intelligence ולצמצם את חשיפת שרת המקור.

הרעיון המנחה בבסיס ה-Deception הוא היפוך יחסי הכוחות: בעוד שתוקף אוטומטי מצפה לתגובה דיכוטומית (הנתיב קיים או לא), המערכת מספקת לו תגובה שנראית "מבטיחה" דף Login לכאורה אמיתי, קובץ Environment לכאורה חשוף וכך גורמת לו להמשיך ולהשקיע מאמץ. כל בקשת המשך מספקת מידע נוסף: אילו שדות התוקף מנסה למלא, אילו Payloads הוא שולח, ובאיזה קצב. המידע הזה הופך את ה-Honeypot ממנגנון הגנה פסיבי למקור מודיעין פעיל, מבלי שהתוקף נחשף לשרת האמיתי כלל.

Credential Telemetry

כאשר תוקף ניסה להתחבר דרך דף ה-Login המזויף, נאסף מידע מוגבל לצורכי Telemetry בלבד. מנגנון זה איפשר לזהות קמפיינים של Password Spraying ו-Credential Stuffing, שימוש בסיסמאות נפוצות, Bot Automation וכלים התקפיים מבלי לפגוע בפרטיות.

ההבחנה בין Credential Stuffing לבין Password Spraying משמעותית לניתוח. ב-Credential Stuffing התוקף משתמש בזוגות שם-משתמש/סיסמה שדלפו ממאגרים אחרים, בהנחה שמשתמשים ממחזרים סיסמאות ב-Password Spraying התוקף מנסה מספר סיסמאות נפוצות מול חשבונות רבים, כדי להימנע מנעילת חשבון. דפוס הניסיונות מספר החשבונות מול מספר הסיסמאות וקצב הניסיון מאפשר להבחין בין השניים, וכל אחד מהם מצביע על סוג קמפיין שונה ועל תשתית תקיפה שונה.

שיקולי אבטחה ואתיקה

מערכת ה-Honeypot תוכננה תוך הקפדה על עקרונות Privacy ו-Data Minimization. לא נשמרו סיסמאות מלאות בוצע Masking לפני שמירה נשמר Hash בלבד לצורכי Correlation לא בוצע Replay של Credentials ולא בוצע ניסיון התחברות למערכות צד שלישי. נשמר Telemetry מינימלי בלבד. המערכת נבנתה למטרות הגנה, מחקר ולימוד של טכניקות Detection ו-Deception ולא נעשה בה כל שימוש התקפי או זדוני.

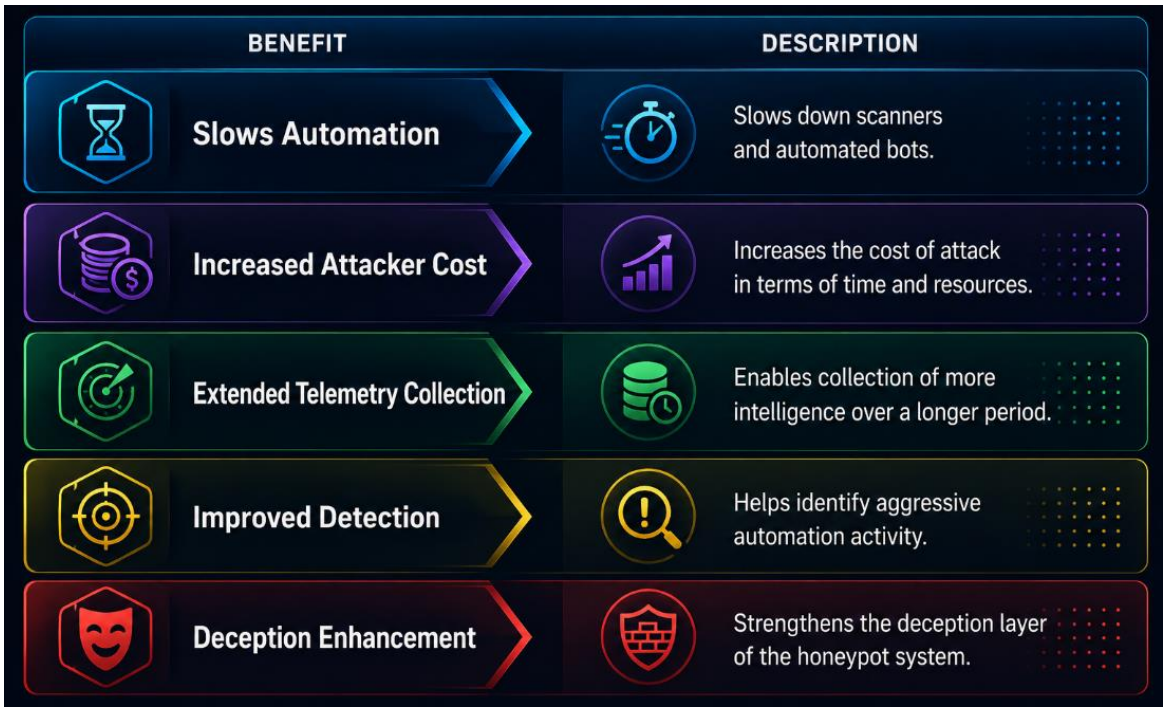
Tarpitting

במסגרת ה-Honeypot הוטמע מנגנון Tarpitting שנועד להאט פעילות אוטומטית ולפגוע ביעילות של סורקים. במקום לחסום מיד בקשות חשודות, המערכת החזירה תגובות בעיכוב אקראי ומבוקר, וכך יצרה עומס על כלי ה-Automation וביזבזה את משאבי התקיפה. גישה זו האטה Scanners, הגדילה את זמן השהיית התוקף בסביבת ה-Honeypot, העלתה את עלות התקיפה במונחי זמן ומשאבים, ופגעה ביעילות של קמפיינים מבוססי Mass Scanning תוך איסוף Telemetry נוסף לאורך זמן.

גישה זו אפשרה:

- להאט Scanners אוטומטיים
- להגדיל את זמן הפעילות של תוקפים בסביבת ה-Honeypot
- להגדיל את עלות התקיפה מבחינת זמן ומשאבים
- לפגוע ביעילות של Mass Scanning Campaigns
- לאסוף Telemetry נוסף לאורך זמן
- לזהות כלי Automation אגרסיביים

ובתמונה:



איסוף מודיעין Intelligence Collection

מערכת ה-Honeypot ומנגנוני ה-Telemetry אפשרו לאסוף מידע מבצעי בזמן אמת מתוך הפעילות העוינת. הנתונים נאספו באמצעות Cloudflare Workers, Firewall Events, Security Events ו-Worker Observability Logs. הטבלה הבאה מסכמת את סוגי הנתונים שנאספו:

סוג נתון	תיאור
IP Address	כתובת המקור של הפעילות
Country	מדינת המקור של הבקשה
ASN	מספר ה-Autonomous System של התשתית
User-Agent	זיהוי דפדפן, בוט או כלי Automation
Request Path	הנתיב שנבדק או הותקף
HTTP Method	סוג הבקשה (GET / POST)
Headers	כותרות HTTP מלאות ל-Fingerprinting
Request Body	חלקי Payload מבקשות חשודות
Attack Score	ה-Threat Score שחושב
Classification	סיווג רמת הסיכון
Matched Rules	חוקי Detection שהופעלו
Fingerprint	מזהה ל-Correlation בין בקשות

בנוסף, המערכת אספה מידע מתקדם על XMLRPC Abuse, ניסיונות Credential Stuffing, User-Agent Rotation, Bot Obfuscation, Automation Frameworks, WordPress Fingerprinting, Campaign Markers ו-Exploitation Payloads. בין היתר זוהו ניסיונות שימוש ב-WordPress API Methods כגון metaWeblog.newPost, לצד Payloads חשודים של POST. המערכת הצליחה לבצע Correlation בין אותו IP, החלפת User-Agent, דפוסי Payload ותזמון התקיפה, וכך לזהות פעילות Automation רחבת היקף.

נתיב xmlrpc.php/ ראוי להסבר קצר: זהו ממשק XML-RPC של WordPress, שבעבר נוצל לרעה לצורך הגברת מתקפות (Amplification) ו-Brute Force מרוכז באמצעות שיטות כגון system.multicall. ניסיונות גישה אליו, ובמיוחד בקשות POST עם מבני XML מורכבים, הם אינדיקציה מובהקת ל-Automation ולא לגלישה אנושית. ה-Correlation בין כתובת IP יחידה לבין מספר User-Agents שונים לאורך זמן (User-Agent Rotation) הוא אחד הסימנים החזקים ביותר לניסיון התחמקות מ-Detection, והוא מאפשר לקשר בקשות לכאורה-נפרדות לאותו קמפיין תקיפה.

אינטגרציית WAF

עקרון האינטגרציה

לאחר ניתוח ה-Telemetry וה-IOC, עודכנו חוקי ה-WAF כך שיזהו ויחסמו דפוסים שחזרו באופן עקבי. בניגוד ל-WAF מסורתי המתמקד בחסימה בלבד, המערכת שילבה בין Detection, Threat Classification, Honeypot Routing, Deception ו-Telemetry. כך הפך ה-Threat Intelligence שנאסף בזמן אמת למדיניות הגנה מבצעית ב-Edge, ובקשות חשודות יכלו להיחסם, לעבור Managed Challenge, להיות מנותבות ל-Honeypot, או להיכנס לתהליך Telemetry.

חוקי WAF לדוגמה

Pattern	מטרת הזיהוי
.php	Reconnaissance-ו PHP Enumeration
shell	Web Shell Discovery
phpunit	ניסיונות RCE מוכרים
adminer	Database Enumeration
.git	מניעת Git Repository Exposure
.env	חסימת ניסיונות חשיפת Secrets
xmlrpc.php	WordPress Automation
wp-login.php	Credential Stuffing

[דוגמאות לחוקי WAF שהוטמעו]

ניתוב ל-Honeypot, Managed Challenge ו-Rate Limiting

חלק משמעותי מהפעילות התמקד בנתיבי WordPress נפוצים, ובמיוחד /wp-login.php ו-/xmlrpc.php. במקום לחסום מיד את כל הבקשות לנתיבים אלו, בוצעה התאמה ייעודית שאיפשרה ניתוב מבוקר של בקשות חשודות אל ה-Honeypot והפיכתן למקור מודיעין. עבור בקשות שסווגו כחשודות אך ללא ודאות מלאה, הופעל Managed Challenge של Cloudflare, אשר סינן תעבורה אוטומטית ובוטים תוך שמירה על חוויית משתמש תקינה. במקביל הופעל Rate Limiting עבור פעילות שתאמה Credential Stuffing, Brute Force, POST Flooding ו-XMLRPC Abuse.

השפעה מבצעית

האינטגרציה בין ה-WAF לבין ה-Honeypot יצרה שכבת הגנה מבצעית שעצרה איומים ב-Edge, צמצמה את חשיפת שרת המקור, שיפרה את ה-Visibility על פעילות עוינת, אספה Threat Intelligence בזמן אמת, וזיהתה תשתיות תקיפה אוטומטיות. כך הודגם כיצד ניתן להפוך מערכת WAF ממנגנון חסימה סטטי לפלטפורמת Detection ו-Threat Intelligence דינמית.

יתרון מהותי בגישה זו הוא לולאת המשוב (Feedback Loop): ה-Telemetry שנאסף ב-Honeypot מזין את חוקי ה-WAF, חוקי ה-WAF מנתבים תעבורה נוספת ל-Honeypot, וזה מזין שוב את הניתוח. כך המערכת משתפרת עם הזמן ומתאימה את עצמה לדפוסי תקיפה חדשים, במקום להישאר תלויה ברשימת חוקים סטטית שנקבעה מראש.

Risk Assessment

מתודולוגיה

הערכת הסיכון התבססה על שילוב בין Likelihood (הסתברות התקיפה), Impact (רמת הנזק האפשרית) ו-Threat Severity (חומרת הפעילות שנצפתה). נשקלו פוטנציאל הפגיעה, סבירות ההתממשות, רמת החשיפה, יכולת ה-Exploitation, ההשפעה על הזמינות והפוטנציאל לחשיפת מידע רגיש.

הגישה המקובלת מציגה את הסיכון כמכפלה של הסתברות בהשפעה. אירוע בעל הסתברות גבוהה והשפעה נמוכה (כמו סריקת Reconnaissance שגרתית) מדורג נמוך-בינוני אירוע בעל הסתברות נמוכה אך השפעה קריטית (כמו הרצת קוד מרחוק מוצלחת) מדורג גבוה למרות נדירותו. הצלבת שני הצירים הללו מאפשרת לתעדף את מאמצי ההגנה אל עבר התרחישים שבהם שילוב ההסתברות וההשפעה מסוכן במיוחד, במקום לפזר משאבים באופן אחיד.

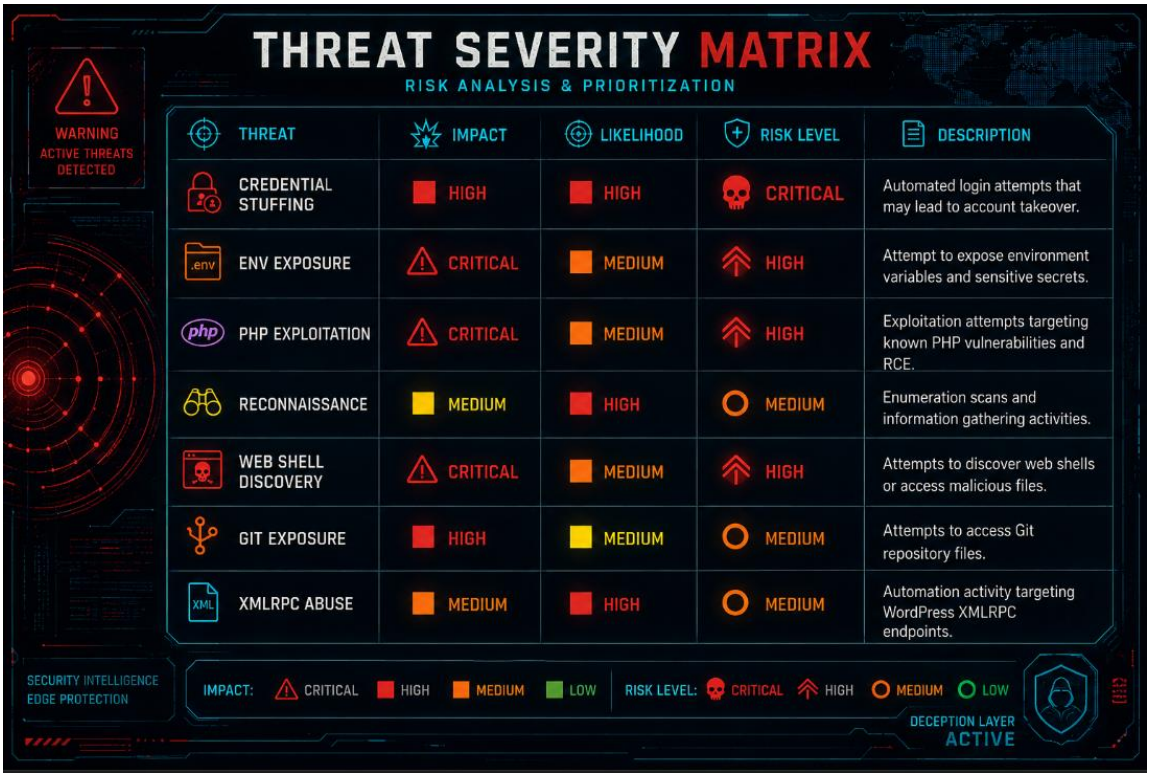
השפעה עסקית

למרות שהאתר אינו מבוסס WordPress או PHP, פעילות ה-Reconnaissance וה-Mass Scanning יצרה סיכונים ממשיים עומס מיותר על שרת המקור, ניסיונות גישה לנתיבים רגישים, סיכון לחשיפת מידע, פוטנציאל לניצול חולשות עתידיות, Credential Abuse, ופגיעה אפשרית בזמינות במקרה של Flooding. השימוש בתשתיות ענן וב-Automation Frameworks העלה את רמת התחכום ואת ה-Scale של הפעילות.

מעבר לנזק הטכני הישיר, קיימת גם עלות עסקית עקיפה משאבי שרת המבוזבזים על טיפול בתעבורה עוינת, פגיעה אפשרית בחוויית המשתמש בזמן עומס, וסיכון מוניטיני במקרה של דליפת מידע. עבור ארגון, גם תקיפה "לא ממוקדת" שנכשלת עלולה לגרור עלויות חקירה, ניטור ותגובה. לכן ההשקעה במניעה ובזיהוי מוקדם ב-Edge משתלמת גם כאשר לא אירעה פריצה בפועל.

הפחתת סיכון והערכה כוללת

הטמעת ה-Honeypot יחד עם WAF איפשרה לצמצם משמעותית את הסיכון באמצעות Edge Protection, Threat Classification, Managed Challenges, Rate Limiting, Honeypot Routing, Deception ואיסוף Threat Intelligence. אף שלא זוהתה פריצה מוצלחת, רמת הסיכון הכוללת הוערכה כ-High בשל היקף הפעילות האוטומטית, השימוש בתשתיות מבוזרות, ניסיונות Exploitation מוכרים ו-Credential Abuse מתמשך. המערכת הצליחה להפחית את ה-Exposure של שרת המקור ולשפר את יכולות ה-Detection.



תוצאות הפרויקט

הישגים מרכזיים

במסגרת הפרויקט הוקמה מערכת Detection ו-Deception מבוססת Cloudflare אשר זיהתה, ניתחה והטעתה פעילות עוינת בזמן אמת. המערכת עצרה ניסיונות Reconnaissance ו-Enumeration ב-Edge; זיהתה Credential Stuffing ו-XMLRPC Abuse באמצעות POST ל-/xmlrpc.php; זיהתה שימוש ב-WordPress API Methods כגון metaWeblog.newPost; מנעה גישה ישירה לשרת המקור באמצעות ניתוב ל-Honeypot; אספה Threat Intelligence מ-Worker Telemetry ו-Firewall Events; זיהתה תשתיות תקיפה מבוססות VPS וענן; וזיהתה User-Agent Rotation ודפוסי Bot Obfuscation. בין האירועים הבולטים זוהה מעבר מ-Reconnaissance בסיסי לניסיונות Exploitation פעילים, כולל שימוש ב-Go-http-client ובכלי Automation נוספים, ובקשות POST חוזרות ל-/xmlrpc.php עם החלפת User-Agent מאותו מקור דפוס מובהק של ניסיון לעקוף Detection.

המערכת הצליחה:

- לעצור ניסיונות Reconnaissance וסריקות Enumeration בשכבת ה-Edge
- לזהות ניסיונות Credential Stuffing ו-XMLRPC Abuse באמצעות POST Requests ל-/xmlrpc.php
- לזהות ניסיונות שימוש ב-WordPress API Methods כגון:
  - metaWeblog.newPost
- לזהות Payloads חשודים ו-Automation Frameworks
- למנוע גישה ישירה לשרת המקור באמצעות ניתוב Requests חשודים ל-Honeypot
- לאסוף Threat Intelligence בזמן אמת מתוך Worker Telemetry ו-Firewall Events
- לזהות תשתיות תקיפה מבוססות VPS ו-Cloud Providers ציבוריים
- לזהות User-Agent Rotation ודפוסי Bot Obfuscation
- לבצע Correlation בין Payloads, Attack Timing ו-Repeated Exploitation Attempts
- ליצור סביבת Deception מלאה אשר הצליחה לגרום לתוקפים לחשוף Payloads, Attack Patterns ו-Credential Attempts

במהלך הניתוח זוהו מספר אירועים משמעותיים אשר הצביעו על מעבר מפעילות Reconnaissance בסיסית לניסיונות Exploitation פעילים. בין היתר זוהו:

- גישה לנתיבי WordPress מוכרים לצורכי Fingerprinting
- שימוש ב-Go-http-client ובכלי Automation נוספים
- מספר בקשות POST ל-/xmlrpc.php אשר כללו XMLRPC Abuse וניסיונות Credential Abuse
- שימוש ב-Campaign Markers ו-Payload Patterns חוזרים
- החלפת User-Agent מאותו מקור תקיפה במטרה לעקוף מנגנוני Detection

יתרונות

המערכת סיפקה מספר יתרונות מבצעיים משמעותיים:

- **Edge Protection** - עצירת פעילות עוינת בשכבת ה-Edge לפני הגעתה לשרת המקור.
- **Reduced Attack Surface** - צמצום משמעותי של שטח התקיפה ושל חשיפת שירותי ה-Web.
- **Improved Visibility** - שיפור יכולות ה-Visibility וה-Telemetry על פעילות עוינת בזמן אמת.
- **Real-Time Detection** - יכולת זיהוי, Classification ו-Threat Scoring בזמן אמת.
- **Threat Intelligence Collection** - איסוף IOC, Payloads, User-Agents ודפוסי תקיפה מתוך פעילות אמת.
- **Deception Capabilities** - שימוש בטכניקות Honeypot ו-Fake Services לצורך הארכת זמן החשיפה של תוקפים ואיסוף מידע נוסף.
- **Credential Intelligence** - זיהוי ניסיונות Credential Abuse ו-XMLRPC Authentication Attempts.
- **Bot & Automation Detection** - זיהוי User-Agent Rotation, Automation Frameworks ודפוסי Bot Activity.
- **Low Deployment Cost** - הטמעה מהירה בעלות נמוכה יחסית באמצעות Cloudflare Workers ו-WAF.
- **Rapid Deployment** - יכולת פריסה מהירה ללא צורך בתשתיות מורכבות או שרתים ייעודיים.
- **Scalable Architecture** - יכולת הרחבה עתידית למנגנוני AI Detection, Threat Correlation ו-Behavioral Analytics.

מגבלות וכיווני המשך

לצד הישגיו, חשוב להכיר במגבלות המערכת. ראשית, זיהוי מבוסס דפוסים ו-Allowlist דורש תחזוקה שוטפת דפוסי תקיפה משתנים, וכלי Automation חדשים מופיעים כל הזמן. שנית, תוקף מתוחכם ובעל מוטיבציה גבוהה (Targeted) עשוי להאט בכוונה את קצב פעילותו ולחקות התנהגות אנושית כדי לחמוק מ-Scoring מבוסס תדירות. שלישית, ה-Deception יעיל בעיקר מול Automation הזדמנותי, ופחות מול תקיפה ידנית ממוקדת.

כיווני ההמשך הטבעיים כוללים שילוב Behavioral Analytics לזיהוי חריגות לאורך זמן, הוספת מנגנוני Machine Learning ל-Scoring דינמי המתעדכן מאליו, העשרת ה-Telemetry במקורות (Feeds) חיצוניים שלThreat Intelligence, והרחבת שכבת ה-Deception לשירותים נוספים מעבר ל-WordPress. כל אלה ניתנים למימוש על גבי אותה ארכיטקטורת Edge, ללא צורך בשינוי מבני.

Legitimate Bot Detection

This section identifies trusted crawlers such as Googlebot, Bingbot and AhrefsBot. Legitimate indexing traffic is logged separately and bypasses the honeypot workflow:

```
const legitBotRules = [
  { name: "Googlebot", category: "search_engine", match: "googlebot" },
  { name: "Bingbot", category: "search_engine", match: "bingbot" },
  { name: "AhrefsBot", category: "seo_crawler", match: "ahrefsbot" }
];

const matchedLegitBot = legitBotRules.find(
  bot => lowerUA.includes(bot.match)
);

if (matchedLegitBot) {
  return fetch(request);
}
```

Exact Match Detection

High-confidence attack paths commonly targeted during reconnaissance and exploitation attempts:

```
const exactBadPaths = [
  "/wp-login.php",
  "/xmlrpc.php",
  "/adminer.php",
  "/phpinfo.php",
  "/.env",
  "/.git/config"
];

if (exactBadPaths.includes(path)) {
  addScore(100, "exact_bad_path");
}
```

Pattern-Based Detection

Searches for suspicious keywords associated with known exploitation techniques:

```
const patterns = [
  { value: "xmlrpc", score: 60 },
  { value: "phpunit", score: 90 },
  { value: "shell", score: 90 },
  { value: ".env", score: 100 }
];

for (const p of patterns) {
  if (path.includes(p.value)) {
    addScore(p.score, `pattern:${p.value}`);
  }
}
```

User-Agent Intelligence

Identifies automation frameworks and offensive security tools:

```
if (
  lowerUA.includes("curl") ||
  lowerUA.includes("sqlmap") ||
  lowerUA.includes("nikto") ||
  lowerUA.includes("masscan")
) {
  addScore(90, "suspicious_user_agent");
}
```

Credential Honeypot Telemetry

Collects credential attack telemetry while masking sensitive data:

```
credentialTelemetry = {
  username: sanitizeText(username),
  maskedPassword: maskPassword(password),
  passwordHash: await sha256(password)
};

addScore(150, "credential_honeypot_submission");
```

Threat Classification Engine

Maps the accumulated score to a threat category:

```
if (score >= 300) {
  classification = "credential_attack_or_critical_exploitation";
} else if (score >= 250) {
  classification = "critical_exploitation_attempt";
} else if (score >= 180) {
  classification = "active_scanner";
}
```

Tarpitting

Introduces artificial delays for high-risk requests:

```
if (score >= 180) {
  await sleep(randomBetween(1500, 5000));
}
```

Threat Telemetry Collection

Logs suspicious activity for threat intelligence and correlation:

```
console.log(JSON.stringify({
  type: "HONEYPOT_HIT",
  ip,
  country,
  path: url.pathname,
  score,
  classification,
  matchedRules
}));
```

Deception Layer

Returns fake responses that simulate vulnerable services:

```
function generateFakeResponse(path, method) {
  if (path.includes("wp-login")) {
    return wordpressLoginPage("");
  }

  if (path.includes(".env")) {
    return "APP_ENV=production";
  }
}
```

סיכום

במסגרת Operation GhostEdge הוקמה מערכת הגנת Web מודרנית המבוססת על Cloudflare WAF, Cloudflare Workers, ארכיטקטורת Threat IntelligenceHoneypot וטכנולוגיית Deception. המערכת אפשרה לזהות, לנתח ולהטעות תוקפים בזמן אמת, תוך הגנה על שרת המקור וצמצום משמעותי של ה-Exposure. הפרויקט הדגים כיצד ניתן להשתמש ביכולות Edge Security להקמת סביבת Detection ו-Deception מודרנית המספקת Prevention, Detection, Threat Intelligence, Telemetry ו-Deception כאחד.

מעבר לכך, הפרויקט הוכיח כי גם תשתית Web קטנה יכולה לאסוף מודיעין סייבר מבצעי בזמן אמת, באמצעות שילוב של Threat Scoring, Worker Telemetry, Honeypot Routing, אינטגרציית WAF וזיהוי בזמן אמת.

ממצאי החקירה הצביעו על פעילות Automation מתקדמת: WordPress Fingerprinting, XMLRPC Abuse, Credential Stuffing, ניסיונות Exploitation, Bot Obfuscation ו-User-Agent Rotation - והמערכת הצליחה לגרום לכלי התקיפה לחשוף Payloads אמיתיים, Attack Workflows ו-Credential Attempts בתוך סביבה מבוקרת.

המסקנה המרכזית היא ששילוב בין Cloudflare Edge Security, ארכיטקטורת Honeypot ו-Threat Intelligence מאפשר ליצור שכבת Detection ו-Deception מבצעית גם ללא תשתיות Enterprise מורכבות - תוך שמירה על Deployment מהיר, Visibility גבוהה ועלות נמוכה יחסית. הפרויקט מהווה הדגמה מעשית לכך שהגנה אפקטיבית אינה נמדדת רק ביכולת לחסום, אלא גם ביכולת ללמוד מן התוקף ולהפוך את התקיפה עצמה לנכס מודיעיני.

על המחבר

גיא תותחני, עוסק ב-CTI ו-SecOps ניתוח תוקפים, Threat Hunting, Honeypots, Deception, Technologies, אבטחת יישומי Web ופיתוח מנגנוני Detection. מתמקד בחיבור בין עולמות ההגנה וההתקפה לצורך הפקת מודיעין סייבר מעשי ושיפור יכולות ההגנה של ארגונים.

מאמר זה מבוסס על מחקר ויישום מעשי של טכניקות Detection, Threat Intelligence ו-Deception בסביבת Web מודרנית.

אשמח לענות על כל שאלה ואם צרכים כל עזרה בנושא זה! זמין ב-linkedin

מקורות מידע

- T1595 - Active Scanning
- T1190 - Exploit Public-Facing Application
- T1505 - Server Software Component (Web Shell)
- T1590 - Gather Victim Network Information
- curl
- wget (GNU)
- sqlmap
- nikto
- Brute-force attack
- Reverse proxy
- Autonomous System (ASN)
- Credential stuffing
- Cloudflare Workers - תיעוד
- Cloudflare Workers - מוצר דף
- Cloudflare WAF - תיעוד
- Cloudflare WAF - Custom Rules / Managed Challenge
- Cloudflare WAF - Managed Rules
- Cloudflare - Rate Limiting Rules
- MITRE ATT&CK - הבית דף
- T1110 - Brute Force
- Masscan
- Honeypot (computing)
- Tarpit (networking)

מבט מבפנים על המנוע הקרנלי של Predator

מאת ניר אברהם

תקציר מנהלים

זה הוא החלק השלישי ואחרון בסדרת המאמר על Predator, במחקר הקודם שלנו, תיעדנו כיצד תוכנת הריגול Predator מתחמקת מבדיקות אנטי-אנליזה ב-iOS ועוקפת את מחווני ההקלטה של iOS. המחקר חשף מה Predator עושה - אך לא כיצד היא משיגה את הגישה העמוקה למערכת שנדרשת כדי לבצע זאת. חלק זה עונה על השאלה הזו. מומלץ לקרוא את חלק א' וחלק ב' של סדרת מאמרים זו.

באמצעות המשך הנדסה לאחור של דגימות Predator ל-iOS, אנו מתבססים על הדוח הראשון שלנו וחושפים כיצד מנוע ניצול הקרנל של Predator מניע את יכולות המעקב שלה. מנוע זה מעולם לא דווח בעבר - עד עכשיו.

שרשרת הניצול שתוצג במאמר זה מכוונת לגרסאות iOS שקודמות ל-17 ולמכשירים עד דור A16. ההוספה של SPTM (Secure Page Table Monitor) על ידי Apple במכשירי A15, שמעבירה את ניהול טבלאות הדפים אל EL2, מהווה מנגנון הקשחה ארכיטקטוני משמעותי נגד טכניקות שינוי קוד הקרנל המתוארות כאן.

הממצאים המרכזיים כוללים:

- **FDGuardNeonRW** - פרימיטיב קריאה/כתיבה לקרנל, Kernel R/W, שמשתמש ברגיסטרים וקטוריים של ARM NEON כערוץ נתונים סמוי לצורך קריאה וכתיבה של זיכרון קרנל שרירותי.
- **עקיפת PAC באמצעות חיפוש gadgets ב-JavaScriptCore** - Predator מחפשת במסגרת JavaScriptCore של Apple עצמה רצף ספציפי של 20 בתים של פקודות ARM64, כדי לזייף מצביעים חתומים ב-PAC.
- **Cache חתימות PAC בן 256 רשומות** - מצביעים חתומים שחושבו מראש ומאונדקסים לפי בית הכתובת, המאפשרים callbacks של hooks בזמן אמת ללא השהיה קריפטוגרפית.
- **RWTransfer** - מנגנון להעברת יכולות קריאה/כתיבה לקרנל בין תהליכים, תוך שימוש ב-file descriptors מוגנים ובמניפולציה של Mach ports.
- **callFunc** - מסגרת להרצת פונקציות מרוחקות, אשר משתלטת על מצב thread באמצעות הודעות Mach exception.
- **21 דגמי מכשירים נתמכים**, החל מ-iPhone XS ועד iPhone 14 Pro Max, המאורגנים ב-5 מחלקות מכשירים.

רקע: מאפייני האבטחה ש-Predator חייבת להביס

(PAC) Pointer Authentication Codes

החל משבב A12 (iPhone XS, 2018), Apple הציגה את PAC - מאפיין חומרה שמוסיף חתימות קריפטוגרפיות למצביעי קוד. לפני שהמעבד עוקב אחר מצביע לפונקציה או אחר כתובת חזרה, הוא מאמת את החתימה. אם תוקף משחית מצביע, בדיקת החתימה נכשלת והמעבד מעלה חריגה.

(KASLR) Kernel Address Space Layout Randomization

iOS מבצעת רנדומיזציה לכתובת הבסיס של הקרנל בכל אתחול, באמצעות KASLR. המשמעות היא שכל קוד הקרנל וכל מבני הנתונים הסטטיים מוזזים בהיסט בלתי צפוי. גם אם לתוקף יש חולשת קרנל, עליו קודם לקבוע את ה-slide הזה כדי לאתר את המבנים שהוא רוצה לבצע בהם מניפולציה. לשם כך נדרש פרימיטיב אמין לקריאת זיכרון קרנל - וזה בדיוק מה ש-FDGuardNeonRW מספק.

ממצא 1: FDGuardNeonRW - גישה לזיכרון הקרנל דרך רגיסטרים וקטוריים

פענוח השם

שם המחלקה **FDGuardNeonRW** מקודד את כל הטכניקה שלה:

Component	Meaning
FD	File Descriptor – the exploit involves the `change_fdguard_np` syscall
Guard	FD guard manipulation to set up the kernel primitive
Neon	ARM NEON vector registers serve as the data channel
RW	Provides both read and write access to kernel memory

זהו פרימיטיב הליבה שמניע את כל יתר הרכיבים ב-Predator. כל hook, כל הזרקה לתהליך, וכל עקיפת PAC תלויים בסופו של דבר ביכולת של FDGuardNeonRW לקרוא ולכתוב זיכרון קרנל שרירותי.

ערוץ הנתונים של NEON

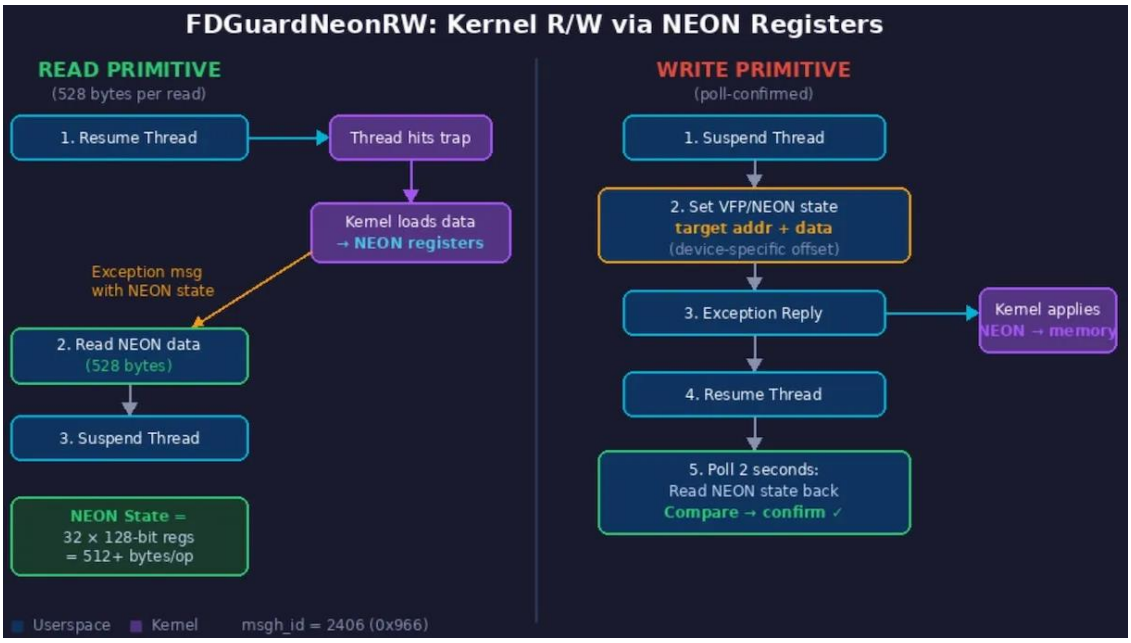
ARM NEON הוא סט של רגיסטרים וקטוריים ברוחב 128 ביט, V0-V31, המשמשים בדרך כלל לפעולות SIMD - חישוב מקבילי על מספר ערכים בו-זמנית. רגיסטרים אלה הם חלק ממצב ה-thread שהקרנל שומר ומשחזר במהלך context switches.

Predator מנצלת זאת באמצעות טכניקה מוכרת היטב: ה-APIs של Mach בשם thread_get_state ו-thread_set_state מאפשרים לקרוא ולכתוב את מצב הרגיסטרים המלא של thread, כולל רגיסטרי NEON, מ-thread אחר שיש לו הרשאות port מתאימות. גישת מניפולציית מצב ה-thread הזו שימשה בשרשראות ניצול של iOS לפחות מאז 2017.

החידוש של Predator הוא ניתוב נתוני קרנל דרך בנק רגיסטרי NEON באופן ספציפי - 32 רגיסטרים וקטוריים בגודל 128 ביט מספקים 528 בתים בכל קריאת thread_get_state, כולל FPSR, FPCR וריפוד יישור, לעומת 272 בתים בלבד הזמינים דרך מצב הרגיסטרים הכלליים.

הערוץ הזה תלוי ברכיב בצד הקרנל שהוקם על ידי שלב הניצול הראשוני - רגיסטרי NEON משמשים כאמצעי ההעברה, אך קריאות וכתיבות זיכרון הקרנל בפועל מבוצעות על ידי קוד שרץ בהרשאות קרנל. הפרטים המלאים של payload זה בצד הקרנל נמצאים מחוץ לתחום הבלוג הזה.

תרשים זרימה המציג את פרימיטיבי הקריאה והכתיבה:



[איור 1: פרימיטיבי הקריאה והכתיבה של FDGuardNeonRW, המשתמשים ברגיסטרי NEON כערוץ נתונים לקרנל ברוחב פס גבוה]

פרימיטיב קריאה - 528 בתים בכל קריאה

פעולת הקריאה מסתמכת על נתיב קוד בקרנל שהותקן על ידי שלב הניצול הראשוני. נתיב קוד זה טוען נתונים מכתובת קרנל נשלטת אל רגיסטרי NEON, ולאחר מכן מגיע לפקודת trap. ה-thread היעד מחודש, מריץ את נתיב הקוד הזה, ומפעיל את ה-trap. מטפל החריגות של Predator מקבל הודעת Mach exception, המוצגת באיור 3 ומזוהה על ידי msgh_id 2406 / 0x966, המכילה את מצב ה-NEON עם נתוני הקרנל. עד 10 ניסיונות חוזרים מטפלים במרוצי תזמון. כל קריאה מחזירה 528 בתים.

פרימיטיב כתיבה - מאומת באמצעות polling

פעולת הכתיבה הופכת את הערוץ. Predator משהה את ה-thread היעד, קוראת את מצב ה-VFP/NEON הנוכחי כקו בסיס, משנה אותו עם כתובת היעד והנתונים במיקום תלוי-מכשיר, ולאחר מכן שולחת תשובת exception עם המצב ששונה. הקרנל משחזר את מצב ה-NEON ששונה אל תוך ה-saved context של ה-thread. כאשר ה-thread ממשיך לרוץ, נתיב הקוד של הניצול בקרנל קורא את הערכים מרגיסטרי NEON וכותב אותם לכתובת הקרנל היעד.

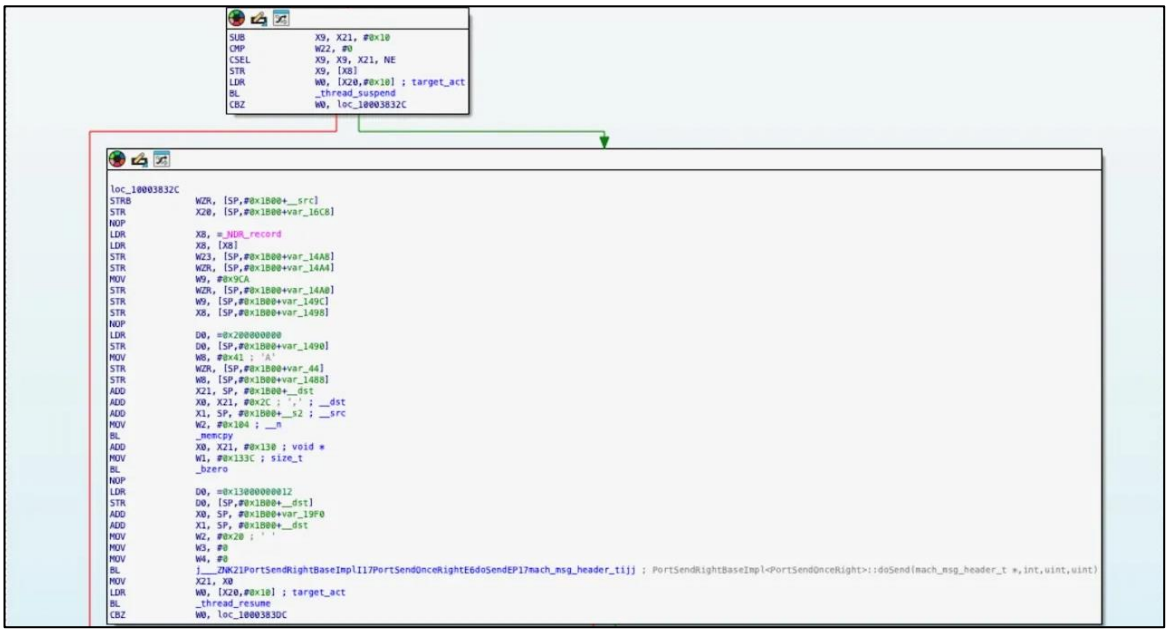
Predator מבצעת polling עד 3 שניות, ומשווה את מצב ה-NEON שנקרא בחזרה מול הנתונים שנשלחו כדי לוודא שהכתיבה הצליחה.

למה רגיסטרי NEON?

מצב NEON הוא:

- **בעל קיבולת גדולה** - יותר מ-512 בתים בכל פעולה.
- **מנוהל על ידי הקרנל** - חלק מהטיפול הרגיל במצב thread.
- **גלוי דרך exceptions** - נכלל בהודעות Mach exception.
- **דו-כיווני** - אותו מנגנון משמש לקריאות ולכתיבות.

פרימיטיב הכתיבה נראה כך:



[איור 2: תצוגת graph של IDA עבור פרימיטיב הכתיבה של NEON - thread_suspend, תשובת exception עם NDR_record, doSend, thread_resume, ולולאת polling של 2 שניות]

```
63    v11 = 10;
64    memset(__src, 0, sizeof(__src));
65    do
66    {
67      __asm { SVC             0 }
68      if ( *v5 )
69      {
70        usleep(0);
71        usleep(0x32u);
72        usleep(0x32u);
73        usleep(0x32u);
74        usleep(0x32u);
75      }
76      v16 = *v3;
77      bzero(v25, 0x1474u);
78      *(_QWORD *)&msg.msgh_bits = v27;
79      msg.msgh_remote_port = v28;
80      msg.msgh_local_port = v16;
81      bzero(&msg.msgh_voucher_port, 0x1478u);
82      msg.msgh_size = 5256;
83      mach_msg(&msg, 258, 0, 0x1488u, v16, 0x3E8u, 0);
84      v25[0] = 0;
85      p_msg = &msg;
86      msgh_remote_port = msg.msgh_remote_port;
87      if ( (msg.msgh_remote_port + 1 > 1 || msg.msgh_local_port - 1 <= 0xFFFFFFFD) && msg.msgh_id == 2406 && v23 == 65 )
88      {
89        memcpy(__dst, v24, sizeof(__dst));
90        memcpy(__src, __dst, sizeof(__src));
91        if ( msg.msgh_remote_port - 1 < 0xFFFFFFFE )
92          break;
93      }
94      else
95      {
96        usleep(0);
97        v29[0] = msg.msgh_remote_port;
98        sub_100009EC8(v29);
99        msgh_remote_port = 0;
100       memset(__src, 0, sizeof(__src));
101     }
102     ThreadPortBase<ThreadPortWeak>::getNeonState(&msg, v10);
103     usleep(0);
104     usleep(0x32u);
105     __asm { SVC             0 }
106     if ( *v5 )
107     {
108       usleep(0);
109       usleep(0x32u);
110       usleep(0x32u);
111       usleep(0x32u);
112       usleep(0x32u);
113     }
114     --v11;
115   }
```

[איור 3: פרימיטיב קריאת NEON לאחר decompilation - לולאת הניסיונות החוזרים (v11 = 10), קבלת mach_msg, אימות של msgh_id == 2406 ושל ספירת מצב VFP השווה ל-65, ולבסוף thread_suspend]

ממצא 2: ציד PAC gadgets ב-JavaScriptCore

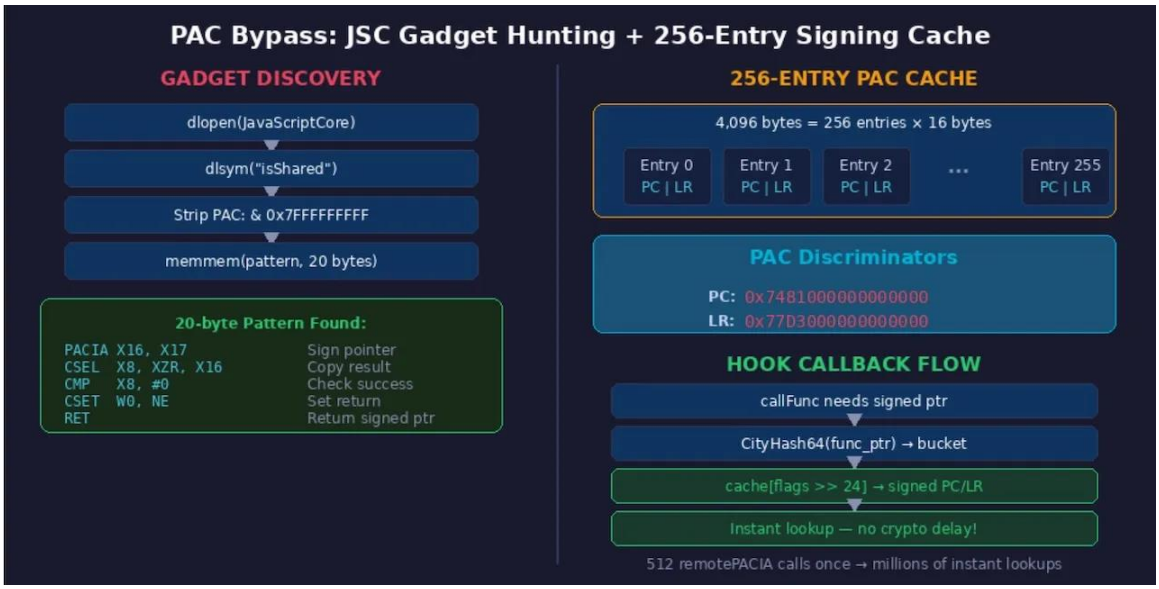
הבעיה

כדי להתקין hooks בתהליכי מערכת, Predator צריכה להסיט את הרצת הפונקציות - כלומר לשנות מצביעי קוד. במכשירים שבהם PAC מופעל, כלומר iPhone XS ומעלה, כל מצביע קוד נושא חתימה קריפטוגרפית. Predator חייבת לזייף חתימות תקפות עבור המצביעים שאליהם היא מנתבת מחדש.

הפתרון: השאלת הקוד של Apple עצמה

במקום להביא מימוש חתימה משלה, Predator מחפשת במסגרת JavaScriptCore של Apple רצף קוד קיים שמבצע חתימת PAC עם קלטים שניתנים לשליטה.

פונקציית היעד היא JSC::JSArrayBuffer::isShared() - פונקציה שקיימת בכל מכשיר iOS כחלק ממנוע ה-JavaScript של Safari:



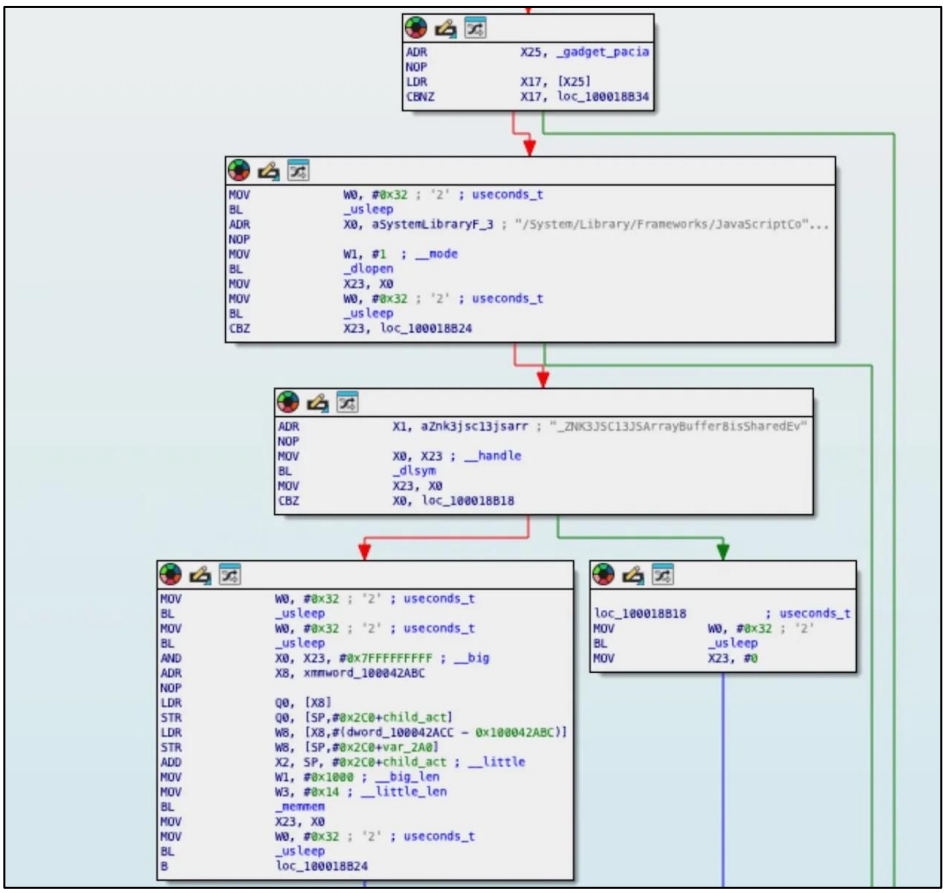
[איור 4: עקיפת PAC באמצעות ציד gadgets ב-JSC ו-cache חתימות שחושב מראש בן 256 רשומות]

ה-gadget בן 20 הבתים

Predator מחפשת בתוך 0x1000 בתים מהסימבול isShared באמצעות memmem() אחר התבנית המדויקת הבאה באורך 20 בתים:

Bytes	ARM64 instruction	Purpose
30 02 C1 DA	PACIA X16, X17	Sign pointer in X16 using X17 as context
E8 03 90 9A	CSEL X8, XZR, X16, EQ	Copy result if valid
1F 01 00 F1	CMP X8, #0	Check if signing succeeded
E0 07 9F 1A	CSET W0, NE	Set return value
C0 03 5F D6	RET	Return signed pointer

פקודת PACIA X16, X17 היא המפתח: היא חותמת את המצביע שנמצא ב-X16 באמצעות X17 כהקשר או discriminator, תוך שימוש במפתח PAC של החומרה. באמצעות שליטה ב-X16 וב-X17, Predator יכולה לזייף חתימות עבור מצביעים שרירותיים:



[איור 5: תצוגת graph של IDA עבור remotePACIA - זרימת ציד ה-gadget: בדיקת cache gadget_pacia ← dlopen(JavaScriptCore) ← dlsym(isShared) ← memmem(pattern, 0x14)]

```
132     v18 = gadget_pacia;
133     if ( !gadget_pacia )
134     {
135       usleep(0x32u);
136       v19 = dlopen("/System/Library/Frameworks/JavaScriptCore.framework/JavaScriptCore", 1);
137       usleep(0x32u);
138       if ( v19 )
139       {
140         v20 = (unsigned __int64)dlsym(v19, "_ZNK3JSC13JSArrayBuffer8isSharedEv");
141         usleep(0x32u);
142         if ( v20 )
143         {
144           usleep(0x32u);
145           *(_QWORD *)child_act = xmmword_100042ABC;
146           v34 = -698416192;
147           v19 = memmem((const void *)(v20 & 0x7FFFFFFFFFLL), 0x1000u, child_act, 0x14u);
148           usleep(0x32u);
149         }
150         else
151         {
152           v19 = 0;
153         }
154       }
155       gadget_pacia = (__int64)v19;
```

[איור 6: חיפוש ה-gadget לאחר dlopen - decompilation, dlsym עבור _ZNK3JSC13JSArrayBuffer8isSharedEv, והתאמת תבנית באמצעות memmem עם תבנית ה-gadget PACIA באורך 20 בתים, המאוחסנת ב-xmmword_100042ABC]

```
0000000100042AA0  45 6E 61 62 6C 65 72 39  73 65 74 75 70 48 6F 6F  Enabler9setupHoo
0000000100042AB0  6B 45 76 45 33 24 5F 31  00 00 00 00 30 02 C1 DA  kEvE3$_1....0...
0000000100042AC0  E8 03 90 9A 1F 01 00 F1  E0 07 9F 1A C0 03 5F D6  ..............._.
0000000100042AD0  4E 53 74 33 5F 5F 31 31  30 5F 5F 66 75 6E 63 74  NSt3__110__funct
```

[איור 7: תצוגת hex בכתובת 0x100042ABC, המציגה את תבנית ה-gadget בת 20 הבתים: PACIA X16, X17 (30 02 C1 DA) מודגשת באזור נתוני הקבועים של הבינארי]

ממצא 3: cache חתימות PAC בן 256 רשומות

כל קריאה ל-remotePACIA כוללת יצירת thread, הגדרת exception ports, שינוי מצב ה-thread בקרנל, הרצת ה-gadget וקבלת התוצאה. תהליך זה אורך מילישניות - הרבה יותר מדי זמן עבור callbacks של hooks שחייבים להסתיים בתוך מיקרו-שניות.

טבלת חתימות שחושבה מראש

Predator בונה בעת האתחול cache של 256 מצביעים חתומים מראש, המכסה כל ערך אפשרי של הבית העליון בכתובת. כאשר hook מופעל, ניתן לשלוף מיידית את המצביע החתום הנכון:

Parameter	Value	Notes
Cache size	4,096 bytes	256 entries × 16 bytes per entry
Entry format	[signed_PC, signed_LR]	8 bytes each
PC discriminator	`0x7481000000000000`	Program Counter signing
LR discriminator	`0x77D3000000000000`	Link Register signing
Index	cache[16 × (flags >> 24)]	Top byte of CPSR flags
Hash function	CityHash64 (Google)	For hashmap bucket lookup

שימוש ב-discriminators נפרדים עבור PC ועבור LR פירושו שכתובת חזרה חתומה אינה יכולה לשמש כתחליף ליעד קפיצה חתום - מנגנון defense-in-depth מצד Apple ש-Predator חייבת לקחת בחשבון. קריאות remotePACIA היקרות, 256 × 2 = 512 קריאות לכל cache, מתבצעות פעם אחת בלבד עבור כל יעד פונקציה ייחודי. כל קריאות ה-hook הבאות הן שליפות מיידיות מה-cache.

ממצא 4: callFunc - הרצת פונקציות מרוחקות

callFunc הוא המנגנון העיקרי של Predator להרצת פונקציות שרירותיות בתהליכים מרוחקים. הוא משלב את ה-PAC, Mach exceptions ומניפולציה של מצב ה-thread לכדי פרימיטיב אחד שניתן לקרוא לו, המקבל מצביע לפונקציה ועד 6 ארגומנטים, x0-x5.

המנגנון

"trojan thread" (מונח פנימי של Predator), יושב בתהליך המרוחק על breakpoint ומייצר Mach exceptions באופן רציף. callFunc מקבל את ה-exception, ממלא את רגיסטרי הארגומנטים, ושולף מ-cache ה-PAC את ערכי ה-PC וה-LR החתומים הנכונים. תשובת ה-exception מכילה את מצב ה-thread ששונה - וכאשר הקרנל מוסר את התשובה, ה-trojan thread ממשיך לרוץ בכתובת פונקציית היעד עם הארגומנטים שצוינו.

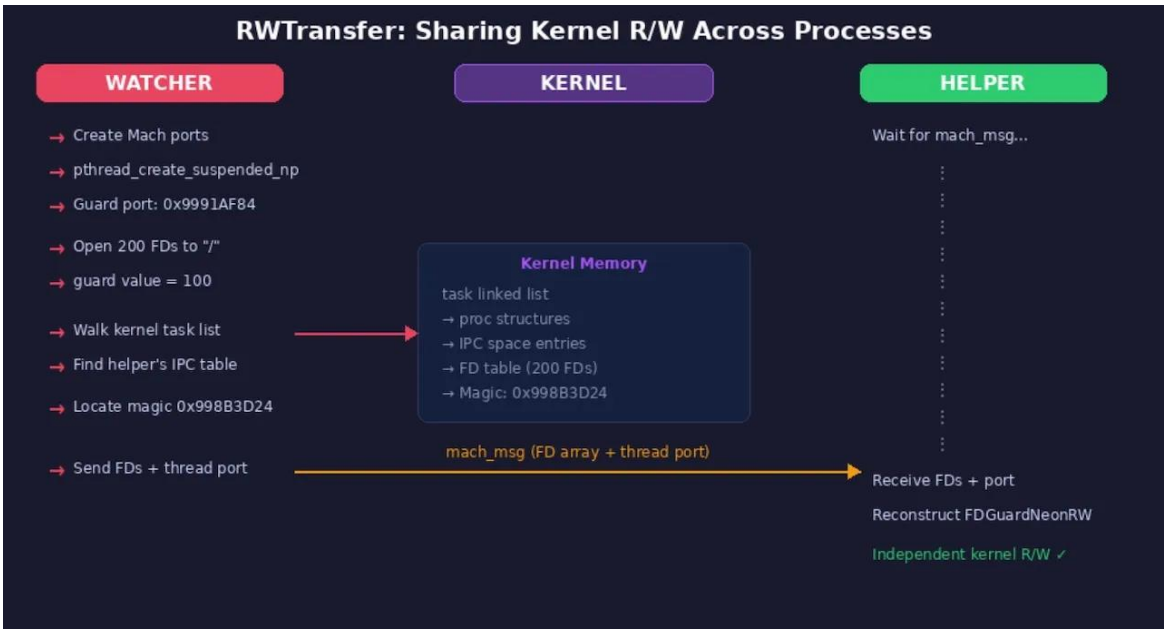
כתובת החזרה מסוג "poison LR" מצביעה בחזרה ל-breakpoint נוסף, כך שכאשר הפונקציה חוזרת, נוצר exception חדש ו-Predator מקבלת שוב שליטה. כך נוצר מנגנון reusable אינסופי לקריאה מרוחקת לפונקציות, כלומר מעין Remote Procedure Call. מקור התייחסות: הפונקציה בכתובת 0x10000B028 מתעדת בלוג את מקור הקובץ שלה כ-"TaskROPDevOff.h"- מה שמאשר שמדובר בטכניקה מבוססת ROP שתוכננה עבור מכשירים שבהם Developer Mode אינו מופעל:

```
LABEL_33:
    bzero(v64, 0x1000u);
    Log::LogManager::Init(v40);
    *(_QWORD *)&msg[0].msgh_bits = "callFunc: building PAC cache for ";
    *(_QWORD *)&msg[0].msgh_remote_port = 34;
    *(_QWORD *)v53 = "...";
    v54[0] = 4;
    sub_10000BA4C(0, "TaskROPDevOff.h", "", 451, msg, &v62, v53);
    v41 = (Log::LogManager *)NSTaskROP::WithoutDeveloperMode::TaskROP<FDGuardNeonRW>::signState(
                               (int)v16,
                               (int)&__dst,
                               v15,
                               1024,
                               0,
                               v64);
    if ( ((unsigned __int8)v41 & 1) == 0 )
    {
      Log::LogManager::Init(v41);
      *(_QWORD *)&msg[0].msgh_bits = "Statement signState(&data, key, sPoisonLRForTrojan, NULL, &cache) failed!";
      *(_QWORD *)&msg[0].msgh_remote_port = 74;
      v32 = msg;
      v33 = 452;
      goto LABEL_40;
    }
    *(_QWORD *)&msg[0].msgh_bits = v15;
    memcpy(&msg[0].msgh_remote_port, v64, 0x1000u);
    sub_10000AB70(&v16[75], msg, msg);
    goto LABEL_35;
  }
  if ( v30.u32[0] < 2uLL )
  {
    v37 = *(_QWORD *)&v29 - 1LL;
    while ( 1 )
    {
      v39 = v36[1];
      if ( v39 == v15 )
      {
        if ( v36[2] == v15 )
          goto LABEL_38;
      }
      else if ( (v39 & v37) != v31 )
      {
        goto LABEL_33;
      }
      v36 = (_QWORD *)*v36;
      if ( !v36 )
        goto LABEL_33;
    }
  }
  while ( 1 )
  {
    v38 = v36[1];
    if ( v38 == v15 )
      break;
    if ( v38 >= *(_QWORD *)&v29 )
      v38 %= *(_QWORD *)&v29;
    if ( v38 != v31 )
      goto LABEL_33;
  }
LABEL_23:
    v36 = (_QWORD *)*v36;
    if ( !v36 )
      goto LABEL_33;
  }
  if ( v36[2] != v15 )
    goto LABEL_23;
LABEL_38:
  memcpy(msg, v36 + 3, 0x1000u);
  Flags = Arm64State::getFlags((Arm64State *)&v50);
  v45 = (char *)msg + 16 * HIBYTE(Flags);
  v46 = *(void **)v45;
  v47 = (void *)*((_QWORD *)v45 + 1);
  Arm64State::setFlags((Arm64State *)&v50, Flags & 0xFFFFFFF1);
  Arm64State::setPC((Arm64State *)&v50, v46);
  Arm64State::setLR((Arm64State *)&v50, v47);
  v42 = (Log::LogManager *)ExceptionMessage<6,7166u,2147483650ull>::Send(&v61, &__dst);
  if ( ((unsigned __int8)v42 & 1) != 0 )
    return 1;
```

[איור 8: callFunc לאחר decompilation - במקרה של cache miss ב cache ה-PAC, הפונקציה מפעילה את signState כדי לבנות cache בן 256 רשומות (LABEL_33). במקרה של cache hit (LABEL_38), מתבצעת שליפה מיידית באמצעות 24 << getFlags, ולאחר מכן הגדרת PC ו-LR באמצעות setPC/setLR עם מצביעים חתומים, לפני הקריאה ל-ExceptionMessage::Send]

ממצא 5: RWTransfer - שיתוף פרימיטיבי קרנל בין תהליכים

הארכיטקטורה של Predator מפצלת יכולות בין מספר תהליכים: "watcher" מנהל את מחזור החיים, בעוד שתהליכי "helper" מבצעים את המעקב בפועל. ה-watcher משיג Kernel R/W באמצעות הניצול הראשוני, אך גם ה-helpers זקוקים לכך. RWTransfer פותר את הבעיה הזו:



[איור 9: פרוטוקול RWTransfer לשיתוף יכולות Kernel R/W בין תהליכים]

Component	Value	Purpose
Transfer magic	`0x998B3D24`	Identifies RW transfer structure in kernel memory
Port guard	`0x9991AF84`	Applied to Mach receive port
File descriptors	200 FDs to "/"	Created via `change_fdguard_np` (guard=100)
Thread creation	`pthread_create_suspended_np`	Dedicated thread for transfer
Delivery	`mach_msg`	FD array + thread port sent to helper

ה-watcher עובר על רשימות מקושרות בקרנל, עם offsets ייחודיים למחלקת המכשיר, כדי למצוא את מבנה ה-task של ה-helper, לאחר מכן את מרחב ה-IPC שלו, ולאחר מכן רשומות port בודדות. זהו אחד הרכיבים המורכבים ביותר ב-Predator - והוא כולל מניפולציה סימולטנית של Mach ports, file descriptors, רשימות מקושרות בקרנל ומצבי thread.

ממצא 6: פתרון מרוחק של מתודות Objective-C

כאשר Predator צריכה לבצע hook למתודת Objective-C בתהליך מרוחק, היא לא תמיד יכולה להסתמך על שאילתות runtime מקומיות. בעוד שמתודות שממומשות בתוך ה-dyld shared cache ממופות לאותן כתובות בכל התהליכים באותו boot, מתודות בבינאריים ספציפיים לאפליקציה כפופות ל-ASLR slides נפרדים לכל תהליך. לכן Predator מריצה את שרשרת פתרון ה-Objective-C runtime המלאה מרחוק באמצעות callFunc, וכך מבטיחה תוצאות נכונות ללא תלות במקום שבו מתודת היעד ממומשת:

Stage	Runtime function	Purpose
1	`sel_registerName(methodName)`	Convert method name to selector
2	`objc_getClass(className)`	Get class object
3	`class_getInstanceMethod(cls, sel)`	Get Method structure
4	`method_getImplementation(method)`	Get implementation pointer

כל קריאת callFunc עוברת דרך כל ה-pipeline: Mach exception ← cache PAC ← מניפולציה של מצב thread. ביצוע כל ארבעת השלבים מרחוק מבטיח פתרון נכון גם עבור מתודות שמחוץ ל-shared cache, שבהן הכתובות המקומיות והמרוחקות יהיו שונות עקב ASLR slides עצמאיים.

ממצא 7: מטריצת תמיכה במכשירים - 21 דגמים ב-5 מחלקות

פריסות מבני הקרנל משתנות בין דגמי iPhone עקב הבדלים בדור ה-SoC, במאפייני אבטחה ובו- iOS kernel builds. Predator מחזיקה קונפיגורציות לפי מחלקת מכשיר, עם offsets מדויקים בקרנל.

Class	SoC	iPhone models	Count
0	A12 Bionic	XS, XS Max, XS Max (China), XR	4
1	A13 Bionic	11, 11 Pro, 11 Pro Max, SE (2nd gen)	4
3	A14 Bionic	12 mini, 12, 12 Pro, 12 Pro Max	4
4	A15/A16	13 mini, 13, 13 Pro, 13 Pro Max, SE (3rd), 14, 14 Plus, 14 Pro, 14 Pro Max	9

הערה: מחלקת מכשיר 2 אינה נמצאת בשימוש בדגימה זו - ככל הנראה היא שמורה לגרסת חומרה מסוימת או אוחדה עם מחלקה אחרת. מכשירים שאינם נתמכים גורמים לפונקציה להחזיר 5, ובכך להפסיק את ההרצה במקום להסתכן ב-kernel panic. בסך הכול: 21 דגמי iPhone המשתרעים על חומרה מהשנים 2018-2022.

פענוח השוואות ה-XOR

הקוד שעבר decompilation, המוצג באיור 10, נראה כאילו הוא מכיל קבועים הקסדצימליים אטומים, אך למעשה מדובר פשוט במחרוזות ASCII המקודדות ב-XOR. הפונקציה קוראת ל-uname() כדי לקבל את מזהה המכונה של המכשיר, לדוגמה iPhone15,3, ולאחר מכן מבצעת השוואות XOR - אם התוצאה שווה לאפס, המכשיר תואם. הקידוד ישיר: 0x3531 הוא ASCII עבור "15" ו-0x332C הוא ASCII עבור ",3", שניהם בסדר בתים little-endian. כל תנאי בפלט ה-decompiled ממופה ישירות למזהה דגם iPhone ספציפי.

לדוגמה, התנאי הראשון בבלוק שמחזיר 4:

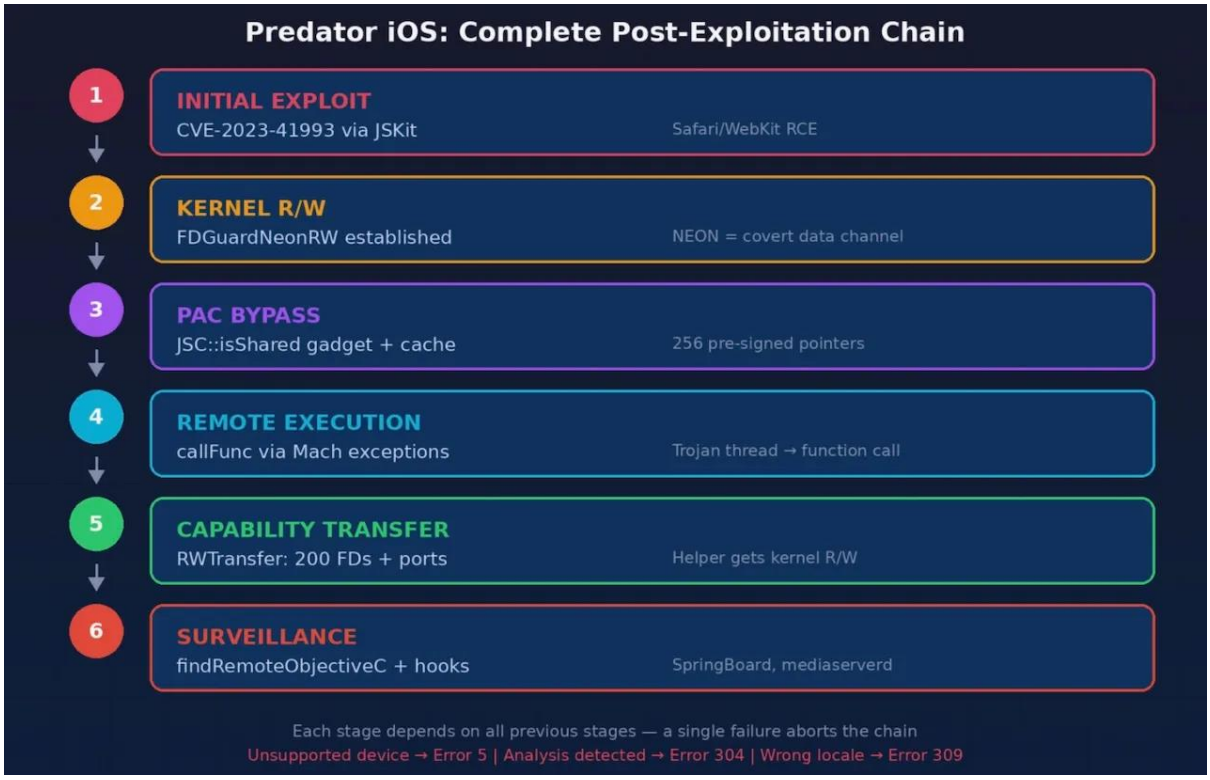
```
qword_1000491A0 ^ 0x3531656E6F685069 → "iPhoneXX"
(char *)&qword_1000491A0 + 3) ^ 0x332C3531656E6F → "oneXX,Y"
```

כאשר שתי תוצאות ה-XOR הן אפס, מחרוזת המכשיר תואמת ל-"iPhone15,3" - כלומר iPhone 14 Pro Max:

```
1  __int64 __fastcall VersionDispatcher::OffsetsVersionByDeviceInit(VersionDispatcher *this)
2  {
3    if ( !(_BYTE)qword_1000491A0 && uname((utsname *)&VersionDispatcher::m_uname) == -1 )
4    {
5      usleep(0);
6      usleep(0x32u);
7      usleep(0x32u);
8  LABEL_27:
9      usleep(0);
10     return 5;
11   }
12   if ( !(qword_1000491A0 ^ 0x3531656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x332C3531656E6FLL)
13     || !(qword_1000491A0 ^ 0x3531656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x322C3531656E6FLL)
14     || !(qword_1000491A0 ^ 0x3431656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x382C3431656E6FLL)
15     || !(qword_1000491A0 ^ 0x3431656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x372C3431656E6FLL)
16     || !(qword_1000491A0 ^ 0x3431656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x362C3431656E6FLL)
17     || !(qword_1000491A0 ^ 0x3431656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x352C3431656E6FLL)
18     || !(qword_1000491A0 ^ 0x3431656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x342C3431656E6FLL)
19     || !(qword_1000491A0 ^ 0x3431656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x332C3431656E6FLL)
20     || !(qword_1000491A0 ^ 0x3431656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x322C3431656E6FLL) )
21   {
22     return 4;
23   }
24   if ( !(qword_1000491A0 ^ 0x3331656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x342C3331656E6FLL)
25     || !(qword_1000491A0 ^ 0x3331656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x332C3331656E6FLL)
26     || !(qword_1000491A0 ^ 0x3331656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x322C3331656E6FLL)
27     || !(qword_1000491A0 ^ 0x3331656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x312C3331656E6FLL) )
28   {
29     return 3;
30   }
31   if ( qword_1000491A0 ^ 0x3231656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x382C3231656E6FLL
32     && qword_1000491A0 ^ 0x3231656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x352C3231656E6FLL
33     && qword_1000491A0 ^ 0x3231656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x332C3231656E6FLL
34     && qword_1000491A0 ^ 0x3231656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x312C3231656E6FLL )
35   {
36     if ( !(qword_1000491A0 ^ 0x3131656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x382C3131656E6FLL)
37       || !(qword_1000491A0 ^ 0x3131656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x362C3131656E6FLL)
38       || !(qword_1000491A0 ^ 0x3131656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x342C3131656E6FLL)
39       || !(qword_1000491A0 ^ 0x3131656E6F685069LL | *(__int64 *)((char *)&qword_1000491A0 + 3) ^ 0x322C3131656E6FLL) )
40     {
41       return 0;
42     }
43     goto LABEL_27;
44   }
45   return 1;
46 }
```

[איור 10: VersionDispatcher::OffsetsVersionByDeviceInit לאחר decompilation - זיהוי מכשיר באמצעות uname() עם השוואת מחרוזות מבוססת XOR, המחזירה מחלקות מכשיר 0-4, ומחזירה 5 עבור דגמים שאינם נתמכים כדי להפסיק את ההרצה באופן בטוח]

שרשרת התקיפה המלאה



[איור 11: שרשרת post-exploitation מלאה, מהניצול הראשוני ועד מעקב פעיל]

כל שלב תלוי בכל השלבים שקדמו לו - כשל יחיד בכל נקודה בשרשרת מבטל את השרשרת כולה. בשילוב עם מערכת קודי השגיאה שתועדה במאמר הראשון שלנו, הדבר מעניק למפעילי Predator מידע דיאגנוסטי מדויק לגבי איזה שלב נכשל ומדוע.

סיכום

FDGuardNeonRW מדגים כי רגיסטרים וקטוריים של ARM NEON, שנועדו לחישוב מקבילי, יכולים להיות מוסבים לשמש כערוץ סמוי לגישה לזיכרון הקרנל. פרימיטיבי הקריאה של 528 בתים והכתיבה המאומתת באמצעות polling אמינים מספיק כדי לתמוך בכל מסגרת ה-post-exploitation של Predator.

עקיפת ה-PAC ממחישה כי מאפייני אבטחה חומרתיים, אף שהם מעלים משמעותית את הרף, ניתנים לעקיפה כאשר לתוקף יש גישת קריאה/כתיבה לקרנל. באמצעות ציד gadgets במסגרות של Apple עצמה וחישוב מראש של cache חתימות, Predator מביסה את pointer authentication עם תקורה זניחה בזמן ריצה.

מנגנון העברת היכולות חושף את רמת התחכום ההנדסי של תוכנות ריגול מסחריות. העברת גישת Kernel R/W בין תהליכים - הכוללת מניפולציה סימולטנית של Mach ports, file descriptors, רשימות מקושרות בקרנל ומצבי thread - משקפת השקעה הנדסית מתמשכת ומקצועית.

ממצאים אלה מדגישים מציאות מפוכחת: ספקי תוכנות ריגול מסחריות משקיעים רבות בהנדסת post-exploitation, ולא רק בגילוי חולשות ראשוניות. הגנה מפני יכולות כאלה דורשת אמצעי אבטחה שפועלים מתחת לרמה שהכלים הללו מצליחים לפגוע בה - attestation שמושרש בחומרה, זיכרון קרנל חתום או אטום, וניטור out-of-band שאינו מסתמך על שלמות מחסנית התוכנה של המכשיר עצמו.

מקורות מידע

1. Jamf Threat Labs, "Predator's Kill Switch: Undocumented Anti-Analysis Techniques in iOS Spyware," January 2026

2. Jamf Threat Labs, "How Predator Spyware Defeats iOS Recording Indicators," February 2026

3. Google Threat Intelligence Group, "Sanctioned but Still Spying: Intellexa's Prolific Zero-Day Exploits Continue," December 2025

4. Apple, Apple Platform Security Guide

לקריאה נוספת

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

על המחבר

ניר אברהם, VP Research, Jamf. לינקדאין:

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

תורגם ונערך על ידי: IL4N10US

מספרי צ׳רץ׳

מאת סופי ציאדה

הקדמה

בואו נדבר על מספרים. אבל לא על המספרים ה״רגילים״ שכולנו מכירים, אלא על יצוג אחר של מספרים, מספרים בקידוד צ׳רץ׳. קידוד צ׳רץ׳ היא דרך להציג את המספרים הטבעיים בתור פונקציות מסדר גבוה.

פונקציה מסדר גבוה היא פונקציה שמקבלת פונקציה כלשהי כקלט, או מחזירה פונקציה כלשהי כפלט. או במילים פשוטות יותר - אלה פונקציות שפועלות על פונקציות.

הקידוד מתאים למספר הטבעי n את הפונקציה $C_n(f,x)$ שמוגדרת כך:

C_n היא פונקציה *שמקבלת* שני פרמטרים:

- הראשון, f, הוא פונקציה שמסומנת ב-f וממפה איבר כלשהו מסוג גנרי T, אל איבר כלשהו מאותו הסוג, T.
- הפרמטר השני, x, הוא איבר כלשהו, מאותו הסוג - T.

הפונקציה *מחזירה* את התוצאה של הפעלת f על x, n פעמים בשרשרת - בפעם ראשונה נפעיל את f על x, בפעם השנייה נפעיל את f על התוצאה של השלב הקודם (כלומר $f(x)$), וכן האלה n פעמים. אם נרצה לכתוב את זה באופן פורמלי נכתוב את זה כך:

$$\begin{aligned} C_n(f,x) &= f(\dots f(x)\dots) \\ &= f \circ f \circ \dots \circ f(x) \\ &= f^n(x) \end{aligned}$$

הופכים את זה לממשי

נסתכל על דוגמה אחת כדי להפוך את זה לפחות מופשט.

נגדיר את f שלנו בתור $f(x) = 2 * x$. מה שאומר שבדוגמאות הבאות, הקבוצה, או הסוג הגנרי, שהזכרנו למעלה, T, יהיה המספרים הממשיים ($\mathbb{R}$).

- הפונקציה C_0 היא הפונקציה שקידוד צ׳רץ׳ מתאים למספר 0. התוצאה של $C_0(f,3)$ תהיה 3. למה? כי הפעלנו את הפונקציה f על 3 אפס פעמים, כלומר, לא הפעלנו, ולכן נשארו עם האיבר המקורי. למעשה, באופן כללי, הפונקציה C_0 תמיד מחזירה את הפרמטר השני שלה כמו שהוא.
- התוצאה של $C_1(f,3)$ היא $f(3) = 2 * 3 = 6$
- התוצאה של $C_2(f,3)$ היא $f(f(3)) = 2 * (2 * 3) = 12$

אז הבנו איך הקידוד ממפה מספר טבעי n לפונקציה C_n. אבל זה רק כיוון אחד.

לשמחתנו, המיפוי ההפוך אפילו יותר פשוט. אם יש לנו פונקציה C_h, שידוע לנו שהיא מקודדת מספר טבעי בקידוד צ'רץ', ואנחנו רוצים לגלות מהו אותו ה-h, נחשב את הפונקציה עם $f(x) = x + 1$ בתור פרמטר ראשון ו-0 בתור פרמטר שני.
למה זה עובד? כי C_h תפעיל את f על 0 בסך-הכל h פעמים, כלומר, תוסיף ל-0 את המספר 1 h פעמים, ולכן התוצאה תהיה h.

אז המיפוי עובד לשני הכיוונים ☺

מטא-מטא פונקציות

בואו נסמן את הפונקציה שמתאימה למספר n את הפונקציה C_n בתור C_{enc} (encode, לא encrypt ☺). את הפונקציה שעושה את המיפוי ההפוך, או הפענוח, נסמן ב-C_{dec}.

אמרנו שהפונקציות C_n, או במילים אחרות מספרי צ'רץ', מייצגות מספרים, אז בואו נגדיר פעולות חיבור וכפל על המספרים האלה. המטרה היא להגדיר פעולות בין המספרים האלה שישמרו על המיפוי נכון. כלומר, נרצה שיתקיים:

$$C_{enc}(n+m) = C_{enc}(n) + C_{enc}(m)$$

הסימון פה טיפה טריקי. מה שקורה מצד שמאל זה חיבור רגיל בין שני טבעיים n, ו-m. על תוצאת החיבור מפעילים את C_{enc} ומקבלים את הפונקציה C_{n+m}.
מה שקורה מצד ימין זה שמפעילים את C_{enc} על כל מספר טבעי בנפרד, מקבלים שתי פונקציות C_n, ו-C_m ומבצעים בינהן חיבור. זה לא חיבור בין מספרים, אלא בין פונקציות. ואת אופן החיבור הזה, אנחנו צריכים להגדיר.

למעשה, התוצאה של החיבור $C_n + C_m$ צריכה להיות הפונקציה C_{n+m}. כלומר, היא מקבלת כפרמטרים פונקציה f וערך התחלתי x ומפעילה את f על x סה"כ $n + m$ פעמים. באותה הצורה נגדיר כפל. התוצאה של $C_n * C_m$ היא הפונקציה C_{n*m}.

מה ש[אלונזו צ'רץ'](#) רצה להראות בקידוד הזה, זה שאפשר לפתור כל בעיה חישובית דרך שימוש בפונקציות בתור טיפוס נתונים בסיסי, בלי לפנות בכלל למספרים. כמובן שלא משתמשים באופן היצוג הזה בפרקטיקה. הרבה יותר פשוט וזול לייצג בזכרון מספר ביצוג בינארי מאשר בתור פונקציה.

אבללל, זה לא ימנע מאיתנו לעשות את זה בכל זאת, כי זה יעזור לנו להבין איך בפועל עובדים חיבור וכפל בקידוד הזה!

קצת הסקלית לנשמה

אנחנו הולכים להגדיר את הקידוד שלנו ב-Haskell, שזו שפה נהדרת למטרה הזאת, כי פונקציות הן first-class citizens אצלה, וקל מאוד לכתוב פונקציות שמטפלות בפונקציות. אני יודעת שהיא בקושי בשימוש, אז אני מבטיחה להסביר כל מילה בקוד.

נגדיר טיפוס נתונים חדש בשם CNumber.

למעשה, CNumber הוא לא טיפוס חדש, אלא מעטפת לז'אנר של טיפוסים:

הוא מתאים לכל פונקציה ש:

- מקבלת כפרמטרים:
  - פונקציה שממפה איבר מ-T לאיבר ב-T , ה-f שאנחנו מכירים ממקודם
  - איבר ב-T , או x
- ומחזירה איבר ב-T.

ה-T הזה יכול להיות כל טיפוס נתונים, string, float32 וכו'.

כלומר T , הוא פרמטר גנרי בפונקציה ש-CNumber עוטף.

נכתוב את זה בהסקלית:

```
newtype CNumber = Nr (forall t. (t -> t) -> t -> t)
```

אנחנו מגדירים פה את CNumber ואומרים שהבנאי שלו, שנקרא Nr, מקבל פונקציה גנרית (שהמשתנה הגנרי שלה הוא t) מהצורה שהגדרנו מקודם.

בהסקל, הטיפוס a -> b -> c מתאר פונקציה שמקבלת כפרמטר ראשון איבר מסוג a וכפרמטר שני איבר מסוג b ומחזירה איבר מסוג c. אז מה שהגדרנו פה היא פונקציה ש:

- הפרמטר הראשון שלה הוא (t -> t), כלומר פונקציה שלוקחת ומחזירה איבר מסוג t.
- הפרמטר השני שלה הוא ערך מהסוג t.
- והיא מחזירה ערך מהסוג t.

אז מה שהגדרנו כאן זה תבנית למספרי צ'רץ' - C_n. זה לא אומר שכל הפונקציות שמתאימות לתבנית *בהכרח* מתארות מספרים בקידוד צ'רץ' - התבנית תתאים בבאופן כללי לפונקציות שמקבלות פונקציות ואיבר, ומחזירות איבר.

בואו נשתמש בהגדרה בשביל להגדיר מספרים:

```
zero = Nr (\ f x -> x )
```

כמו שאמרנו מקודם C_0 , לא מפעילה את f בכלל ומחזירה את x כמו שהוא.

שזה בדיוק מה שכתוב פה - אנחנו מעבירים ל-Nr פונקציית למדא, שמחזירה את הערך שהיא מקבלת.

```
one = Nr (\ f x -> f x )
```

C_1 מפעילה את f על x פעם אחת.

```
two = Nr (\ f x -> f (f x) )
```

באותו האופן C_2 , מפעילה פעמיים.

אז הצלחנו לייצג את הקידוד בכיוון הראשון, נעבור לכיוון השני - מפונקציה למספר.
נניח שיש לנו עצם מסוג CNumber, וידוע שהוא קידוד של מספר טבעי כלשהו. איך נוכל לשחזר את המספר שהוא מייצג? כבר פתרנו את הבעיה הזאת מקודם, ובהסקל נכתוב את זה ככה:

```
eval :: CNumber -> Int
eval (Nr Cn) = Cn (+1) 0
```

הפונקציה eval, מקבלת פונקציית צ'רץ', מסוג Nr, וקוראת לה עם הפרמטרים (+1), שמקבילה ל-$f(x) = x + 1$ בהסקל, והמספר 0. התוצאה היא המספר הטבעי שהפונקציה מייצגת.

לפני שנממש את הפונקציה שמחברת שני מספרי צ'רץ', נממש אחת פשוטה יותר בשם succ. מה שהיא עושה זה לקבל עצם מסוג CNumber, ולהחזיר את העצם הבא אחריו ביצוג. כלומר, אם היא קיבלה את C_1one/ היא תחזיר את C_2two/. הפונקציה הזו מקבילה לפונקציית העוקב של המספרים הטבעיים.

```
succ :: CNumber -> CNumber
succ (Nr Cn) = Nr (\ f x -> f (Cn f x) )
```

נניח ש-succ מקבלת CNumber שמתאר את הטבעי n.
היא מחזירה כתוצאה פונקצייה חדשה, שהיא גם מספר צ'רץ' – כי היא מקבלת פונקציה f ופרמטר x. הפונקציה החדשה תשתמש ב-Cn, שהיא פונקציית צ'רץ', כדי להפעיל את f על x n פעמים (כי זה בהגדרה מה ש-Cn, או C_n, עושה), ואז תפעיל עוד פעם אחת את f על התוצאה הסופית. וכך בעצם קיבלנו את העצם מסוג CNumber שמתאר את $n + 1$, או C_{n+1}.

חיבור, חיסור וכפל בצ'רצ'ית

כעת, יהיה לנו הרבה יותר קל להגדיר את פונקציית החיבור. אנחנו רוצים פונקציה בשם add שתקבל שני עצמים מסוג CNumber ותחזיר את ה-CNumber שמתאר את החיבור של העצמים, לפי חוקי החיבור שהגדרנו למעלה:

```
add :: CNumber -> CNumber -> CNumber
add (Nr Ca) (Nr Cb) = Nr (\ f x -> Ca f (Cb f x))
```

נניח ש-Ca ו-Cb הם מספרי צ'רץ' שמייצגים את הטבעיים a ו-b בהתאמה. הפונקציה שבסוגריים הכי פנימיים משתמשת קודם ב-Cb בשביל להפעיל את f על x , b פעמים. אחר-כך אנחנו משתמשים ב-Ca בשביל להפעיל על התוצאה של הסוגריים הפנימיים את f, עוד a פעמים. בסה"כ אנחנו מפעילים את f על x, $a+b$ פעמים, שזה בדיוק מה שרצינו.

אבל אנחנו מסוגלים לכתוב את זה יותר יפה. יש לנו כפרמטר את הפונקציה Ca שיודעת להפעיל פונקציות על איבר כלשהו a פעמים. למה לא שנשתמש בה בשביל להפעיל את succ על Cb Nr וככה למצוא את ה-CNumber שמתאר את $a+b$?

אם מפעילים את succ על Cb פעם אחת מקבלים את ה-CNumber שמייצג את $b+1$.

ולכן אם נפעיל את succ על Cb a פעמים באמצעות Ca נקבל את ה-CNumber שמייצג את $a+b$:

```
add :: CNumber -> CNumber -> CNumber
add (Nr Ca) (Nr Cb) = Ca succ (Nr Cb)
```

יש דרך אפילו יותר קצרה לכתוב את זה, והיא:

```
add :: CNumber -> CNumber -> CNumber
add (Nr Ca) = Ca succ
```

אבל אנחנו לא מנסים להגיע לקצרנות, אלא לקריאות ☺

נשאר לנו לממש רק פונקציית הכפל! אציין שזה תרגיל נהדר בעיניי וכדאי לכם לנסות לפתור אותו בעצמכם. כמובן שאתם יכולים פשוט להסתכל על התשובה בהמשך, אבל אז יהרס כל הכיף ☺

אנחנו רוצים להשתמש עוד פעם ב-Ca בשביל לחשב את ה-CNumber החדש. נעשה את זה ככה:

```
mult :: CNumber -> CNumber -> CNumber
mult (Nr Ca) (Nr Cb) = Ca (add (Nr Cb) zero
```

נעבור שלב שלב.

- התוצאה של add (Nr Cb) היא פונקציה שמקבלת CNumber כלשהו ומחברת אליו את Nr Cb (דרך הפעלה של succ b פעמים).
- אנחנו משתמשים ב-Ca בשביל להפעיל על zero, ה-CNumber שמייצג את 0, את התוצאה של השלב הקודם a פעמים.
- בכל הפעלה של add (Nr Cb) אנחנו מפעילים את succ b פעמים. אנחנו עושים את זה a פעמים, ולכן הפעלנו את succ בסה"כ $a * b$ פעמים. בעצם, קיבלנו את ה-CNumber שמייצג את $a * b$.

באותה צורה אפשר להגדיר חזקה, וטטרציה.

אני פשוט אראה את הקוד, ואשאיר את ההבנה כתרגיל לקורא ☺

```
mult :: CNumber -> CNumber -> CNumber
mult (Nr Ca) (Nr Cb) = Ca (add (Nr Cb)) zero
```

סיכום

קידוד צ'רץ' מציע דרך לייצוג מספרים באמצעות פונקציות. הרעיון הזה ניתן להרחבה גם לטיפוסים אחרים, כמו בולאנים, מספרים שליליים ואפילו מספרים ממשיים. בעצם, פונקציות הן אבני בנייה ורסטיליות מספיק בשביל להחליף לגמרי נתונים פרימיטיביים. התרגיל המחשבתי הזה עוזר לנו לחשוב מחדש על מה הם מספרים, במהות שלהם, ועל דרכים אלטרנטיביות לייצג אותם.

על המחברת

סופי ציאדה - Senior Full Stack Developer

המאמר נכתב במקור ב-2020 ו[פורסם](#) בבלוג שלי (גם ב[אנגלית](#)), בהשראת [אתגר קוד](#) באתר CodeWars.

[Sophie Saiada](#) · [GitHub](#) · [LinkedIn](#)

פירוק Anti-Cheat לגורמים

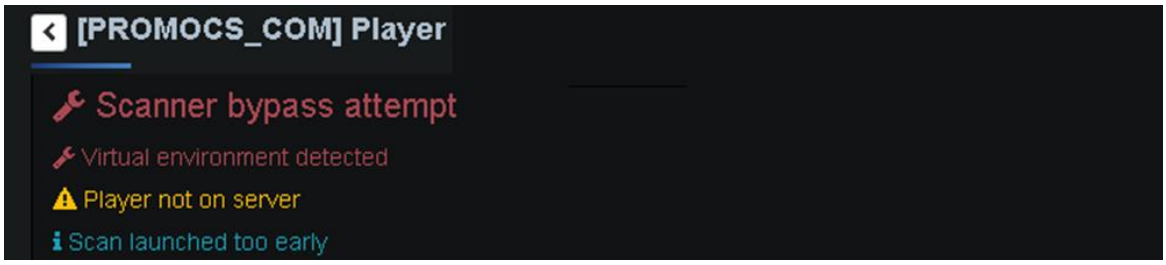
מאת דור נאמני (Dor00tkit)

תקציר

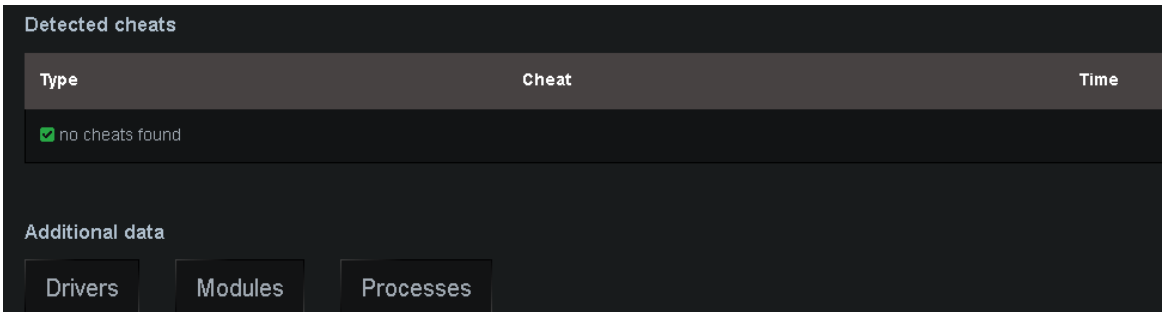
אז לא מזמן הייתה לי דודא לשחק Counter-Strike 1.6. וביום בהיר אחד אחרי כמה משחקים טובים האדמין של אותו סרבר חשב שאני משתמש בצ'יטים (דווקא אני שחקן לא רע, כנראה נפלתי על סרבר של נובים) - והוא ביקש ממני להיכנס לאתר fungun.top, להוריד משם תוכנה בשם Easy Cheat Detector, לבצע סריקה ולשלוח לו את התוצאה.

הערה: המאמר מתבסס על ההנחה המוקדמת שיש לקורא היכרות עם הנדסה לאחור, Windows API, שיטות Anti-Debug, שפת C ו-Python.

כמובן שאני לא אוריד תוכנה מפוקפקת למחשב האישי שלי אז הורדתי את זה ל-VM והרצתי. בסיום הסריקה שמתי לב שהתוכנה זיהתה שאני ב-VM:



בנוסף, שמתי לב שבדו"ח יש מידע על התהליכים, ה-DLLs של המשחק, והדרייברים במערכת:



סיקרן אותי להבין איך התוכנה הזאת עובדת מאחורי הקלעים והאם אפשר לערבב אותה. יאללה, בואו נתחיל!

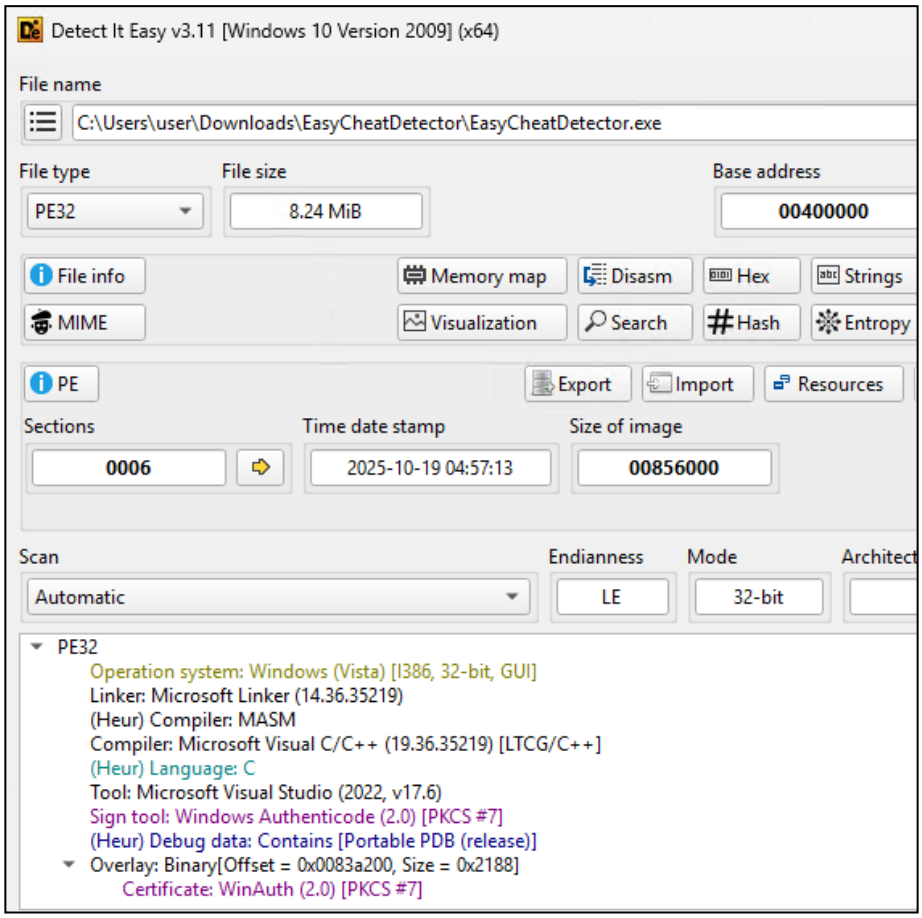
אז עוד לפני שנלכלך את הידיים, שווה להעיף מבט באתר הרשמי (במידה ויש) או בכל מידע אחר אודות אותה תוכנה הזמין באינטרנט. הרבה פעמים זה ייתן לנו לא מעט מידע מקדים. בהסתכלות בעמוד GitHub של התוכנה יש את המידע היבש שציינתי למעלה.

ניתוח מקדים

הערה: המחקר בוצע על גרסאות 2.74 ו-2.82 של Easy Cheat Detector.

הדבר הראשון שעשיתי זה להסתכל על הבינארי באופן באופן יבש ולהבין באיזה שפה הוא נכתב, האם הוא packed, איזה APIs הוא מייבא, האם יש לו עוד Resources וכמובן מחרוזות מעניינות. נראה שהתוכנה נכתבה בשפת C/CPP, קומפלה כ-32 ביט, ויש לה גם חתימה דיגטלית (בתכל'ס מדובר בתעודה חתומה עצמית ככה שזה לא מעיד על האמינות של התוכנה).

לשם כך השתמשתי בכלי Detect It Easy. Detect It Easy (DiE) הוא כלי מתקדם לזיהוי וניתוח סוגי קבצים, הנמצא בשימוש נרחב בקרב חוקרי נוזקות, אנשי אבטחת מידע ומהנדסים לאחור. הכלי משלב מנגנוני זיהוי מבוססי חתימות יחד עם ניתוח היוריסטי, ומאפשר ביצוע בדיקות קבצים מדויקות ויעילות במגוון פלטפורמות, כולל Windows, Linux ו-macOS. כך הוא נראה:



הארכיטקטורה הגמישה שלו, המבוססת על סקריפטים, מספקת יכולת הרחבה והתאמה גבוהה, והופכת אותו לאחד הכלים הוורסטיליים ביותר בתחום. DiE תומך במגוון רחב של פורמטי קבצים, קבצי הרצה, תמונות מערכת הפעלה, אורזים (Packers), מהדרים (Compilers) וסוגי קבצים נוספים, ובכך מאפשר סיווג וניתוח מקיפים של קבצים.

Imports

נעבור לפונקציות המעניינות שהתוכנה מייבאה, יש המון, אחלק את זה לקטגוריות:

- התעסקות עם ה-**Resource**, הפונקציות: FindResource, LoadResource ועוד. בדיקה של המידע הנמצא ב-Resource מראה שיש שם Certificate (הזיהוי לפי ה-Magic number, (הערכים:?? ?? 30 82)). ה-Certificate מוגן בסיסמה. בנוסף יש גם את הפונקציה PFXImportCertStore שיודעת לייבא Certificate.

  התעסקות עם **קבצים**, הפונקציות: CreateFile, ReadFile, SetFilePointer, FindFirstFileEx, FindNextFile - יכולים לשמש לגישה לקבצים של המשחק, למודולים ולדרייברים.

- התעסקות עם **תהליכים**, הפונקציות: First32Process, Next32Process, OpenProcess, VirtualQueryEx, VirtualAllocEx, WriteProcessMemory, ReadProcessMemory - יכולים לשמש לאיסוף מידע על התהליכים הרצים, איסוף מידע על התהליך של המשחק.
- התעסקות עם **Threads**, הפונקציות: Thread32First, Thread32Next, OpenThread, GetThreadContext, SetThreadContext, ResumeThread, SwitchToThread.
- התעסקות עם **Registry**, הפונקציות: RegOpenKeyEx, RegEnumKeyEx, RegCreateKeyEx, RegQueryInfoKey, RegQueryValueEx, RegEnumValue, RegGetValue, RegSetValueEx - יכולים לשמש לאיסוף ואחסון מידע, למשל איסוף מידע על התוכנות המותקנות ודרייברים, שמירת מידע של התוכנה עצמה.
- התעסקות עם Services, הפונקציות: OpenSCManager, OpenService, QueryServiceStatusEx, QueryServiceConfig, EnumDependentServices - יכולים לשמש לאיסוף מידע על הדרייברים וה-Services הקיימים במערכת.
- התעסקות עם **Debugging**, הפונקציות: DebugActiveProcess, WaitForDebugEvent, ContinueDebugEvent, CheckRemoteDebuggerPresent, IsDebuggerPresent - יכולים לשמש לביצוע Debugging על תהליך מסוים, זיהוי של Debugging.

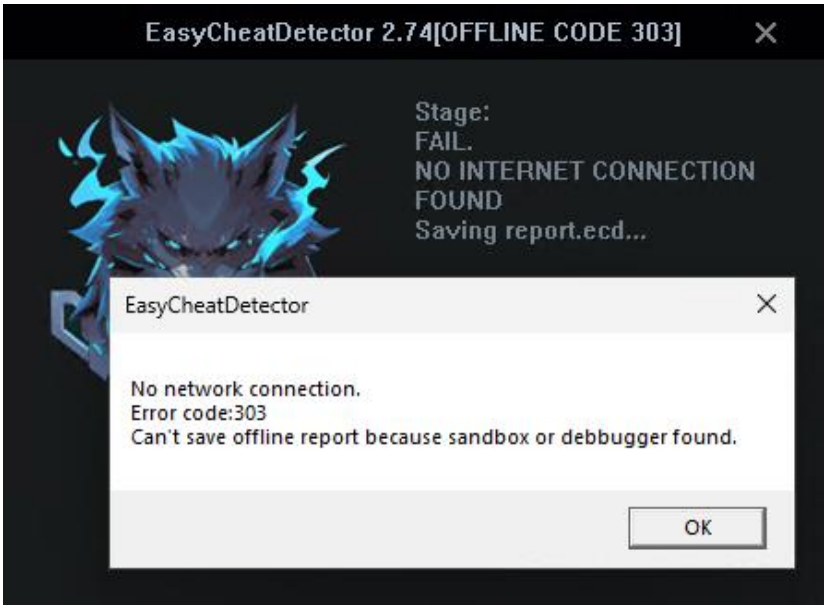
מחרוזות מעניינות

(hacked client?) Antihack executable is corrupted! ECD ANTIHACK RULE ECD Antihack binary is corrupted. FUCKINGHACKERS THIS CUTE BEAST CAN PROTECT ANTIHACK EXECUTABLE FROM BAD HACKERS This antihack tool now support next games: Minecraft (not all clients!), RUST (withot EAC), CS 1.6, CS SOURCE, CS GO, CS 2 **[SELF PROTECT] Found critical error.** blackbone tao::json

ניתן לשים לב שאחת המחרוזות מרמזת שיש לתוכנה מנגנון **Self-Protect** שאמור להגן על עצמה מפני חוקרים כמונו. בהמשך נראה איך ניתן לעקוף את אותו המנגנון. בנוסף ניתן לראות שהמחרוזת **blackbone** מופיעה כמה פעמים, חיפוש בגוגל מגלה שזו למעשה ספרייה נוחה המאפשרת לגשת לזיכרון של תהליכים. ולסיום, המחרוזת האחרונה ברשימה מגלה שנעשה שימוש בספרייה נוספת בשם taoJSON.

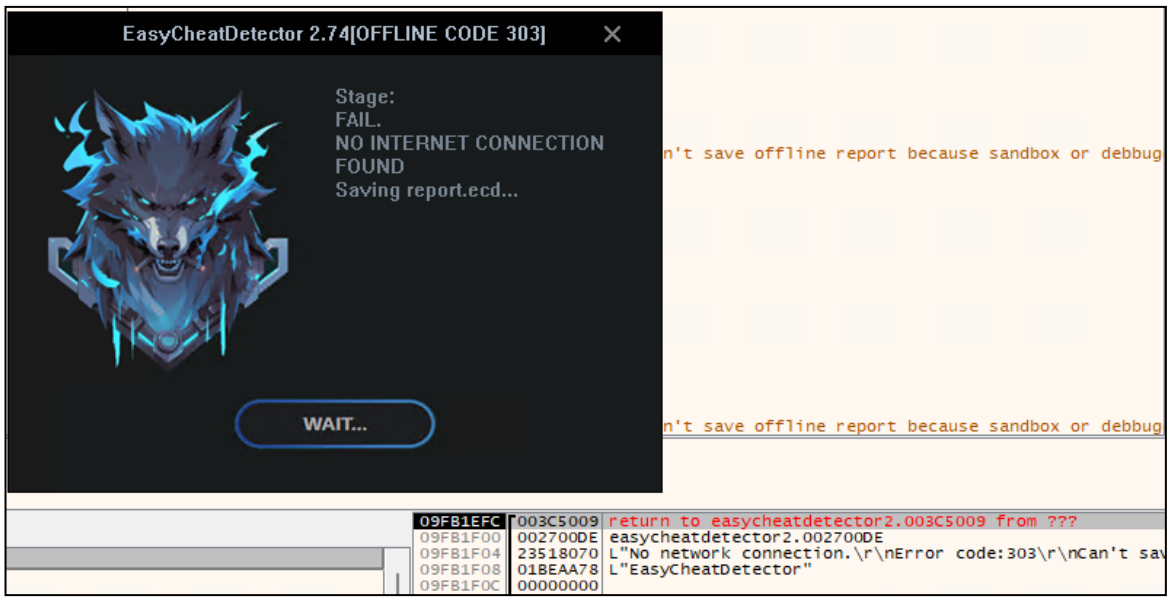
ניתוח משולב - סטטי ודינמי

את המחקר הסטטי נבצע באמצעות IDA Pro, ואת המחקר הדינמי בעזרת x32dbg. הדבר השני שניסיתי זה לראות איך התוכנה מתנהגת שאין אינטרנט. ככה לרוב אני מעדיף להתחיל מחקר בסביבה מבודדת, שלא תדליף מידע על הסביבה שאני עובד בה. נריץ את התוכנה, נלחץ Scan, התוכנה תרוץ ותסרוק ו..



נראה שהתוכנה מזהה שאנחנו רצים בתוך VM ולכן מסרבת לשמור את הדו"ח מקומית.

חיפוש אחרי המחרוזת הזאת באופן סטטי לא מביאה תוצאות. כנראה היא מפוענחת רק בזמן ריצה. לפי החלון שקפץ נראה שנעשה שימוש בפונקציה MessageBox על מנת להציג את השגיאה הזאת. נפתח Debugger, נשים Breakpoint על MessageBox ונריץ...



כעת אפשר להתחיל לעקוב וללכת קצת אחורה.

כשניסיתי לעקוב אחרי הקטע קוד ב-IDA Pro אז קיבלתי את השגיאה הידועה לשמצה " decompilation failure too big function". נסדר את זה, נחכה כמה דקות טובות עד ש IDA תסיים לדקמפל את הפונקציה הענקית. ונוכל להמשיך במחקר. נראה שקצת לפני שמגיעים לקוד שקורא לפונקציה MessageBox נבדקים כמה משתנים גלובלים:

```
else if ( byte_6E97AD || byte_6D292A || byte_6E97AF || v7129 )
{
  v2935 = sub_246430(dword_6E9854);
  v2949 = sub_2219C0(v8498);
  v2948 = v2949;
  v8896 = 0xC30;
  v2939 = v2949;
  v2947 = sub_1253F0(v8497, dword_6E97B0);
  v2946 = v2947;
  LOBYTE(v8896) = 0x31;
  v2943 = v2947;
  v2945 = sub_221800(v8496);
  v2944 = v2945;
  LOBYTE(v8896) = 0x32;
  v2942 = v2945;
  v2941 = sub_235ED0(v8495, v2945, v2943);
  v2940 = v2941;
  LOBYTE(v8896) = 0x33;
  v2938 = v2941;
  v2937 = sub_235ED0(v8494, v2941, v2939);
  v2936 = v2937;
  LOBYTE(v8896) = 0x34;
  v2934 = sub_246430(v2937);
  v2933 = hWndParent;
  MessageBoxW(hWndParent, v2934, v2935, 0);// Can't save offiline report because sandbox or debugger found.
  LOBYTE(v8896) = 0x33;
  sub_246A50(v8494);
  LOBYTE(v8896) = 0x32;
  sub_246A50(v8495);
  LOBYTE(v8896) = 0x31;
```

אם נעקוב אחרי הרפרנסים של כל אחד מהם, נראה שהם נבדקים המון פעמים. נתחיל לפרק אחד אחד.

זיהוי Debugger או VM על פי זמן

נעקוב אחרי המשתנה הגלובלי byte_6E97AD, נראה שהמון קטעי קוד המאתחלים אותו ל-**1** נראים כך:

```
v171 = GetTickCount() - TickCount;
v172 = 0;
if ( v171 > qword_6E97C0 )
  qword_6E97C0 = GetTickCount() - TickCount;
if ( GetTickCount() - TickCount > 0x7D0 && GetTickCount() - TickCount < 0x9C40 )
{
  byte_6E97AD = 1;
  LOBYTE(v274) = 3;
```

השיטה הזו היא שיטת זיהוי Debugger / VM דיי מוכרת. התוכנה קוראת לפונקציה GetTickCount בפעם הראשונה (שלב זה אינו מופיע בתמונה), מבצעת קטע קוד, ולאחר מכן קוראת שוב ל-GetTickCount. בהמשך מתבצעת השוואה בין התוצאות של הקריאות כדי למדוד כמה זמן לקח לקטע הקוד לרוץ.

כאשר מריצים את התוכנה תחת Debugger, ובמיוחד אם משתמשים ב-Breakpoints או ב-Single-step, הביצוע מואט באופן משמעותי. באופן דומה, גם ריצה בתוך VM עשויה להיות איטית יותר עקב Overhead של וירטואליזציה, שתלוי בקונפיגורציית ה-VM ובמערכת ה-Host. אם הזמן שנמדד חורג מהזמן הצפוי להרצת אותו קטע קוד, התוכנה יכולה להסיק שהיא רצה תחת Debugger או בתוך סביבה וירטואלית.

דרכים אפשריות לעקוף את הבדיקה:

1. פשוט, לא לשים Breakpoints ולא לבצע Single-step בטווח הקוד של הבדיקות.
2. לקנפג את ה-VM עם חומרה טובה. אצלי הבדיקה עוברת.
3. הוק לפונקציה GetTickCount שתחזיר ערכים מפוברקים.
4. פיצפוץ הקוד. נוכל לפצפץ את החלק שבודק, את החלק המאתחל את המשתנה הגלובלי ל-1 שיאתחל אותו ל-0, וכד'.

המשתנה הגלובלי השני הוא byte_6D292A, הוא גם כן נמצא באותם אזורים שבודקים את byte_6E97AD. בניגוד ל-byte_6E97AD שמאותחל ל-0, המשתנה הגלובלי byte_6D292A מאותחל ל-1, ורוב הכתיבות אליו מאתחלות אותו ל-0:

xrefs to byte_6D292A

Direct	Type	Address	Text
Up	r	sub_1702C0+27E	mov al, byte_6D292A
Up	w	sub_168B70+63	mov byte_6D292A, 0
Up	w	sub_1702C0:loc_170537	mov byte_6D292A, 0
...	w	sub_1B6AF0+63	mov byte_6D292A, 0
...	w	sub_1B9680+63	mov byte_6D292A, 0
...	w	sub_1D6340+63	mov byte_6D292A, 0
...	w	sub_1F5970+63	mov byte_6D292A, 0
...	w	sub_1F9280+63	mov byte_6D292A, 0
...	w	sub_20CC00+63	mov byte_6D292A, 0
...	w	sub_2106E0+63	mov byte_6D292A, 0
Up	r	sub_1702C0+44832	movzx eax, byte_6D292A
Up	r	sub_1702C0:loc_1ADAF1	movzx ecx, byte_6D292A
...	r	sub_1702C0+45482	movzx ecx, byte_6D292A
Up	r	sub_1702C0+3C0BC	movzx edx, byte_6D292A
Up	r	sub_1702C0+44153	movzx edx, byte_6D292A
Up	r	sub_1702C0+44BC6	movzx edx, byte_6D292A

רוב האתחולים של byte_6D292A נראים ככה:

```
v92 = this;
v273 = 0;
v272 = 0;
v170 = sub_55B799() % 0x32 + 0x19;
byte_6D292A = 0;
v1 = sub_55B799();
v3 = v1 % 0x32;
result = v1 / 0x32;
if ( v3 <= 0x14 )
{
  for ( i = 0; ; ++i )
  {
    result = i;
    if ( i >= v170 )
      break;
    if ( sub_55B799() % 0x32 <= 0xF )
    {
      TickCount = GetTickCount();
      v266 = sub_55B799();
      sub_54B7A0(v326, 0, 0x100);
      sub_54B7A0(v325, 0, 0x200);
```

אבל יש אחד שמשך את תשומת הלב שלי והוא נראה ככה:

```
sub_50360();
qword_6E7768 = sub_2E59C0(dword_6E77A0);
if ( sub_31E050() )
{
  byte_6D292A = 0;
  byte_6E97AD = 0;
}
```

למעשה ניתן לראות שגם byte_6E97AD מאותחל ל-0 במידה והפונקציה sub_31E050 מחזירה ערך שונה מ-0.

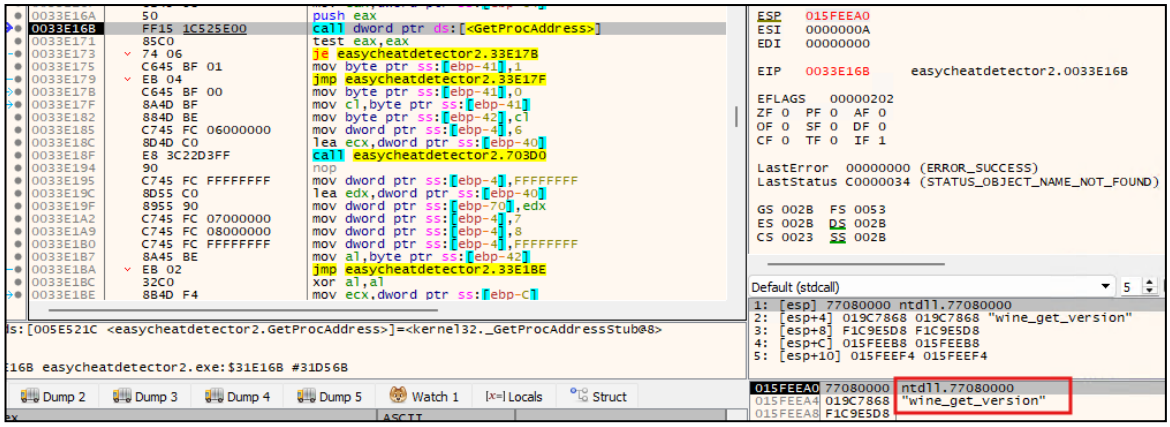
יין ישמח לבב אנוש

חיפוש רפרנסים לפונקציה sub_31E050 מראה שהיא נקראת לא מעט פעמים. ככה הפונקציה נראת:

```
bool sub_31E050()
{
  const CHAR *lpProcName; // [esp+14h] [ebp-60h]
  int v2; // [esp+1Ch] [ebp-58h]
  const WCHAR *lpModuleName; // [esp+24h] [ebp-50h]
  _DWORD *v4; // [esp+28h] [ebp-4Ch]
  bool v5; // [esp+32h] [ebp-42h]
  _BYTE v6[24]; // [esp+34h] [ebp-40h] BYREF
  _BYTE v7[24]; // [esp+4Ch] [ebp-28h] BYREF
  int v8; // [esp+70h] [ebp-4h]

  if ( dword_6FB7F0 > *(*NtCurrentTeb()->ThreadLocalStoragePointer + 4) )
  {
    sub_549732(&dword_6FB7F0);
    if ( dword_6FB7F0 == 0xFFFFFFFF )
    {
      v8 = 0;
      v4 = sub_31E1E0(v7);
      LOBYTE(v8) = 1;
      lpModuleName = sub_125A90(v4);
      dword_6FBA08 = GetModuleHandleW(lpModuleName);
      LOBYTE(v8) = 2;
      sub_2622E0(v7);
      v8 = 0xFFFFFFFF;
      sub_5496E1(&dword_6FB7F0);
    }
  }
  if ( !dword_6FBA08 )
    return 0;
  v2 = sub_31E2B0(v6);
  v8 = 5;
  lpProcName = sub_219FC0(v2);
  v5 = GetProcAddress(dword_6FBA08, lpProcName) != 0;
  v8 = 6;
  sub_503D0(v6);
  v8 = 0xFFFFFFFF;
  return v5;
}
```

נראה שהפונקציה מנסה למצוא פונקציה מסוימת ע"י שימוש בפונקציה GetProcAddress, מכיוון שהמחרוזות מפוענחות בזמן ריצה הדרך המהירה למצוא את השם של הפונקציה שאותה מחפשים היא בזמן ריצה, למשל באמצעות Debugger:



אז נראה שהתוכנית מחפשת אחרי הפונקציה wine_get_version בתוך Ntdll. לפי מה שזכור לי לא אמורה להיות פונקציה כזאת ב-Ntdll. האם Wine אומר לכם משהו? (גם אם לא, אז חיפוש קצר בגוגל מביא תוצאות טובות) לי כן. Wine הוא כלי שמאפשר להריץ תוכנות Windows במערכות מבוססות Linux ללא צורך ב-VM. האם יש מצב שכותב התוכנה משתמש במערכת מבוססת Linux בשביל לדבג את התוכנה בלי כל/חלק מההגנות, או על מנת לדלג על חלקים מסוימים? או שיש שחקנים שמריצים את המשחק על גבי Wine?

זיהוי Debugger באמצעות Debug Registers

נמשיך עם המשתנה הגלובלי השלישי, byte_6E97AF, חיפוש רפרנסים אליו מביא את הרפרנס המעניין הבא:

```
CurrentThreadId = GetCurrentThreadId();
v6797 = OpenThread(0x1FFFFFu, 0, CurrentThreadId);
if ( v6797 )
{
  sub_54B7A0(&v7164, 0, 0x2CC);
  v7164.ContextFlags = 0x10010;
  memset(&v7164.Dr0, 0, 0xC);
  v7164.Dr3 = &v7164;
  v7164.Dr7 &= 0xFFFFFFAA;
  SetThreadContext(v6797, &v7164);
  GetThreadContext(v6797, &v7164);
  if ( v7164.Dr0 || v7164.Dr1 || v7164.Dr2 || v7164.Dr3 != &v7164 )
    byte_6E97AF = 1;
  memset(&v7164.Dr0, 0, 0x10);
  v7164.Dr7 &= 0xFFFFFFAA;
  SetThreadContext(v6797, &v7164);
  CloseHandle(v6797);
}
```

גם זאת שיטה די מוכרת לזיהוי Debugger. בקצרה, שיטה זאת בודקת התערבות של Debugger ע"י כתיבה וקריאה חוזרת של Debug Registers ב-Thread הנוכחי, כאשר כל שינוי בלתי צפוי בערכים מצביע על שימוש ב-Hardware breakpoints המעיד על הרצת התוכנה תחת Debugger.

דרכים לעקוף את הבדיקה

1. פשוט, לא להשתמש ב-Hardware breakpoints רק עד לאחר סיום ריצת אותו קטע קוד של הבדיקה.
2. לבצע הוק לפונקציה GetThreadContext ולפברק את הערכים שהיא מחזירה. בנוסף נצטרך גם להגן על ה-Debug Registers מפני שינוי לא רצוי באמצעות הוק לפונקציה SetThreadContext.
3. פיצפוץ הקוד. נוכל לפצפץ את החלק שדורס, שבודק, ואת החלק המאתחל את המשתנה הגלובלי ל-**1** שיאתחל אותו ל-**0**, וכד'.

יש לא מעט רפרנסים שמאתחלים את המשתנה v7129. נתחיל עם הפונקציה sub_2F3030:

```
char sub_2F3030()
{
  HANDLE hObject; // [esp+0h] [ebp-4h]

  hObject = CreateFileW(L"\\\\?\\\\A3E64E55_fl", GENERIC_READ, 0, 0, OPEN_EXISTING, 0, 0);
  if ( hObject == 0xFFFFFFFF )
    return FALSE;
  CloseHandle(hObject);
  return TRUE;
}
```

מה זאת הגישה הזאת לנתיב המוזר הזה?

```
\\?\\A3E64E55_fl
```

חיפוש בגוגל מראה שזה שייך ל-ANY.RUN שהוא Sandbox מוכר מאוד.

```
int sub_2F3070()
{
  _DWORD v1[6]; // [esp-20h] [ebp-8Ch] BYREF
  int v2; // [esp-8h] [ebp-74h]
  int v3; // [esp-4h] [ebp-70h]
  _BYTE *v4; // [esp+8h] [ebp-64h]
  _DWORD *v5; // [esp+Ch] [ebp-60h]
  _DWORD *v6; // [esp+10h] [ebp-5Ch]
  int v7; // [esp+14h] [ebp-58h]
  _DWORD *v8; // [esp+18h] [ebp-54h]
  _DWORD *v9; // [esp+1Ch] [ebp-50h]
  int v10; // [esp+20h] [ebp-4Ch]
  int v11; // [esp+24h] [ebp-48h]
  char v12; // [esp+29h] [ebp-43h] BYREF
  unsigned __int8 v13; // [esp+2Ah] [ebp-42h]
  char v14; // [esp+2Bh] [ebp-41h]
  _BYTE v15[24]; // [esp+2Ch] [ebp-40h] BYREF
  _BYTE v16[24]; // [esp+44h] [ebp-28h] BYREF
  int v17; // [esp+68h] [ebp-4h]

  v7 = 0;
  v3 = 0;
  v2 = 0;
  v11 = sub_2F3190(&v12, v15);
  v10 = v11;
  v17 = 0;
  v6 = v1;
  v5 = sub_2D85D0(v1, v11, &unk_6E99A0);
  LOBYTE(v17) = 0;
  v9 = sub_2B9E30(v16, v1[0], v1[1], v1[2], v1[3], v1[4]);
  v8 = v9;
  LOBYTE(v17) = 2;
  v14 = sub_2C1C90(v9, v2, v3);
  v13 = v14;
  LOBYTE(v17) = 3;
  sub_2622E0(v16);
  v4 = v16;
  v17 = 6;
  sub_2622E0(v15);
  return v13;
}
```

הפונקציה מפענחת בזמן ריצה מחרוזת ומנסה לבדוק האם הנתיב הבא קיים:

```
C:\Program Files\KernelLogger
```

שגם הוא מסתבר שייך ל-ANY.RUN. הפונקציה sub_2F3400:

```
v23 = 0;
sub_145030(v29, L"srvpost.exe");
dwProcessId = sub_2D9B50(v29);
sub_2622E0(v29);
v24 = 0xFFFFFFFF;
if ( !dwProcessId )
  return 0;
sub_3696D0(v25);
v24 = 4;
if ( sub_369C80(dwProcessId, 0x438u) < 0 )
{
  v24 = 0xFFFFFFFF;
  sub_369B00(v25);
  return 0;
}
v14 = v26;
v13 = v26;
sub_145030(v27, L"winanr.dll");
LOBYTE(v24) = 5;
v23 |= 1u;
v12 = sub_36CCC0(v1, v27, 0, 2);
v11 = v12;
v24 = 6;
v23 |= 2u;
v10 = *v12;
v17 = v10 != 0;
v22 = v10 != 0;
if ( v10 )
  goto LABEL_6;
v9 = v26;
v8 = v26;
sub_145030(v28, L"winsanr.dll");
```

הפונקציה בודקת האם יש במערכת תהליך בשם srvpost.exe והאם המודולים winanr.dll ו-winanr.dll טעונים. שוב, חיפוש בגוגל מביא עוד פעם תוצאה שזה שייך ל-ANY.RUN.

בהמשך מגיעים לקטע קוד הבא:

```
v3549 = (void *)sub_20C050((int)v7868);
v3548 = v3549;
LOBYTE(v8466) = 0xD5;
v268 = (const WCHAR *)sub_246430(v3549);
ModuleHandleW = GetModuleHandleW(v268);       // cmdvrt32.dll
LOBYTE(v8466) = 0xC8;
sub_246A50(v7868);
if ( ModuleHandleW )
{
  v3546 = 7;
  v466[0xF4] = &v331;
  v466[0xF3] = sub_236F80(&v3546);
  LOBYTE(v8466) = 0xD6;
  v466[0xF2] = &v327;
  v466[0xF1] = sub_20C140(&v327);
  LOBYTE(v8466) = 0xD6;
  v3545[1] = sub_2425E0(&unk_6E99D8, v327, v328, v329, *(int *)v330, *(int *)&v330[4], *(int *)&v330[8]);
  LOBYTE(v8466) = 0xC8;
  sub_2429F0(v331, v332, v333, v334);
  sandbox_vm_checks = 1;
}
```

בזמן ריצה מפוענחת המחרוזת cmdvrt32.dll ומנסים לבדוק האם ה-DLL הזה טעון בתהליך הנוכחי. גם הפעם נעזרתי בגוגל כדי להבין לאיזה מוצר זה שייך והתוצאות הצביעו על Comodo Antivirus.

בהמשך יש את הקטע קוד הבא:

```
 }
 if ( (unsigned __int8)SetCalendarInfoA != 0xE9
   && (unsigned __int8)SetLocaleInfoW == 0xE9
   && (unsigned __int8)SetLocaleInfoA == 0xE9 )
 {
   v3545[0] = 1;
   v466[0xF0] = &v331;
   v466[0xEF] = sub_236F80(v3545);
   LOBYTE(v8466) = 0xD8;
   v466[0xEE] = &v327;
   v466[0xED] = sub_20C1E0(&v327);
   LOBYTE(v8466) = 0xD8;
   v3544[1] = sub_2425E0(&unk_6E99D8, v327, v328, v329, *(int *)v330, *(int *)&v330[4], *(int *)&v330[8]);
   LOBYTE(v8466) = 0xC8;
   sub_2429F0(v331, v332, v333, v334);
   sandbox_vm_checks = 1;
 }
 else if ( (unsigned __int8)SetCalendarInfoA != 0xE8
        && (unsigned __int8)SetLocaleInfoW == 0xE8
        && (unsigned __int8)SetLocaleInfoA == 0xE8 )
 {
   v3544[0] = 1;
   v466[0xEC] = &v331;
   v466[0xEB] = sub_236F80(v3544);
   LOBYTE(v8466) = 0xDA;
   v466[0xEA] = &v327;
   v466[0xE9] = sub_20C360(&v327);
   LOBYTE(v8466) = 0xDA;
   v3543[1] = sub_2425E0(&unk_6E99D8, v327, v328, v329, *(int *)v330, *(int *)&v330[4], *(int *)&v330[8]);
   LOBYTE(v8466) = 0xC8;
   sub_2429F0(v331, v332, v333, v334);
   sandbox_vm_checks = 1;
```

הקטע קוד הזה בודק האם יש הוקים לפונקציות הבאות:

SetCalendarInfoA, SetLocaleInfoW, SetLocaleInfoA

אולי זה גם רלוונטי ל-Comodo Antivirus או ל-Sandbox כלשהו שמבצע הוק לפונקציות האלה.

VMware

הפונקציה sub_2F4630:

```
v5 = 0;
v13 = sub_2F4810(v22);                          // "vmtoolsd.exe"
v12 = v13;
v3 = 0;
v11 = sub_2D9C10(v1, v13);
v10 = v11;
LOBYTE(v3) = 1;
v15[2] = v11;
v9 = (v11[1] - *v11) >> 2;
v8 = v9;
LOBYTE(v3) = 0;
sub_2597B0(v1);
v3 = 2;
sub_2622E0(v22);
v4 = v22;
v3 = 0xFFFFFFFF;
if ( v8 )
  return 1;
v16 = 0;
v3 = 5;
v7 = sub_2F4900(v21);                           // "SOFTWARE\\VMware, Inc.\\VMware Tools"
v6 = v7;
LOBYTE(v3) = 6;
sub_13DF20(&v16, v15, HKEY_LOCAL_MACHINE, v7, 0x20119u);
```

ניסיון לבדוק האם רצים מעל VMware באמצעות חיפוש אחרי תהליך vmtoolsd.exe. ובדיקה נוספת אחרי המפתח הבא ברגסטרי:

```
HKLM\SOFTWARE\VMware, Inc.\VMware Tools
```

VirtualBox

הפונקציה sub_2F4190:

```
char sub_2F4190()
{
  _BYTE v1[12]; // [esp+4h] [ebp-88h] BYREF
  _BYTE *v2; // [esp+14h] [ebp-78h]
  int v3; // [esp+18h] [ebp-74h]
  int v4; // [esp+1Ch] [ebp-70h]
  _DWORD *v5; // [esp+20h] [ebp-6Ch]
  _DWORD *v6; // [esp+24h] [ebp-68h]
  int v7; // [esp+28h] [ebp-64h]
  int v8; // [esp+2Ch] [ebp-60h]
  LPCWSTR lpFileName; // [esp+30h] [ebp-5Ch]
  _DWORD *v10; // [esp+34h] [ebp-58h]
  _DWORD *v11; // [esp+38h] [ebp-54h]
  BOOL v12; // [esp+3Ch] [ebp-50h]
  _DWORD *v13; // [esp+40h] [ebp-4Ch]
  HANDLE hObject; // [esp+44h] [ebp-48h]
  bool v15; // [esp+48h] [ebp-41h]
  _BYTE v16[24]; // [esp+4Ch] [ebp-40h] BYREF
  _BYTE v17[24]; // [esp+64h] [ebp-28h] BYREF
  int v18; // [esp+88h] [ebp-4h]

  v3 = 0;
  v11 = sub_2F4320(v17);                        // "\\.\VBoxMiniRdrDN"
  v10 = v11;
  v18 = 0;
```

התת מחרוזת "VBox" המופיעה במחרוזת ככל הנראה דיי מסגירה את VirtualBox אך כדי להיות בטוח גם פה השתמשתי בגוגל.

זיהוי VM גנרי

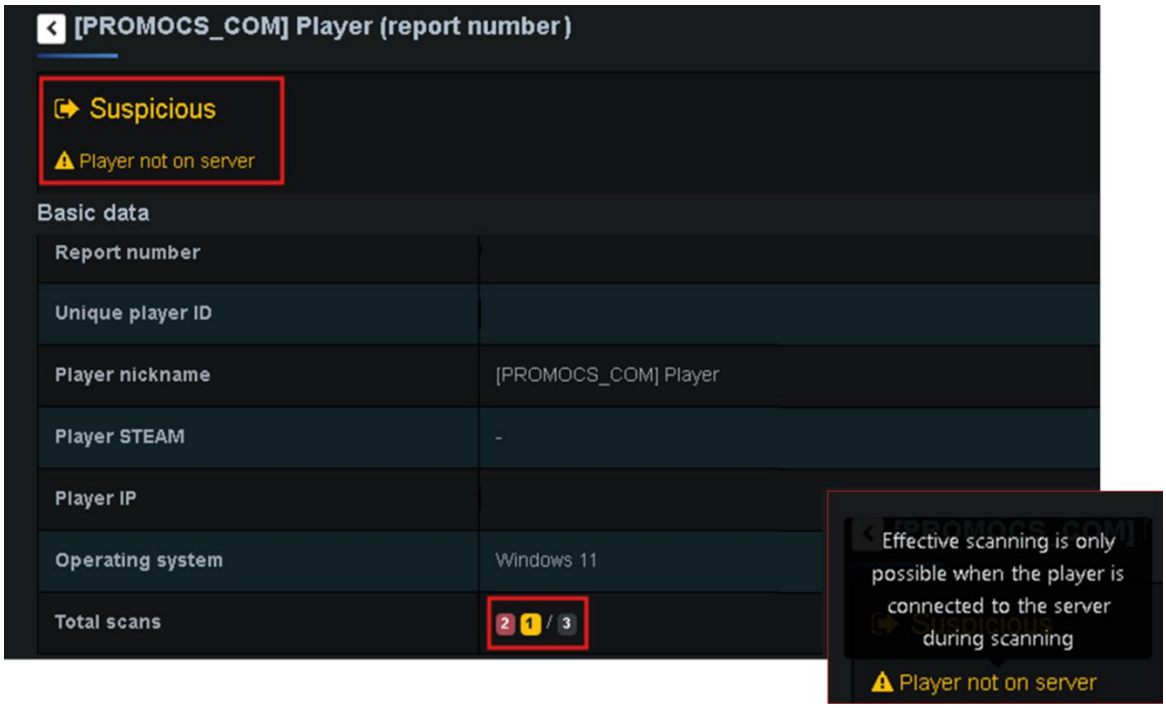
הפונקציה sub_2F3720:

```
v49 = 1;
v13 = sub_2F3BD0(v54);                  // "SYSTEM\HardwareConfig\Current"
v12[1] = v13;
sub_13DF20(&v38, v12, HKEY_LOCAL_MACHINE, v13, 0x20119u);// KEY_READ | KEY_WOW64_64KEY
LOBYTE(v49) = 2;
sub_2622E0(v54);
v6 = v54;
LOBYTE(v49) = 0;
v37 = v12[0] == 0;
v41 = v12[0] == 0;
v47 = v12[0] == 0;
if ( !v12[0] )
{
  v11 = sub_2F3D60(v53);                // "SystemFamily"
  v10 = v11;
  LOBYTE(v49) = 5;
  sub_13E0A0(&v38, v9, v11);            // call to RegGetValueW()
  LOBYTE(v49) = 8;
  sub_2622E0(v53);
  v5 = v53;
  LOBYTE(v49) = 7;
  if ( !sub_238850(v9) )
    goto LABEL_5;
  v34 = sub_2F3E50(v52);                // "Virtual Machine"
  v33 = v34;
```

פונקציה זאת מבצעת בדיקה על סוג המשפחה של היצרן של המחשב (לדוגמה: Latitude, רלוונטי ל-DELL) ע"י שליפת המידע מתוך המפתח הבא בר'גסטרי:

```
HKLM\SYSTEM\HardwareConfig\Current
```

ה-VM שלי רץ על Hyper-V ואכן הערך של SystemFamily אצלי הוא "Virtual Machine". בהמשך, הפונקציה גם בודקת את הערך של SystemProductName. נערוך את הערכים האלה - ננסה לסרוק שוב:



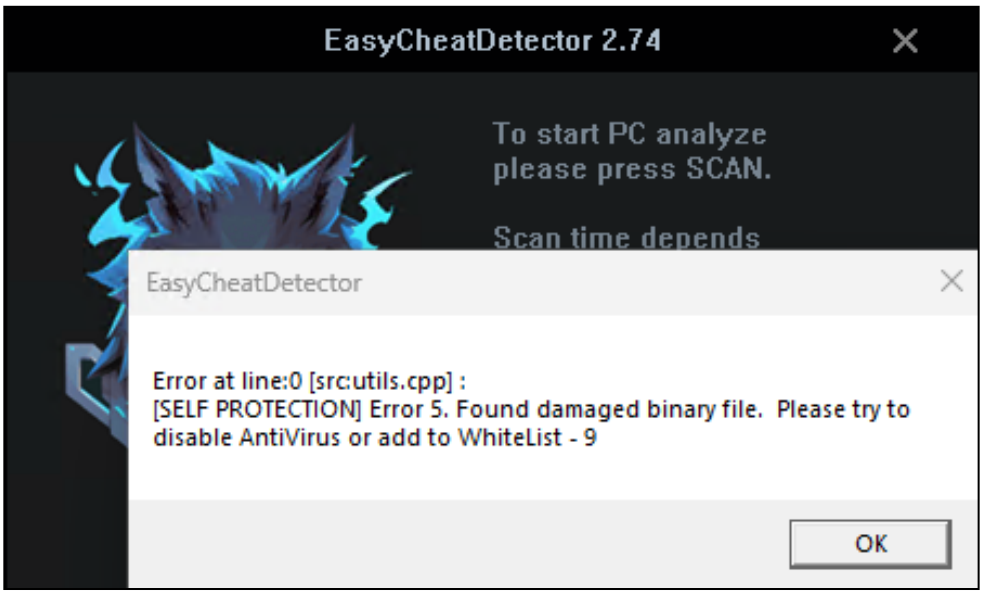
הצלחנו לעקוף את הזיהוי של ה-VM! בנוסף, ניתן לראות את מספר הפעמים שאותו שחקן ביצע סריקה, כלומר שומרים את ההיסטוריה של הסריקות.

מנגנון Self Protect

אגב אם אתם זוכרים, אחת המחרוזות המעניינות שהופיעה היא:

[SELF PROTECT] Found critical error.

בינתיים כל הבדיקות שסקרנו מטריגים את ההודעה על "Can't save offline report because sandbox or debugger found" בהתחלה, אך לא את המחרוזות שמכילה את ה-"SELF PROTECT". לאחר ששמתי Breakpoint בקוד של התוכנה עצמה (בניגוד ל-Breakpoint ששמתי ב-DLL, למשל **kernel32.dll**) השגיאה הבאה קפצה:



כזכור, בעת שימוש ב-Software breakpoint ה-Debugger למעשה מפצפץ את ההוראה ומחליף אותה בהוראה INT3.

מנגנון ה-SELF PROTECT של התוכנה מזהה שהקוד פוצפץ ומקפיץ את השגיאה. בניתוח דינמי באמצעות Debugger אחד הפיצ'רים הבסיסיים והחיוניים הוא שימוש ב-Software breakpoint, כמובן שאפשר להשתמש גם ב-Hardware breakpoint אך כזכור החיסרון העיקרי הוא שהם מוגבלים ל-4 בלבד (מגבלה זאת מתייחסת למעבדי Intel). ישנה סוג של "תחלופה" (לא לגמרי) נוספת והיא שימוש ב-EPT Hooking אבל זה נראלי יותר מתאים למרחיקי לכת.

אז כדי שנוכל לעבוד כמו בני אדם נצטרך למצוא היכן המנגנון הזה ולהשבית אותו. כדי למצוא את המנגנון הזה אני בחרתי באותה השיטה שהצגתי בהתחלה, והיא לשים Breakpoint על הפונקציה MessageBox ולבחון את ה-Stack trace ולראות מי קרא לה.

זאת הפונקציה:

```
if ( sub_2CDEF0(&v42) )
{
  qword_6E97D0 = 0LL;
  memset(&Msg, 0, sizeof(Msg));
  sub_145030(v88, L"-noupdate");
  LOBYTE(v76) = 0x33;
  v70 = sub_2D16F0(v89, v88);
  LOBYTE(v76) = 0x34;
  sub_2622E0(v88);
  v15 = v88;
  LOBYTE(v76) = 0x23;
  if ( v70 )
  {
    byte_6E7782 = 1;
    dword_6E7710 = GetTickCount() + 0x64;
  }
  hThread = CreateThread(0, 0, sub_224BF0, 0, 0, 0);
  while ( GetMessageW(&Msg, 0, 0, 0) )
  {
    TranslateMessage(&Msg);
    DispatchMessageW(&Msg);
  }
  GdiplusShutdown(v43);
  LOBYTE(v76) = 0x37;
  sub_2622E0(v89);
  return 0;
}
else
{
  v30 = v42 + 0x64;
  LOBYTE(v76) = 0x24;
  v17 = sub_22FB30(v87, v42 + 0x64);
  v69 |= 1u;
  v31 = v87;
  v32 = sub_232A10(v81, "Error ", v87);
  v33 = v32;
  LOBYTE(v76) = 0x25;
  v34 = sub_219FC0(v32);
  v46 = v34;
  v35 = sub_22F800(v80);
  v41 = v35;
  LOBYTE(v76) = 0x26;
  v47 = sub_219FC0(v35);
  lpText = v47;
  MessageBoxA(0, v47, v46, 0);  // "Found damaged binary file. Please try to disable AntiVirus or add to WhiteList"
```

אז הפונקציה שעושה את כל הבלאגן היא sub_2CDEF0 ולפי הרפרנסים אליה היא נקראת לא מעט פעמים. מדובר בפונקציה גדולה יחסית, כאשר החלק המרכזי שלה מבצע קריאה של קובץ התוכנה מהדיסק לזיכרון, ולאחר מכן משווה מקטעים מהקוד שרץ בתהליך אל מול המידע המקביל בקובץ שנקרא מהדיסק. במקרה שבו מזוהה פיצפוץ, הפונקציה מחזירה FALSE, והתוכנה תציג את השגיאה שראינו למעלה.

דרכים לעקוף את הבדיקה

1. פיצפוץ פקודת ההשוואה בין האוגרים, כך שתוצאת בדיקת השוויון תמיד תעבור בהצלחה.
2. פיצפוץ הפונקציה שתחזיר **TRUE** כבר בהתחלה.
3. פיצפוץ הקריאות לפונקציה.

פיצ'רים מעניינים

בחלקים הקרובים נסקור דברים מעניינים שהתוכנה עושה. התוכנה עצמה דיי עוזרת לנו ואומרת לנו בכל שלב מה היא עושה (אמנם היא מפענחת את המחרוזות בזמן ריצה אבל זה לא באמת מקשה עלינו):

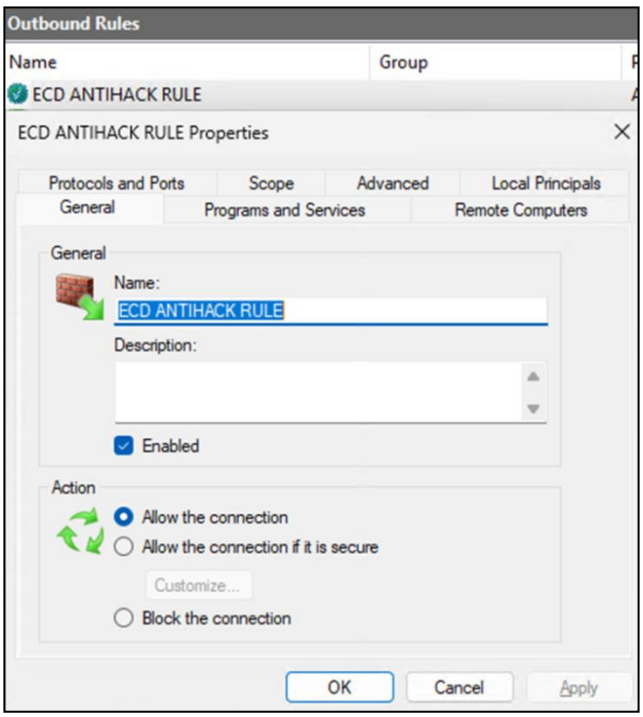
Place	Description
00170F3E: scan+C7E	"01 - Stage: Search game process"
00172F22: scan+2C62	"02 - Stage: Search game installation"
00178C4B: scan+898B	"03 - Stage: Game found PID: "
00179195: scan+8ED5	"04 - Stage: Check for admin [4]"
00179742: scan+9482	"05 - Stage: Adding firewall rules"
00179841: scan+9581	"06 - Stage: Scan processes"
0017BE0E: scan+BB4E	"07 - Stage: Access WMI repository..."
0017C063: scan+BDA3	"08 - Stage: Preparing req"
0017C94F: scan+C68F	"09 - Stage: Read db"
0017DCCA: scan+DA0A	"10 - Stage: Connect web"
00180317: scan+10057	"11 - Stage: Search for mirrors..."
00180530: scan+10270	"12 - Stage: ERROR Mirror not found. Go to offline mode..."
001835FA: scan+1333A	"13 - Stage: Check cheat db"
001840AB: scan+13DEB	"14 - Stage: Get path list..."
00184803: scan+14543	"15 - Stage: Scan registry..."
00184D33: scan+14A73	"16 - Stage: Get OS info..."
001851D8: scan+14F18	"17 - Stage: Try to open process"
002DE800: sub_2DE050+7B0	"18 - Stage: Scan memory 1/3...Progress: 0% (x B of y MB)"
0018638B: scan+160CB	"19 - Stage: Scan modules 1/3"
00186A22: scan+16762	"20 - Stage: Scan modules 2/3"
0018A8C5: scan+1A605	"21 - Stage: Scan modules 3/3"
0018BE29: scan+1BB69	"22 - Stage: Read game server"
0018C963: scan+1C6A3	"23 - Stage: Detect game multiple proc"
0018E965: scan+1E6A5	"24 - Stage: Retrive childs..."
001904CC: scan+2020C	"25 - Stage: Scan for injectors...Progress: 0%" loop
0018EF09: scan+1EC49	"25 - Stage: Scan for injectors...Progress: 0%..."
00192263: scan+21FA3	"26 - Stage: Scan network..."
00192BCB: scan+2290B	"27 - Stage: Scan cpu"
00196AF9: scan+26839	"28 - Stage: Scan modules"
001980C4: scan+27E04	"29 - Stage: Scan virtual mem"
0019C01F: scan+2BD5F	"30 - Stage: Scan executables"
0019C3C0: scan+2C100	"31 - Stage: Scan patches"
0019CCED: scan+2CA2D	"32 - Stage: Read user game info"
0019D5A1: scan+2D2E1	"33 - Stage: Retrive game server..."
0019D95C: scan+2D69C	"34 - Stage: Try to debugging 1/3..."
0019DC91: scan+2D9D1	"34.1 Stage: Try to debugging 1/3 Search hidden cheats..."
0019FE6E: scan+2FBAE	"34.2 Stage: Try to debugging 3/3..."
001A1933: scan+31673	"34.3 Stage: Try to debugging 3/3 Cheat list dumping..."
001A23CB: scan+3210B	"35 - Stage: Scan histories"
001A35DD: scan+3331D	"36 Stage: Scan fastrun processes.."
001A5646: scan+35386	"37 - Stage: Cheat [0x0003AF] Scanning. Please wait...Processed cheats 0%" loop
001A65F4: scan+36334	"38.1 - Stage: Write scan 1-5"
001A928E: scan+38FCE	"38.1.1 - Stage: End preparing"
001A9D3B: scan+39A7B	"38.2 - Stage: Write scan 2-5"
001AAD9E: scan+3AADE	"38.3 - Stage: Write scan 3-5"
001AAFF9: scan+3AD39	"38.4 - Stage: Write scan 4-5"
001AB4A0: scan+3B1E0	"38.5 - Stage: Write scan 5-5"
001ABB77: scan+3B8B7	"39 - Stage: Log actions"
001ABCE5: scan+3BA25	"40 - Stage: Write access info"
001AC0C2: scan+3BE02	"41.1 Stage: Prepare data 1 / 3"

הוספת חוק ל-Firewall

על מנת שהתוכנה תוכל לבצע תקשורת ברשת היא מוסיפה חוק ל-Firewall באמצעות שימוש ב-COM Objects. הפונקציה sub_324640 משתמשת ב-INetFwPolicy2, INetFwRule:

```
app_name_ = SysAllocString(app_name);
hr = CoCreateInstance(
       &rclsid_HNetCfg_FwPolicy2,
       NULL,
       CLSCTX_INPROC_SERVER,
       &riid_INetFwPolicy2,
       (LPVOID *)&ppv);
if ( hr >= S_OK )
{
  get_Rules = (int (__stdcall *)(LPVOID, INetFwRules **))ppv->lpVtbl->get_Rules;
  hr = get_Rules(ppv, &rules);
  if ( hr >= S_OK )
  {
    rule = 0;
    Item = rules->lpVtbl->Item;
    hr = Item(rules, name, &rule);
    if ( hr >= S_OK && rule )
    {
      put_ApplicationName = rule->lpVtbl->put_ApplicationName;
      hr = put_ApplicationName(rule, app_name_);
      ((void (__thiscall *)(INetFwRule *, INetFwRule *))rule->lpVtbl->Release)(rule, rule);
      v32 = hr >= S_OK;
      v54 = hr >= S_OK;
    }
    else
    {
      hr = CoCreateInstance(
             &rclsid_HNetCfg_FwRule,
             NULL,
             CLSCTX_INPROC_SERVER,
             &riid_INetFwRule,
             (LPVOID *)&ppv_);
      if ( hr >= S_OK )
      {
        v22 = (void (__stdcall *)(LPVOID, BSTR))ppv_->lpVtbl->put_Name;
        v22(ppv_, name);
        v21 = (void (__stdcall *)(LPVOID, BSTR))ppv_->lpVtbl->put_ApplicationName;
        v21(ppv_, app_name_);
        v20 = (void (__stdcall *)(LPVOID, int))ppv_->lpVtbl->put_Action;
        v20(ppv_, 1);
        v19 = (void (__stdcall *)(LPVOID, unsigned int))ppv_->lpVtbl->put_Enabled;
        v19(ppv_, 0xFFFFFFFF);
        v18 = (void (__stdcall *)(LPVOID, int))ppv_->lpVtbl->put_Protocol;
        v18(ppv_, 6);
        v17 = (void (__stdcall *)(LPVOID, int))ppv_->lpVtbl->put_Direction;
        v17(ppv_, 2);
        v16 = (void (__stdcall *)(LPVOID, int))ppv_->lpVtbl->put_Profiles;
        v16(ppv_, 0x7FFFFFFF);
        Add = rules->lpVtbl->Add;
        hr = Add(rules, ppv_);
        v31 = hr >= S_OK;
        v54 = hr >= S_OK;
```

נעזרתי בפלאגין ComIDA על מנת לקבל פלט דיקומפייל אצתטי:



חישוב מזהה מחשב (HWID)

הפונקציה calc_hwid שנמצאת ב-0x3095A0 (לפי RVA) מבצעת את שאילתות ה-WMI הבאות:

```
SELECT UniqueId, ProcessorId, Name FROM Win32_Processor
SELECT Manufacturer, SerialNumber FROM Win32_BIOS
SELECT Product, Model, Manufacturer, SerialNumber FROM Win32_BaseBoard
```

לאחר מכן היא משרשרת את התוצאות לפי קבוצות של מחלקות (Classes). לדוגמה, עבור המחלקה Win32_Processor מתקבלת המחרוזת:

```
“0000000000000000Intel(R) Core(TM) 7 160UL”
```

זה שרשור של הערכים: UniqueId (שבדר"כ הוא ריק, נבדק גם מחוץ ל-VM) בתוספת הערך של ProcessorId (המחרוזת: 0000000000000000) ובתוספת הערך של Name (המחרוזת: Intel(R) Core(TM) 7 160UL). על כל מחרוזת שמתקבלת מקבוצת שדות כזו, הפונקציה מחשבת גיבוב MD5. מתוך כל תוצאת MD5 נלקחים 12 הבתים הראשונים, ולאחר מכן כל התוצאות משורשרות יחד למחרוזת אחת, כאשר התו - משמש כמפריד בין הקבוצות.

```
MD5(UniqueId,ProcessorId,Name)[:12] -
MD5(Manufacturer,SerialNumber)[:12] -
MD5(Product,Model,Manufacturer,SerialNumber)[:12]
```

לתוכנה יש גם מנגנון זיהוי לבדיקת האם יש WMI spoofing ע"י בדיקה האם יש הוק על המתודה IWbemClassObject::Get הנמצאת ב-fastprox.dll.

חישוב מזהה HDD ID

תחילה מנסים לחשב את המזהה של ה-HDD באמצעות שימוש ב-IOCTL_STORAGE_QUERY_PROPERTY IOCTL (הערך 0x2D1400). המבנה STORAGE_PROPERTY_QUERY מאותחל כך:

- השדה PropertyId מאותחל ל-StorageDeviceProperty (הערך 0), ערך זה מציין ביצוע שאילתה ל-Device descriptor, שאמור להחזיר לנו מבנה מסוג STORAGE_DEVICE_DESCRIPTOR בבאפר הפלט (lpOutBuffer) המסופק לפונקציה DeviceIoControl.
- השדה QueryType גם כן מאותחל לערך 0 המציין ביצוע שאילתה (בניגוד לערך 1 המציין "האם בקשה זאת נתמכת?").
- השדה AdditionalParameters שגם כן מאותחל לאפסים.

לאחר מכן מתבצעת שליפת המחרוזת של SerialNumber באמצעות השדה SerialNumberOffset המציין את ה-offset של SerialNumber ביחס לתחילתו של הבאפר:

```
hDevice = CreateFileW(lpFileName, 0xC0000000, 3u, 0, 3u, 0, 0);// "\\.\PhysicalDrive0"
if ( hDevice == 0xFFFFFFFF )
{
  sub_147580(a1, word_664F88);
  v47 |= 1u;
  sub_26C5B0(v35);
  v6 = v35;
  v26 = 0;
  return a1;
}
else
{
  memset(InBuffer, 0, 0xC);
  v5 = &v45;
  BytesReturned[4] = &v37;
  v43 = &v37;
  v37 = 0;
  v38 = 0;
  v39 = 0;
  sub_263FF0(0x400);
  LOBYTE(v26) = 7;
  BytesReturned[0] = 0;
  v34 = &v37;
  BytesReturned[3] = v38 - v37;
  nOutBufferSize = v38 - v37;
  BytesReturned[2] = v37;
  BytesReturned[1] = v37;
  lpOutBuffer = v37;
  if ( DeviceIoControl(hDevice, &loc_2D1400, InBuffer, 0xCu, v37, v38 - v37, BytesReturned, 0) )
  {
    v28 = v37;
    v27 = v37;
    v33 = v37;
    if ( v37->SerialNumberOffset )
    {
      v25 = v37;
      v24 = v37;
      lpMultiByteStr = v37 + v33->SerialNumberOffset;
      cchWideChar = MultiByteToWideChar(0, 0, lpMultiByteStr, 0xFFFFFFFF, 0, 0);
```

במידה ושיטת ה-IOCTL נכשלת (למשל בריצה ב-VM מעל Hyper-V השדה SerialNumberOffset חוזר ריק), החישוב מזהה של ה-HDD יתבצע באמצעות שאילתת WMI:

```
SELECT Model, SerialNumber FROM Win32_DiskDrive
```

בדומה לתוצאה הסופית של חישוב ה-HWID, גם כאן מתבצע חישוב MD5 על המזהה HDD, ולבסוף מחזירים את ה-8 בתים הראשונים של ה-MD5.

מי שרוצה להרחיב עוד על HWID ושיטות הסוואה מוזמן לקרוא את החומרים הבאים:

- https://www.unknowncheats.me/forum/anti-cheat-bypass/333662-methods-retrieving-unique-identifiers-hwids-pc.html
- https://www.unknowncheats.me/forum/apex-legends/751041-hwid-spoofer-technical-architecture.html
- https://www.unknowncheats.me/forum/anti-cheat-bypass/323218-smbios-kernel-hook-source.html
- https://www.unknowncheats.me/forum/anti-cheat-bypass/287926-kernelmode-smbios-hardware-id-spoofing.html
- https://www.unknowncheats.me/forum/anti-cheat-bypass/560809-firmwaretablehandler.html
- https://www.unknowncheats.me/forum/anti-cheat-bypass/310941-HDD-serial-spoofer-hooking.html

תקשורת ראשונית עם השרת

התוכנה מנסה לתקשר עם הדומיין fungun.top ואם הדומיין כרגע לא זמין אז היא תנסה עוד כמה דומיינים כמו fungun.net במידה וגם הם לא זמינים אז התקשורת תתבצע מול כתובות IP. לתוכנה יש תמיכה בעדכון גרסה. הבקשה הראשונה שנשלחת מבוססת על המידע הבא:

```
{
"base": "00000000",
"HDD_id": "ffffffff",
"hw_id": "ffffffffffffffffffffffff",
"parent": "explorer.exe",
"scan": "63fd0706",
"scanner": "2.74",
"spoof": 0,
"xpbuild": 0
}
```

חלק מהשדות כבר דיי ברורים ולא צריך להסביר.

- השדה hw_id זה גרסה מקוצרת של ה-HWID שראינו למעלה.
- השדה scan מכיל את תוצאת החישוב (ליתר דיוק - רק את 8 התווים הראשונים) MD5 של הקובץ מהדיסק.
- השדה spoof יכיל את הערך 1 במידה וזוהה WMI spoofing.
- השדה xpbuild אני מניח, מציין האם נעשה שימוש בגרסה של Windows XP.

אך לפני שהבקשה נשלחת, היא מוצפנת באמצעות מפתח RSA ציבורי המוטמע בתוך התוכנה:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuJnBZglM3aQVOb89gEhk
tJqvd64hWQLv6aGAeQiI2OWFDgSAVoYEEciNpSa6eFnQB6C3oeZFVFuthRhkS3hH
4wu4cX8akCBVMGzARUJ2Ra9Qrqw5PNXv9PGXgxl73tz9QpoTFMRPS++DExnL9z7W
+PRiJrEmfT1ok3IXSeH7MLaAllyIwu3AK+UpE4bCtHlFU8LcwyznfuizLwge0zcP
oQQKhRw14gr+drW2E4LKvdk1B0ueRxGMxcv93L147CEytpw2gFXnucxThrlH3yFk
HEYtnDWFbkZ+tzQHCuwCywCc0mJXfb+DmAOktc7dkDZLwC+3yHYYcr1dfseMHqXI
sQIDAQAB
-----END PUBLIC KEY-----
```

דוגמה מלאה של הבקשת POST:

```
POST /ska/ecd.php?method=update HTTP/1.1
Accept-Encoding: deflate, gzip
Host: fungun.top
Accept: application/json
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.5) Gecko/20100101 Firefox/45.0
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Content-Length: 403

data=rsnWJUgIniBImmOWOMPlDzhiuj7FAtF9q8xl6PCYP%25252B76SvcR87wN9yL50u1%25252F2
sS%25252F3UjYHCPRoUwUNORV0RyNFm26dfZVDnBBeCibSMjvO7aQSQJm0HWVkNbDsVUWoko
Jtuyw45CNr2EQm188XxIIoeij0ZbqbRtP7u5RF4kgJPOqAqeLS91Pc%25252BaqC7qzsS6tdBqDwu1w
VkwS7%25252FADGhJQV6NGOPg7JrRKZpQlTzBZefz6q8Qz3h1fb3VHXASAOYYH7e9VlDjT1GOHZA
7CgovBSXNawIIE1Rt%25252FfG8CFQpSLaCGQPYyJi7Ymrqe8Di%25252F2aMm29ciymXyEqaeYjTX
Kea0Hg%25253D%25253D
```

במידה והבקשה תקינה (וגם אם לא שלחנו יותר מדי בקשות בזמן קצר), נקבל תשובה כזאת:

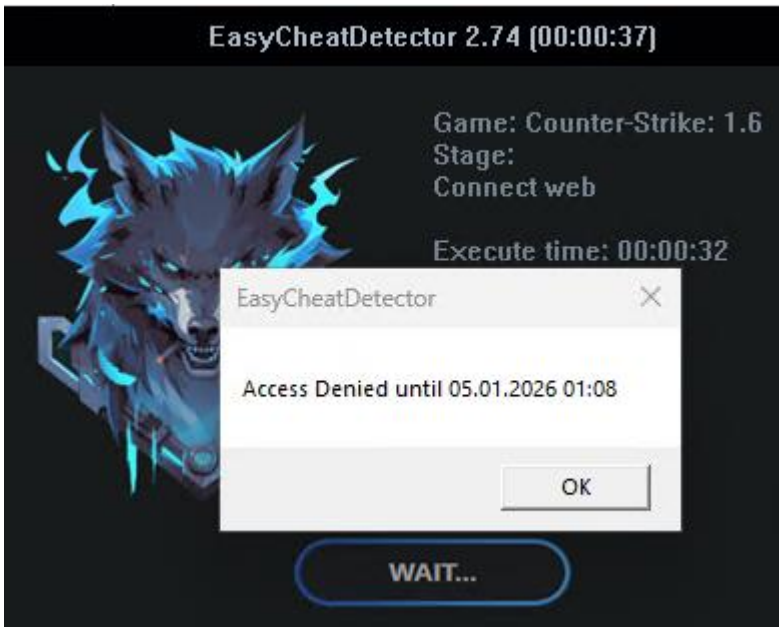
```
{
"success":true,
"unixtime":0,
"scanner1":"https:\/\/github.com\/UnrealKaraulov\/EasyCheatDetector\/releases\/download\/
2.82\/EasyCheatDetector.zip",
"scanner2":"https:\/\/fungun.net\/ska\/ecd.php?method=download",
"base":"Base64(AES-256(cheats_db))",
"scanner3":"https:\/\/fungun.net\/ska\/ecd.php?method=download&exe=1",
"scan_hash":"d5bb165ea0e591aac04294d0c98cada9"
}
```

אחרי קצת מחקר סטטי ודינמי גיליתי שאת המידע הנמצא בשדה base, ניתן לפענח באמצעות מפתח ידוע המוטבע בתוך התוכנה. כאן ניתן למצוא דוגמה לפלט המפוענח - נראה שהוא מכיל חתימות לזיהוי ציטים.

במידה ושלחנו בקשה לא תקינה או יותר מדי בקשות - נקבל תשובה כזאת:

```
{"success": false, "error": "Access Denied until 05.01.2026 01:08"}
```

והתוכנה תציג את זה גם למשתמש:

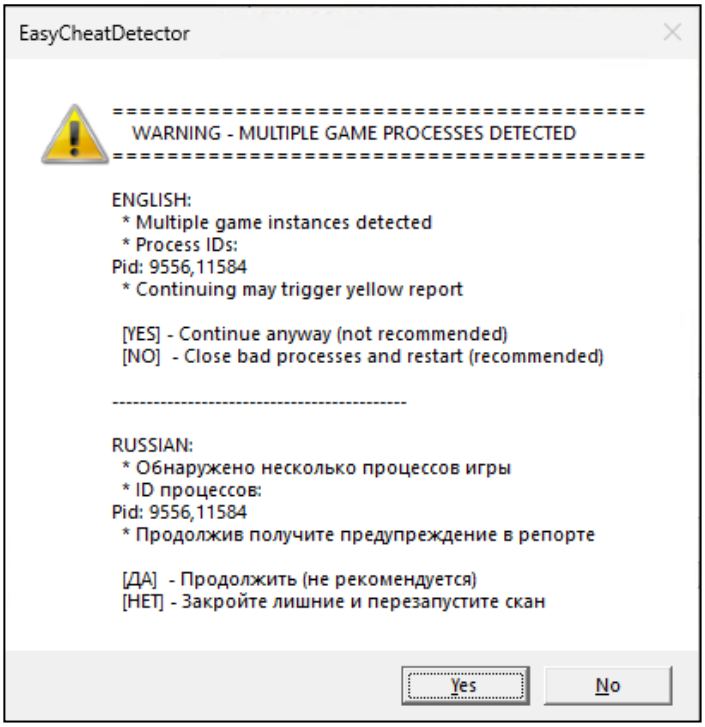


מציאת תהליך המשחק

במידה ורצים מעל Wine, החיפוש של תהליך המשחק יתבצע באמצעות CreateToolhelp32Snapshot, Process32First, Process32Next. במידה ולא רצים מעל Wine, אז החיפוש נעשה באמצעות הפונקציה NtQuerySystemInformation ע"י שימוש ב-SystemExtendedProcessInformation - לא זוכר שנתקלתי בזה בעבר, זאת אינה שיטה נפוצה. התוכנה תחפש אחרי התהליכים הבאים:

```
hl.exe
cstrike.exe
czero.exe
cs.exe
cs_new.exe
cstrike_new.exe
xash.exe
game_start.exe
css.exe
rustclient.exe
cs2.exe
hl2.exe
csgo.exe
left4dead2.exe
```

במידה ונמצאים כמה תהליכים מהרשימה פתוחים ברקע, התוכנה תציג את האזהרה הבאה:



במידה ונמצא התהליך של המשחק (במקרה שלנו - **hl.exe**), לאחר מכן מתבצע חיפוש האם אחד מה-DLLs הבאים טעונים באותו תהליך של המשחק:

```
hw.dll
v8.dll
sw.dll
```

ולבסוף חיפוש תיקיית valve תחת התיקייה הראשית שממנה רץ התהליך של המשחק.

שם השחקן

מציאת שם השחקן מתבצעת באמצעות 3 דרכים שונות:

- שליפה מהנתיב הבא ברגסטרי:

```
HKCU\Software\Valve\Steam\LastGameNameUsed
```

```
update_window_text(v5488, v5486, 0xF);        // "32 - Stage: Read user game info"
LOBYTE(v10587) = 0xEC;
wrap_free_0(v1183);
v10587 = 0x3B9;
wrap_free_0(v1182);
v5484 = sub_209650(v1179);
v5483 = v5484;
v10587 = 0x7EE;
v5482 = sub_2D78B0(v1180, v5484, 0);
v5481 = v5482;
...
  wrap_free_0(v1177);
  v10587 = 0x3B9;
  sub_4F860(v1176);
  v9990 = 0;
  sub_174A80(&v9990);
  v10587 = 0x7F4;
  v5472 = sub_209730(v1175);
  v5471 = v5472;
  LOBYTE(v10587) = 0xF5;
  v10538 = *wrap_RegOpenKeyExW(&v9990, v2923, HKEY_CURRENT_USER, v5472, &g_samDesired);// "Software\Valve\Steam"
  LOBYTE(v10587) = 0xF4;
  wrap_free_0(v1175);
  if ( sub_13BC80(&v10538) )
  {
    something_with_string_3(v2442, &aLastgamenameus);
    LOBYTE(v10587) = 0xF6;
    wrap_RegGetValueW_sz(&v9990, v2549, v2442);// "LastGameNameUsed"
```

- שליפה מתוך הקובץ config.cfg.
- שליפה מהזיכרון של המשחק ע"י שימוש באופסטים ידועים. יש לא מעט מידע על זה ברשת.

מזהה השחקן (steamid)

נשלף באמצעות המידע שנמצא ברגסטרי במפתח הבא:

```
HKCU\Software\Valve\Steam\ActiveProcess\ActiveUser
```

חיבורי רשת

התוכנה משתמשת ב-2 פונקציות דיי דומות על מנת לאסוף את החיבורי רשת מהתהליך של המשחק.

הפונקציה הראשונה enumerate_tcp_connections_for_pid נמצאת ב-0x2D2A50 (לפי RVA).

הפונקציה השנייה enumerate_udp_connections_for_pid נמצאת ב-0x2D2EB0 (לפי RVA).

הפונקציות GetExtendedTcpTable, GetExtendedUdpTable מאפשרות לקרוא את טבלת חיבורי ה-TCP\UDP הפעילים, כולל שיוך ל-PID של תהליך מסוים. הפרמטר המעניין הוא TableClass, הפונקציות מעבירת את הערך TCP_TABLE_OWNER_PID_CONNECTIONS ו-UDP_TABLE_OWNER_PID בהתאמה:

```
pTcpTable = 0;
ExtendedTcpTable = 0;
pdwSize = 0x1C;
if ( pid )
{
  pTcpTable = wrap_alloc(pdwSize);
  if ( pTcpTable )
  {
    if ( GetExtendedTcpTable(pTcpTable, &pdwSize, 0, 2u, TCP_TABLE_OWNER_PID_CONNECTIONS, 0) != 0x7A
      || (wrap_free_6(pTcpTable), (pTcpTable = wrap_alloc(pdwSize)) != 0) )
    {
      ExtendedTcpTable = GetExtendedTcpTable(pTcpTable, &pdwSize, 0, AF_INET, TCP_TABLE_OWNER_PID_CONNECTIONS, 0);
      if ( !ExtendedTcpTable )
      {
        for ( i = 0; i < pTcpTable->dwNumEntries; ++i )
        {
          if ( pTcpTable->table[i].dwOwningPid == pid )
          {
            pStringBuf = 0;
            wrap_memset(v2, 0, 0x800u);
            LOWORD(v14) = 2;
            pAddr[0] = pTcpTable->table[i].dwRemoteAddr;
            HIWORD(v14) = pTcpTable->table[i].dwRemotePort;
            InetNtopW(2, pAddr, &pStringBuf, 0x401u);
            sub_147430(v4);
```

איסוף מידע על התהליכים

התוכנה אוספת מידע על כל התהליכים במערכת באותה דרך שבה היא ניסתה למצוא את התהליך של המשחק. כפי שנראה בהמשך בחלק של הדו"ח המלא, זה המידע שנאסף עבור תהליך:

- הגודל של הקובץ על הדיסק.
- הנתיב המלא שלו.
- כמה מופעים יש לאותו תהליך (רלוונטי לתהליכי svchost.exe המייצגים Services במערכת).

התוכנה משתמשת בשתי שיטות לשליפת הנתיב המלא של כל תהליך במערכת.

- הראשונה מבוססת על GetModuleFileNameEx, שמקבלת HANDLE של תהליך, וגם HMODULE של מודול מסוים, במקרה הזה התוכנה מעבירה NULL, כדי לחלץ את הנתיב של התהליך עצמו. מאחורי הקלעים הפונקציה GetModuleFileNameEx משתמשת בפונקציה NtQueryInformationProcess עם הפרמטר ProcessImageFileNameWin32 כדי לשלוף את הנתיב של התהליך.

- השיטה השנייה מבוססת על הפונקציה GetProcessImageFileName, שגם משתמשת בפונקציה NtQueryInformationProcess אך היא מעבירה לה את הפרמטר ProcessImageFileName.

קצת internals למי שמעוניין: בקריאה עם הפרמטר ProcessImageFileName הנתיב המלא נשלף מתוך השדה SeAuditProcessCreationInfo.ImageFileName הנמצא במבנה EPROCESS. הנתיב המלא חוזר בפורמט NT Path, דוגמה:

```
\Device\HarddiskVolume4\Windows\System32\smss.exe
```

בקריאה לפונקציה עם הפרמטר ProcessImageFileNameWin32, הנתיב המלא נשלף דרך השדה ImageFilePointer במבנה EPROCESS. השדה ImageFilePointer הוא למעשה מצביע למבנה נתונים FILE_OBJECT של התהליך והוא מייצג את הקובץ, איתו ניתן לשלוף את הנתיב המלא באמצעות הפונקציה IoQueryFileDosDeviceName. הנתיב המלא חוזר בפורמט Win32 Path, דוגמה:

```
C:\Windows\System32\smss.exe
```

ההבדל המרכזי בין השניים הוא שמקור המידע שונה.

לאחר שליפת הנתיבים המלאים בשתי השיטות, התוכנה מבצעת בדיקה של שמות הקבצים בלבד, כדי לזהות אי התאמה שעשויות להעיד על מניפולציה שנעשתה ע"י תהליך מסוים, כנראה, על מנת לרמות ועל הדרך להסתיר עקבות. במידה וזוהתה אי התאמה, המידע הזה יישמר כן גם בדו"ח המלא, יירשם תחת קטגוריית "injectors", דוגמה:

```
"injectors": {
  "1": {
    "path": "C:\\Windows\\System32\\calc.exe",
    "old_path": "c:\\windows\\system32\\smss.exe"
  }
}
```

השדה "path" מכיל את הנתיב שהפונקציה GetModuleFileNameEx החזירה. והשדה "old_path" מכיל את הנתיב (אחרי שעבר המרה ל-Win32 Path ע"י שימוש בפונקציה QueryDosDevice) שהפונקציה GetProcessImageFileName החזירה.

איסוף מידע על הדרייברים

את המידע על הדרייברים במערכת התוכנה אוספת באמצעות מעבר על המפתח הבא ברגסטרי:

```
HKLM\SYSTEM\CurrentControlSet\Services
```

על מנת להבדיל בין קרנל דרייבר לבין סתם Service התוכנה מוודא שאכן הערך של Type הוא 1.

```
LODWORD(v6) = sub_5467A0(__PAIR64__(v5, v61), 0x989680LL);
v114 = v6 - 0x2B6109100LL;
something_with_string_3(lpValue_1, L"Type");
LOBYTE(v93) = 0xE;
wrap_RegGetValueW_dword(&hKey_3, lstatus_3_and_value, lpValue_1);
LOBYTE(v93) = 0xF;
wrap_free_2(lpValue_1);
v31 = lpValue_1;
LOBYTE(v93) = 0xD;
something_with_string_3(lpValue_2, L"ImagePath");
LOBYTE(v93) = 0x12;
wrap_RegGetValueW_sz(&hKey_3, lstatus_4, lpValue_2);
LOBYTE(v93) = 0x15;
wrap_free_2(lpValue_2);
v52 = lpValue_2;
LOBYTE(v93) = 0x14;
if ( sub_334C50(lstatus_3_and_value) )
{
  if ( sub_249320(lstatus_4) )
  {
    v51 = 1;
    Type_value = sub_249250(lstatus_3_and_value);
    v87 = Type_value;
    if ( *Type_value == 1 )           // is a Kernel driver?
    {
      v50 = 1;
      v86 = sub_2493E0(lstatus_4);
      v85 = v86;
      if ( *(v86 + 0x10) )
      {
```

כפי שנראה בהמשך בחלק של הדו"ח המלא, המידע שנאסף עבור דרייבר:

- שם המפתח ברגסטרי.
- הגודל של הקובץ על הדיסק.
- הנתיב המלא שלו.
- תיאור (Description) של הדרייבר (לדוגמה: "TCP/IP Protocol Driver" המייצג את הדרייבר tcpip).

למה זה בכלל מעניין לדעת איזה דרייברים קיימים במערכת? תוקפים משתמשים לעיתים בדרייברים זדוניים כדי לעקוף מנגנוני הגנה ולרמות במשחקים. מאחר שדרייברים רצים בקרנל הם בעלי הרשאות גבוהות מאוד, דבר זה מקשה משמעותית על מנגנוני זיהוי צ'יטים. זה תמיד היה משחק של חתול ועכבר ומאוד קשה לנצח תוקף שרץ בקרנל.

שם מחשב

קבלת מידע על שם המחשב מתבצעת באמצעות שאילתת WMI הבאה:

```
SELECT Name FROM Win32_ComputerSystem
```

מערכת הפעלה

קבלת מידע על מערכת ההפעלה מתבצעת באמצעות 2 השאילתות WMI הבאות:

```
SELECT Version FROM Win32_OperatingSystem
SELECT Name FROM Win32_OperatingSystem
```

במידה ומסיבה כלשהי השליפה של המידע באמצעות WMI נכשלה, תתבצע שליפה מהמפתח הבא ברגסטרי:

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName
```

היסטוריית הורדות

התוכנה מחפשת אחרי דפדפנים מסוימים במערכת על מנת לבדוק (בהמשך) את ההיסטוריה של ההורדות.

החיפוש של אותם דפדפנים מתבצע ע"י בדיקה האם הקבצים הרלוונטים שלהם נמצאים בנתיבים הבאים:

```
"%AppData%\Local\360extremebrowser"
"%AppData%\Local\AVG"
"%AppData%\Local\BraveSoftware"
"%AppData%\Local\CCleaner Browser"
"%AppData%\Local\Google"
"%AppData%\Local\Microsoft"
"%AppData%\Local\Vivaldi"
"%AppData%\Local\Yandex"
"%AppData%\Roaming\Mozilla"
"%AppData%\Roaming\Opera Software"
"%AppData%\Roaming\Waterfox"
"%AppData%\Roaming\dolphin_anty"
```

זיהוי צ'יטים

לתוכנה יש לא מעט שיטות לתפוס צי'טים.

חתימות צ'יטים

באופן דומה למה שראינו בחלק של "תקשורת ראשונית עם השרת", במידה והתוכנה לא מצליחה לגשת לשרתים שלה, היא מפענחת את החתימות מתוך סקשן ה-Resource באופן דיי זהה.

הפונקציה scan_for_cheats שנמצאת ב-0x14C610 (לפי RVA) אחראית על זיהוי של צ'יטים מבוססים חתימות. הפונקציה עוברת על כל רשומה (פורמט JSON) שנמצאת במאגר החתימות, כל רשומה מתארת זיהוי של צ'יט מסוים. חלק מהשדות בכל רשומה מקודדים ע"י Base64 בנוסף ל-XOR, הפונקציה תפענח אותן לפני השימוש. המפתח הבא משמש לפענוח השדות arg1, arg2, arg3, arg4 באמצעות XOR:

```
std::string g_PrivateKey1
```

דוגמה:

```
"arg1":
"XVpLXl8eGwIFDx5FFXE0HgU"

output:
"../demoplayer.dll"

"arg2":
"QEdVDQxKRkU"

output:
"33176927"

"arg3" input:
"HgEITlMbFREC"

output:
"multihack"
```

על מנת לפענח את השדה command, משתמשים באותו מפתח + משרשרים אליו את הערך של השדה cheat_id, לדוגמה:

```
std::string g_PrivateKey11627

"command":
"MBwBWVE0FR8MKg5SLz4jGg"

output:
"CheckGameDirHash"
```

השדה command מציין את שיטת הבדיקה למציאת הצ'יט, יש 12 שיטות שונות:

1. CheckGameDirHash - לוקחים את אותו קובץ מהתיקייה הראשית של המשחק (לרוב זה DLL), מחשבים את ה-MD5 של אותו קובץ, לוקחים רק את ה-8 תווים הראשונים של התוצאה ואז משווים אל מול תוצאה ידועה מראש. בדוגמה למעלה, ההשוואה מתבצעת אל מול הערך שנמצא ב-arg2, כלומר עם הערך 33176927. יש רשומות שלא מכילות MD5 לבדיקה - אלא רק שם קובץ, במקרים כאלה במידה והקובץ קיים זה מספיק בשביל להפליל.
2. CheckGameDirHash2 - נראה שיש רשומה אחת כזאת בלבד. השדה is_test מכיל את הערך 1. השדה מכיל שם של קובץ, והשדה arg2 נראה שמכיל תוצאת חישוב של MD5. נראה ששיטה זו לא באמת ממומשת בקוד ולכן היא אינה רלוונטית.
3. SearchCustomCheats - אין הרבה רשומות שמכילות את השיטה הזאת, נראה שהיא גם לא באמת ממומשת בקוד. רק השדה info מכיל מידע, שאר השדות ריקים. השדה is_test מכיל את הערך 1. בחלק מהרשומות השדה is_future גם כן מכיל את הערך 1.
4. SearchDragonFireMem - חיפוש מחרוזות בתהליך של המשחק. מספיקה מחרוזת אחת שנתפסה על מנת להפליל.
5. SearchFoxPing - עוברת על החיבורי רשת (של המשחק) על פי תבנית Regex ומחפשת האם יש תקשורת לדומיין מסוים.
6. SearchGameDirHash - השדה arg1 יכול להכיל: (1) נתיב המציין את התיקייה בתוך התיקייה של המשחק. או, (2) יכיל את התו "/" - המציין מעבר על כל התיקיות בתיקייה הראשית של המשחק. השדה arg2 מכיל את שם הקובץ אותו מחפשים על פי תבנית Regex. במידה והקובץ קיים זה מספיק בשביל להפליל, אין פה השוואה מול MD5. שיטה זאת קצת מזכירה את שיטה מספר 1.
7. SearchHistoryDirHash - עוברת על ההיסטוריה של ההורדות בדפדפן ומחפשת לפי תבנית Regex האם סיומת של קובץ מסוים ירד מאתר מסוים (למשל unknowncheats.me מככב שם בחתימות).
8. SearchMemOrder - חיפוש מחרוזות לפי תבנית Regex לפי סדר (כלומר המחרוזות צריכות להיות אחת אחרי השנייה) בתהליך של המשחק.
9. SearchProcessRegex - מציאת תהליך במערכת לפי תבנית של Regex.
10. SearchRegDirHash - חיפוש ברגסטרי על תבנית Regex תחת המפתחות הבאים:

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\FeatureUsage\AppSwitchedHKLM
\SYSTEM\ControlSet001\Services\bam\State\UserSettings\<SID>
```

11. SearchRootDirHash - על פי הקוד לא נראה ששיטה זאת ממומשת. על פי הרשומות, נראה שהשדה arg1 יכול לציין את התיקייה שבה יש לחפש את שם הקובץ המאוחסן בשדה arg2.
12. SearchUserDirHash - חיפוש אחרי קובץ בתיקייה מסוימת (למשל Desktop, Downloads) בתיקייה הראשית של המשתמש. דוגמה: C:\Users\[USER]\Downloads\wardgos_clear.exe. ישנה תמיכה גם בתבנית Regex לפי שם הקובץ.

במידה והקובץ קיים זה מספיק בשביל להפליל, אין פה השוואה מול MD5. ניתן לעקוף את כל הבדיקות האלה בקלות ע"י שינוי מחרוזות של שמות הקבצים, של המחרוזות עצמם שנמצאות בתוך צ'יט מסוים.

מציאת חלונות חשודים

התוכנה תבצע מעבר על חלונות המשחק באמצעות הפונקציה EnumWindows ולאחר מכן היא תבצע 2 בדיקות:

- האם המשחק נמצב במצב "Run in a window" (כלומר לא "Full Screen").
- האם אחד החלונות נמצא במצב "נראה" - שימוש בפונקציה IsWindowVisible.

```
MACRO_BOOL __thiscall FindConsoleWindowEnumProc(void *this, HWND hWnd, LPARAM GamePid)
{
  DWORD dwProcessId; // [esp+0h] [ebp-4h] BYREF

  dwProcessId = this;
  GetWindowThreadProcessId(hWnd, &dwProcessId);
  if ( dwProcessId == GamePid && IsConsoleWindowClass(hWnd) && !g_hWnd )
    g_hWnd = hWnd;
  return TRUE;
}
```

במידה ונמצאה התאמה לאחת הבדיקות - המידע נרשם בדו"ח תחת המפתח "hacknames" עם התיאור "HACK WINDOW FOUND".

מציאת DLLs חשודים בתוך תהליך המשחק

כחלק ממערך ההגנה שלה, התוכנה מפעילה מנגנון ייעודי לזיהוי DLLs זדוניים בתוך המשחק המנסים לתפוס טרמפ על פונקצית הציור הרשמית כדי להציג מידע אסור על המסך (כמו ESP Overlay). על מנת לבצע זאת, התוכנה מבצעת Attach למשחק כ-Debugger ע"י שימוש בפונקציה DebugActiveProcess ושמה Hardware breakpoint בפונקציה HUD_Frame השייכת ל-client.dll, פונקציה זאת רצה בכל Frame. הגדרת ה-Hardware breakpoint מתבצעת באמצעות שימוש בפונקציה SetThreadContext. ברגע ש-Thread כלשהו קורא לפונקציה המנוטרת, התוכנה תקבל Debug Event, תשלוף את כתובת החזרה מראש המחסנית, היא תבדוק לאיזה DLL שייכת הכתובת כדי לבדוק האם הכתובת הזו נמצאת בתוך רשימת ה-DLLs המוחרגים:

```
hw.dll
v8.dll
engine.dll
client_save.dll
hwpatcher.dll
view_demo_helper.dll
vstdlibnewrevv.dll
libavformat-57.dll
```

```
nitro_api2.dll
steamptc.dll
gtlib.asi
steam_api.dll
```

אם הכתובת שייכת לאחד מהם, מתעלמים. במידה ולא, המידע על אותו DLL יישמר בדו"ח תחת המפתח "hack_modules".

מכיוון שהבדיקות כאן מבוססות על מידע שהגיע מה-PEB וגם מסריקת הזיכרון ע"י שימוש בפונקציה NtQueryVirtualMemory (חיפוש אחרי אזורי זיכרון מסוג MEM_IMAGE והצלבה מול חתימות MZ ו-PE) יש כמה דרכים לשבור את הזיהוי של התוכנה:

- ניתן להסתיר את הרפרנס ל-DLL בתוך ה-PEB ע"י שימוש בטכניקת Unlinking.
- השחתת חתימות ה-MZ ו-PE.
- הזרקת Shellcode - למשל שימוש בטכניקת Reflective DLL Injection - אבל כפי שתראו בהמשך התוכנה יודעת לזהות גם את זה.
- שינוי השם של ה-DLL לאחד השמות של ה-DLLs המוחרגים (הערה: זה דיי מחשיד שיש 2 DLLs עם אותו שם) או, שיטה יותר חשאית - פיצפוץ DLL מרשימת המוחרגים.

למידע נוסף על הזרקות קוד:

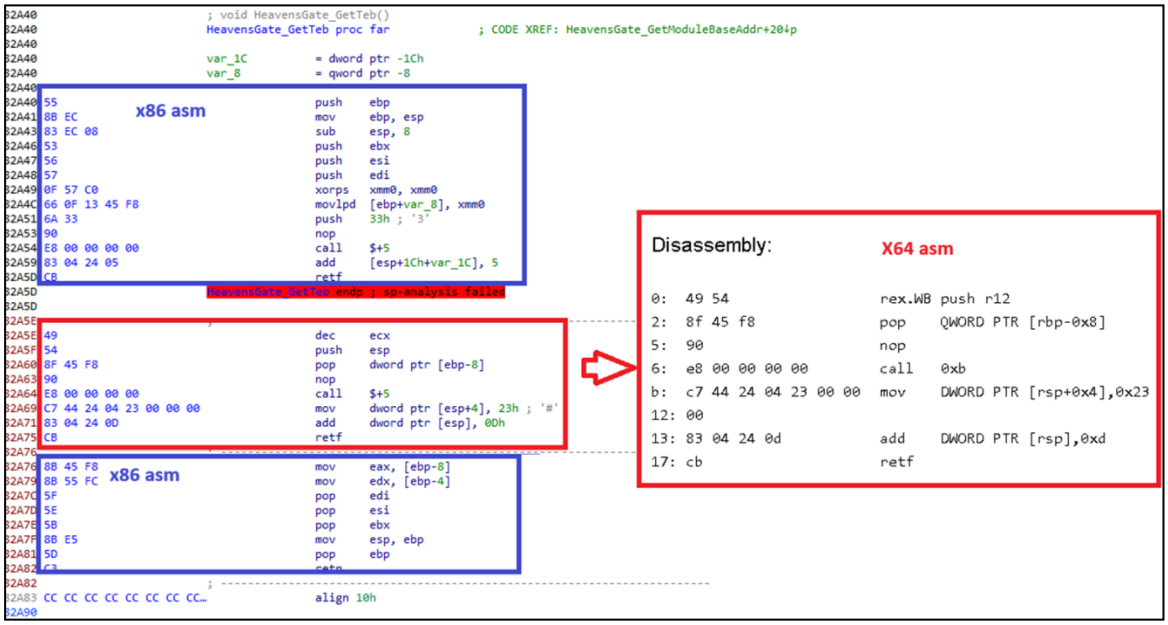
- https://www.digitalwhisper.co.il/files/Zines/0x98/DW152-2-CodeInjection.pdf
- https://www.cyberark.com/resources/threat-research-blog/masking-malicious-memory-artifacts-part-iii-bypassing-defensive-scanners
- https://www.huntandhackett.com/blog/concealed-code-execution-techniques-and-detection
- https://www.unknowncheats.me/forum/anti-cheat-bypass/321071-erasing-pe-header-idea.html

מציאת אזורי זיכרון חשודים - פתחו של גן עדן

התוכנה עוברת על הזיכרון של תהליך המשחק ומחפשת אחרי אזורי זיכרון המוגדרים RWX (קריאה, כתיבה, הרצת קוד) שאינם שייכים למודול מסוים (כלומר Type שווה ל MEM_PRIVATE). את המידע על הזיכרון של תהליך המשחק היא אוספת באמצעות שימוש בפונקציה NtQueryVirtualMemory. אממה, התוכנה קוראת לפונקציה בדרך חשאית משהו. התוכנה עושה שימוש בטכניקת Heaven's Gate.

למה דרך חשאית? כי גם אם אנחנו יודעים שהתוכנה עושה שימוש כלשהו בקריאת מערכת כלשהי - במידה ונשים Breakpoint על אותה קריאת מערכת - זאת לא בהכרח הקריאה שהיינו מעוניינים בה. מכיוון שמדובר בתהליך 32 ביט ה-Debugger ימקם את ה-Breakpoint בגרסה ה-32 ביט של Ntdll. בנוסף, לא כל

Debugger תומך (למשל שימוש ב-Breakpoint בגרסה 64 ביט של Ntdll, או למשל לבצע Single step במעבר בין 32 ל-64 ולהפך) בדיבאגינג של Wow64 - למשל x32dbg לא תומך בזה. בעוד ש-WinDbg כן.



בתמונה למעלה רואים דוגמה לפונקציה המשתמשת בטכניקת Heaven's Gate על מנת לשלוף את כתובת ה-TEB. נראה שאותו קטע קוד בצד ימין (מוקף בצבע אדום) זהה לקטע קוד הנמצא ב-GitHub.

למי שרוצה internals על Heaven's Gate:

- http://blog.rewolf.pl/blog/?p=102
- https://cloud.google.com/blog/topics/threat-intelligence/wow64-subsystem-internals-and-hooking-techniques/

לסיום, במידה ונמצאו קטעי זיכרון העומדים בתנאים שהוזכרו למעלה - המידע עליהם יישמר בדו"ח תחת המפתח "map_injections".

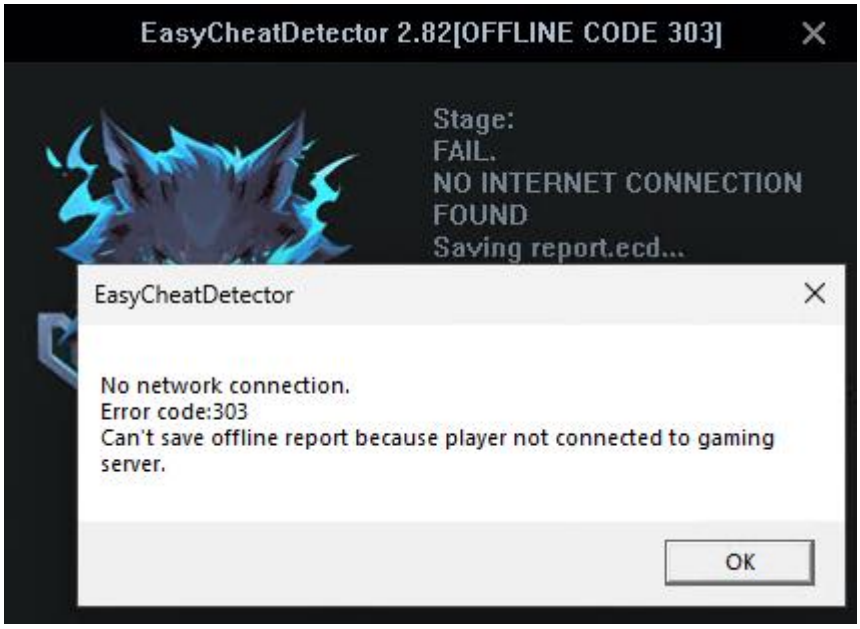
בדיקות נוספות

- בדיקה האם אחד ה-DLLs בתהליך מתחיל במחרוזת "Injected_" - בבדיקה הזאת יש לתוכנה באג, הבדיקה היא Case-sensitive והתוכנה משתמשת בשם של ה-DLL באותיות קטנות.
- מעבר על אזורי הזיכרון באמצעות הפונקציה NtQueryVirtualMemory, וחיפוש אחרי PE ו-MZ, לאחר מכן ניסיון שליפת שם המודול ע"י שימוש בפונקציה NtQueryVirtualMemory עם הפרמטר MemoryMappedFilenameInformation, במידה והיא לא מצליחה לשלוף את השם של המודול מסיבה כלשהי - דיווח.
- חיפוש אחרי הוקים ב opengl32.dll.

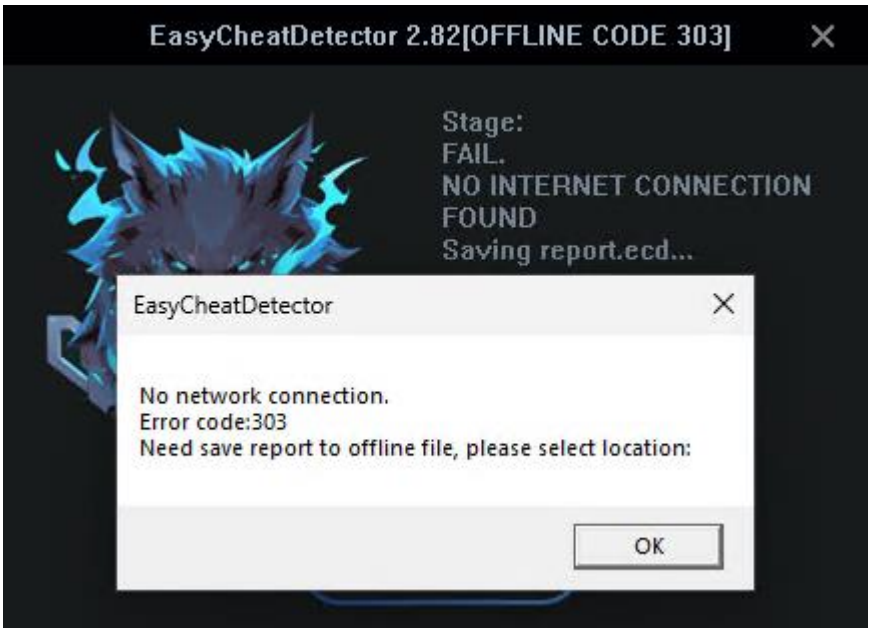
הדו"ח המלא

דו"ח מקומי

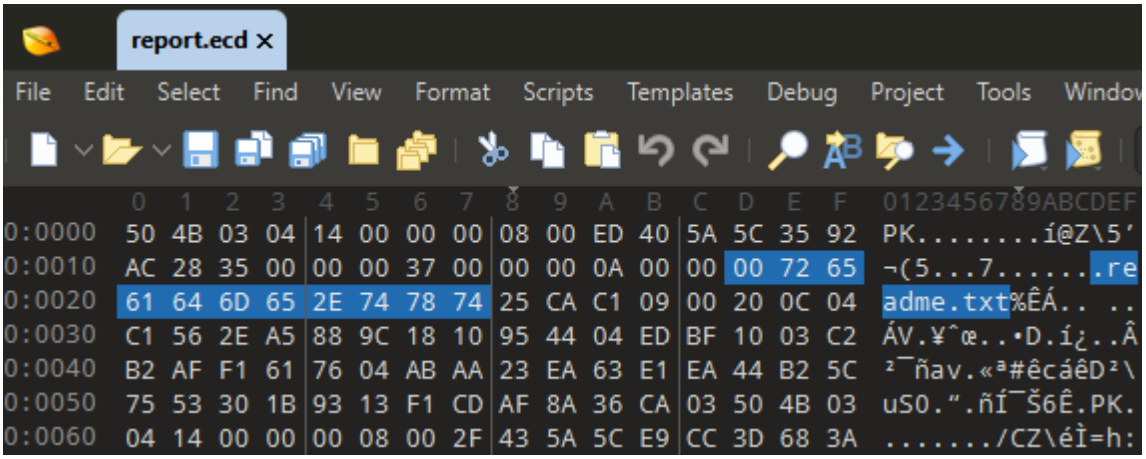
כזכור, במידה ואין אינטרנט אז הדו"ח של הסריקה אמור (כל עוד לא התוכנה לא זיהתה Sandbox או Debugger בבדיקות השונות) להישמר מקומית:



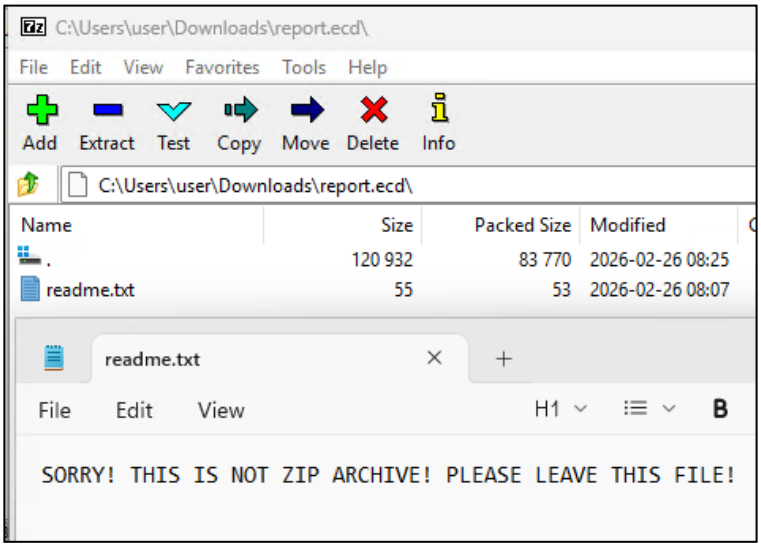
נראה שהתוכנה מסרבת לשמור את הדו"ח אם השחקן לא מחובר לסרבר מסוים. בעיה. מה נעשה? נתחבר לשרת מקומי (Localhost, עם בוטים) וככה עקפנו את הבדיקה הזאת:



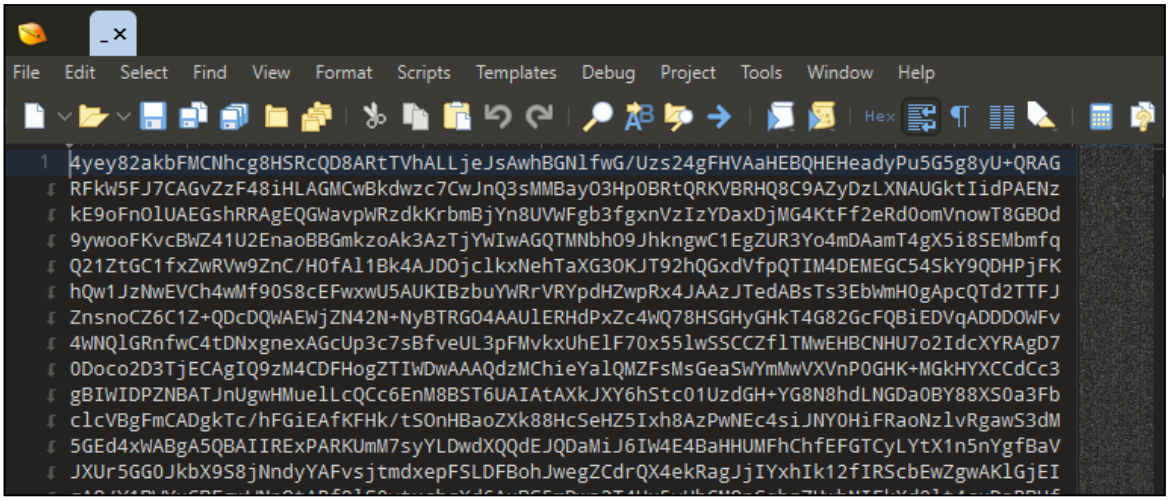
לאחר שנבחר מיקום, נראה קובץ בשם report עם סיומת ecd, מה זה הקובץ הזה?



נראה כמו קובץ Zip, בנוסף ניתן לראות שיש שם את המחרוזת readme.txt. נחלץ את מה שיש ב-Zip, נתחיל בבדיקה מה יש בקובץ readme.txt:



הקובץ הבא הוא ".":



נראה שהמידע מקודד באמצעות Base64, אך ניסיון לפענח את המידע לא מביא פלט קריא. לאחר קצת מחקר סטטי ודינמי הצלחתי לפענח את המידע וכתבתי סקריפט שמקבל קובץ report.ecd ומפענח את המידע, ניתן לראות כאן דוגמה לדו"ח המלא.

דו"ח מקוון

במידה ויש אינטרנט במכונה, הדו"ח של הסריקה אמור להישלח באופן מקוון לאותו שרת שראינו בחלק של "תקשורת ראשונית עם השרת", הבקשה נראית כך:

```
POST /ska/ecd.php?method=report HTTP/1.1
Accept:: */*
Cache-Control: no-cache
Connection: close
Content-Type: application/x-www-form-urlencoded
Host: fungun.top
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.5) Gecko/20100101 Firefox/45.0

data=...
```

המידע מוצפן/מקודד באותו אופן שהוא נשמר מקומית.

במידה והבקשה תקינה (וגם אם לא שלחנו יותר מדי בקשות בזמן קצר), נקבל תשובה כזאת:

```
{
"success": true,
"report_id": 123
}
```

סיכום

במאמר זה סקרנו יחד את תוכנת האנטי צ'יט Easy Cheat Detector וראינו שהמטרה שלה היא לא לחסום פעולות באופן אקטיבי, אלא רק לזהות ולדווח על צ'יטים. במהלך הניתוח ראינו איך התוכנה מנסה לגלות אם מריצים אותה ב-Debugger או בסביבה וירטואלית, ואיך היא אוספת מידע כמו מזהי חומרה (HWID), דרייברים, ורשימת תהליכים במערכת. בנוסף, החלק העיקרי והמעניין שסקרנו הוא השיטות שהתוכנה בודקת האם השחקן משתמש בצי'טים.

בסוף התהליך התוכנה אורזת ומצפינה את כל המידע הזה ושולחת אותו לשרת שלה. השורה התחתונה שראינו יחד היא שלמרות כל הבדיקות, עדיין אפשר לעקוף אותה יחסית בקלות.

על המחבר

דור נאמני (Dor00tkit), קורא אדוק של Digital Whisper עוד מגיל צעיר, תמיד נמשכתי לעולם ה-Low Level והנדסה לאחור בעיקר מהמאמרים שפורסמו פה. יש משהו מיוחד בלחזור למקום שליווה אותי בתחילת הדרך. עבורי, כתיבת המאמר הזה היא סוג של סגירת מעגל והזדמנות קטנה להחזיר למקום שהשפיע עליי כל כך הרבה.

תודות

לבורא עולם, למשפחה שלי, למייסדים והעורכים של Digital Whisper, ניקס וקהילת האקינג IL. כסיף, רז(יאל), ירדן שפיר, ליאור קשת, d4d, Amirag, elicn, zvikam, Dvd848, b0rlord, Sub, reaction, megabeets, Antartic, Zerith , Rainbow_Dash, omerk2511, Improvement, J_Cobra, CescFabregas, Syst3m ShuTd0wn, P, iTK98 ,Alex Ionescu, XenoKovah - OpenSecurityTraining2, LiveOverflow, Gynvael Coldwind, Sinaei, Daax, ergot86, Satoshi Tanda, Pavel Yosifovich, Michael Maltsev. מ.ב, ה.ג, א.ד, ק.ד, ע.פ, א.י, ע.מ, ט.ח, א.ר, ע.ז, ר.ק, ב.ה, א.מ, ע.ד, י.ש, ת.ב, ב.ש, ח.פ, ש.ג, ז.ר.

מקורות מידע

- https://www.digitalwhisper.co.il/files/Zines/0x04/DW4-3-Anti-Anti-Debugging.pdf
- https://www.digitalwhisper.co.il/files/Zines/0x58/DW88-3-AntiReversing.pdf
- https://www.digitalwhisper.co.il/files/Zines/0x5A/DW90-3-PE.pdf
- https://www.digitalwhisper.co.il/files/Zines/0x10/DW16-2-SignatureDetectionBypass.pdf
- https://www.unknowncheats.me/

אמרנו לחכות. ההמתנה נגמרה.

על קריפטוגרפיה פוסט-קוונטית: למה ה-Harvest Now, Decrypt Later הוא כבר לא מדע בדיוני אלא בעיה תפעולית

מאת גיא ברנהרט-מגן (Profero, CTO)

רקע

בינתיים המתמטיקה התקדמה. שני מאמרים שפורסמו במרץ 2026 הראו ששבירת הצפנה מבוססת עקומים אליפטיים דורשת היום הרבה פחות ממה שחשבנו, וזה מספיק קרוב כדי שזו תהיה בעיה הנדסית קצרת-טווח ולא שאלה רחוקה בפיזיקה. תקני ההצפנה אושרו, סופיים ומוכנים לפריסה. אם אתם מאחסנים או מעבירים מידע רגיש שמוגן בהצפנה א-סימטרית חלשה, תוכנית המעבר שלכם כבר צריכה להיות קיימת. אם חוויתם אירוע ומידע מוצפן נגנב, ייתכן שהוא כבר בתור לפענוח עתידי. בדקו את ה-HSM שלכם, תתחילו עכשיו.

הקדמה

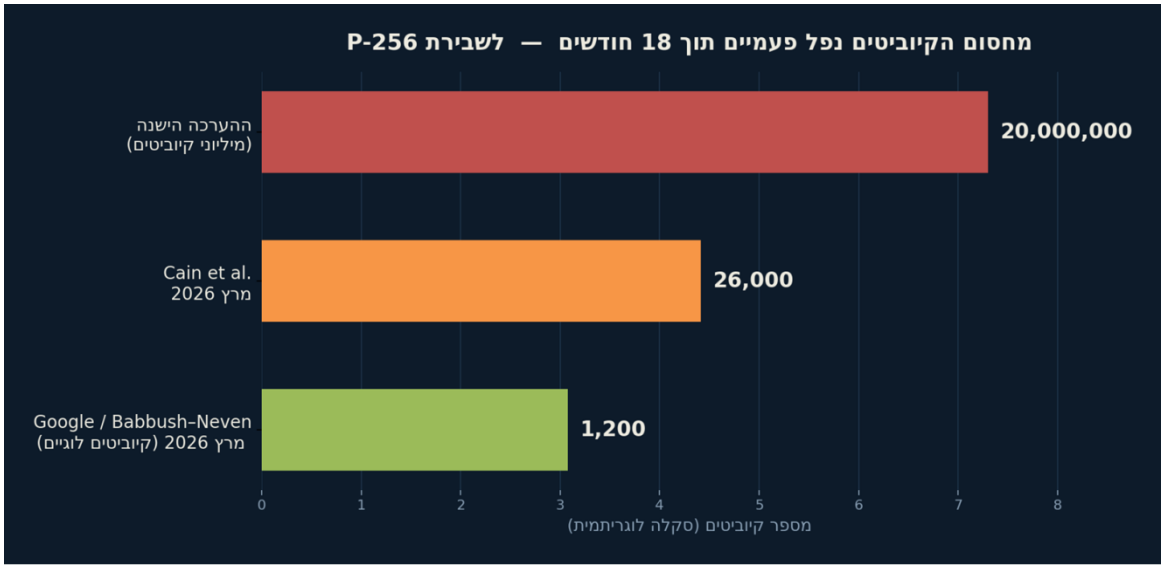
לפני כשנה כתבתי פוסט בשם "להבין קריפטוגרפיה קוונטית: להפריד בין עובדות לדמיון". הטענה הייתה פשוטה: ההייפ סביב מחשבים קוונטיים ששוברים הצפנה מודרנית התקדם הרבה יותר מהר מהמציאות (אני לא מאמין גדול ב-FUD). מחשבים קוונטיים חזקים היו רחוקים, תקני PQC עוד היו בגיבוש, והעצה הייתה ברורה: תישארו מעודכנים, אל תיכנסו לפאניקה, וחכו שהתקנים יתייצבו לפני שאתם מהמרים על תשתית.

זו הייתה ההחלטה הנכונה לאותו הזמן. אני כותב את המאמר הזה כי המציאות השתנתה, וכדאי שתבינו איך. בהמשך נעבור על מה שזז מבחינה טכנית, על האיום המעשי שכבר קיים גם בלי מחשב קוונטי בהווה, ועל מה שאפשר לעשות כבר עכשיו.

מה השתנה?

שני המניעים המרכזיים טכניים, לא פוליטיים.

דרישת הקיוביטים צנחה. כשכתבתי את הפוסט המקורי, שבירת הצפנת עקומים אליפטיים ב-256 ביט דרשה מחשב קוונטי ענק עם תיקון שגיאות: מיליוני קיוביטים פיזיים לפי רוב ההערכות. ב-31 במרץ 2026 פרסמו ראיין באבוש והרטמוט נוון מ-Google מחקר שמראה שאלגוריתם Shor נגד P-256 דורש היום פחות מ-1,200 קיוביטים לוגיים ו-90 מיליון שערי Toffoli. זו הפחתה של פי 20 במספר הקיוביטים הפיזיים. כמה ימים אחר כך הראו Cain ושותפיו שעם ארכיטקטורת אטומים ניטרליים וקישוריות לא-מקומית אפשר לפתור את בעיית הלוגריתם הבדיד של P-256 בתוך כמה ימים, על מערכת עם כ-26,000 קיוביטים פיזיים. זה כבר נראה כמו הנדסה קצרת-טווח, לא פיזיקה ספקולטיבית.



[תרשים להמחשה - סקלה לוגריתמית, על בסיס Neven & Babbush ו-Cain et al. (2026)]

קונצנזוס המומחים השתנה. פיליפו ולסורדה, מהנדס קריפטוגרפיה שאני מכבד את שיקול הדעת שלו, שינה את עמדתו לגבי לוחות הזמנים של CRQC. ההערכה החדשה שלו מצביעה על הסתברות משמעותית למחשב קוונטי רלוונטי-קריפטוגרפית (CRQC - Cryptographically Relevant Quantum Computer) עד 2029. הוא אומר במפורש שהנטל עכשיו על הספקנים להוכיח הסתברות אפסית, לא על מחנה הדחיפות להוכיח ודאות.

הערה למי שרוצה לדייק: קיוביט לוגי הוא קיוביט "מתוקן-שגיאות" שמורכב מהרבה קיוביטים פיזיים רועשים. לכן פחות מ-1,200 הקיוביטים הלוגיים אצל Google מתורגמים למספר פיזי גדול בהרבה, והעבודה של Cain ושותפיו, שמדברת על כ-26,000 קיוביטים פיזיים, היא הערכה בארכיטקטורה אחרת לגמרי. שתי התוצאות מצביעות לאותו כיוון: המספרים מצטמצמים, וכל אחת מהן לבדה כבר מקרבת את הסף לטווח שאפשר לדמיין אותו כפרויקט הנדסי.

PQC כבר פרוס, לא רק מתוקנן. NIST סיימו את התקנים FIP, 203, 204 ו-205 באוגוסט 2024. אבל מה שאני מצביע עליו הוא מה שקרה אחרי: Chrome שלח את ML-KEM ב-TLS כברירת מחדל באמצע 2024, ו-

OpenSSL ו-BoringSSL הצטרפו. עד 2025 החלפת מפתחות היברידית פוסט-קוונטית כבר הייתה זמינה ב-stack שרוב הארגונים מריצים ממילא. הטענה שטענתי בפוסט המקורי, לחכות שהתקנים יתייצבו לפני שמהמרים על תשתית, כבר לא תקפה. התקנים התייצבו, הספריות מעודכנות, והפער בפריסה נסגר. לא נשאר חסם טכני.

רגע, מה הם ML-KEM ו-ML-DSM? ML-KEM (לשעבר Kyber, מתוקנן כ-FIPS 203)? אלו מנגנוני אנקפסולציית מפתחות מבוסס סריגים, מחליף להחלפת מפתחות כמו ECDH. ML-DSM (לשעבר Dilithium, FIPS 204) הוא סכמת חתימה דיגיטלית מקבילה, מחליף ל-ECDSA ו-RSA. SLH-DSA (FIPS 205) הוא חתימה מבוססת hash, גיבוי שמרני יותר. כל השלושה תוכננו כך שגם מחשב קוונטי לא ישבור אותם ביעילות. OpenSSL ו-BoringSSL, אגב, הן ספריות ה-TLS שמריצות חלק גדול מהאינטרנט, ולכן הרגע שבו הן מוסיפות תמיכה הוא הרגע שבו האלגוריתם נעשה זמין בפועל לכולם.

האיום שלא צריך את 2029 כדי להתחיל

לאנשי Incident Response יש כאן חשיפה ספציפית. יריבים מדינתיים (וכמה קבוצות פשע ממומנות היטב) לא צריכים מחשב קוונטי היום כדי להרוויח מאחד ב-2029. הם צריכים שהמידע עדיין יהיה שווה כשהחישוב יגיע. זה המודל של "אסוף עכשיו, פענח אחר כך" (Harvest Now, Decrypt Later): אוספים תעבורה מוצפנת היום, שומרים אותה, ומפענחים כש-CRQC נעשה זמין.

[תרשים להמחשה]

אם הארגון שלכם חווה אירוע (מידע שנפרץ, אירוע שרשרת אספקה, חדירה מדינתית), יש הסתברות סבירה שמידע מוצפן הוצא יחד עם plaintext. מפתחות הסשן של TLS, לחיצות היד של VPN, ארכיוני מייל מוצפנים, פעולות חתימת קוד: כל זה יכול לשבת באחסון קר על תשתית של מישהו אחר, ולחכות.

לא כל מידע נולד שווה כאן. רשומה רפואית, סוד מסחרי, מפתח חתימה ארוך-טווח או מידע מסווג עדיין יהיו רגישים בעוד חמש או עשר שנים, ולכן הם המטרה האטרקטיבית ל-Harvest Now, Decrypt Later. לעומת זאת, תעבורה שהערך שלה פג אחרי שעה הרבה פחות מעניינת ליריב סבלני. כשאתם מתעדפים, השאלה הראשונה היא כמה זמן המידע צריך להישאר חסוי, ולא רק כמה הוא רגיש היום. תגובה לאירוע ממוסגרת בדרך כלל סביב מה שתוקפים יכולים לעשות היום. האיום הזה ממסגר מחדש את השאלה: מה המידע שכבר יש להם יכול לעשות לכם בעוד שלוש שנים?

הצפנה סימטרית ו-HSM

אלגוריתם Grover חוצה את אורך המפתח האפקטיבי תחת מתקפה קוונטית, הסיכון האמיתי מרוכז בהצפנה א-סימטרית: החלפת מפתחות (ECDH, RSA), חתימות דיגיטליות (ECDSA, RSA-PSS), וכל מה שבנוי על הקושי של לוגריתמים בדידים או פירוק לגורמים. אלה האלגוריתמים ש-Shor שובר ביעילות.

שיקול חומרה אחד ספציפי: אם הארגון משתמש ב-HSM (Hardware Security Modules), לאחסון מפתחות, חתימת קוד או פעולות CA, בדקו אם ה-Fireware ותמיכת האלגוריתמים מכסים את ML-KEM ו-ML-DSM. הרבה HSM בייצור היום לא תומכים. מחזורי השדרוג של HSM ארוכים ולפעמים דורשים החלפה פיזית. אם CRQC יגיע לפני שה-HSM שלכם תומך באלגוריתמים פוסט-קוונטיים, שכבת הגנת המפתחות הופכת לצוואר הבקבוק. הוסיפו סקירת HSM לתכנית העבודה שלכם.

מה לעשות עכשיו?

זו בעיה של תכנון, ואין צורך בפאניקה. אבל היא חייבת להתחיל עכשיו, כי לוח הזמנים של מעבר ההצפנה הא-סימטרית ארוך.

1. **רשימת מצאי קריפטוגרפית.** מפו איפה הצפנה א-סימטרית נמצאת בשימוש: נקודות סיום TLS, קצות VPN, תהליכי חתימת קוד, שורשי אטסטציה של חומרה, טוקני אימות API, רשויות אישורים. אי אפשר להעביר את מה שאתם לא רואים, ורוב הארגונים מגלים שיש להם הרבה יותר חשיפה ממה שציפו ברגע שהם מסתכלים.
2. **תעדפו החלפת מפתחות.** ML-KEM זמין, מתוקנן ופריס ב-TLS 1.3 דרך מצבים היברידיים כבר היום. זו העדיפות הגבוהה ביותר, כי היא מנטרלת ישירות את איום ה-Harvest Now, Decrypt Later עבור תעבורה עתידית. תעבורה שמוגנת ב-ML-KEM היום בטוחה גם אם CRQC יגיע ב-2029.

3. **תכננו מעבר חתימות.** ML-DSM הוא המחליף של ECDSA ו-RSA-PSS. מחזורי החיים של אישורים אומרים שזה לוקח יותר זמן: קחו בחשבון שדרוגי CA, תמיכת HSM ותאימות לקוחות. תתחילו את מחזור התכנון עכשיו כדי שהביצוע לא ייכפה עליכם תחת לחץ.
4. **הוציאו משירות שורשים קלאסיים.** מערכות אטסטציה של חומרה ושורשי אישורים ארוכי-חיים הם הכי קשים לרוטציה. הם דורשים את זמן ההיערכות הארוך ביותר ואת הגיבוי הארגוני הגדול ביותר. שימו אותם על מפת הדרכים היום.

המספרים האלה לא נועדו להפחיד אלא לתעדף. ארגון שמתחיל היום באינוונטר ובמפת דרכים ייכנס לחלון של 2029 עם בחירה, לא עם חירום. זה ההבדל היחיד, והוא כל ההבדל.

הזווית של תגובה לאירוע

אם חוויתם חדירה מאומתת (במיוחד כזו עם חשד למעורבות מדינתית), היקף האירוע יכול להיות גדול יותר ממה שהערכת נקודת-הזמן הראתה. מידע מוצפן שהוצא ב-2023, ב-2024 או ב-2025 נמצא עכשיו בתחום של פענוח קוונטי עתידי. זה לא משנה את המיטיגציה שכבר השלמתם, אבל זה משנה את חישוב הסיכון ארוך-הטווח למידע רגיש שנחשף.

תגובה לאירוע תמיד כללה הערכה של איזה מידע נלקח. מכאן והלאה היא צריכה לכלול גם את השאלה: מה הייתה ההגנה הקריפטוגרפית על המידע הזה, וכמה זמן ההגנה הזו תחזיק מעמד?

סדר עדיפויות מעשי

סדר העדיפויות פשוט: קודם החלפת מפתחות, אחר כך חתימות, ובסוף שורשים ארוכי-חיים. החלפת מפתחות עוצרת את הדימום של Harvest Now, Decrypt Later כאן ועכשיו; חתימות ושורשים הם עבודת תשתית ארוכה שצריך בעיקר להתחיל בזמן.

ה-ZKP - מה שסופר ומה שקרה

הוכחת אפס-ידע (ZKP - Zero-Knowledge Proof) היא פרוטוקול שבו צד אחד משכנע צד אחר שטענה נכונה, בלי לחשוף שום מידע מעבר לעצם נכונותה. גוגל ייעלה את אלגוריתם Shor (זה ששובר הצפנה א-סימטרית כמו RSA ועקומים אליפטיים ברגע שיש מחשב קוונטי עם מספיק קיוביטים), אבל במקום לפרסם את המעגל המיטבי עצמו היא פרסמה הוכחת אפס-ידע על העלות שלו: הנה התוצאה, בלי לחשוף איך. זו דרך מעניינת לחשוף יכולת בלי למסור את השיטה, וזה החלק הבאמת מעניין בכל הסיפור.

הסיפור כפי שהתגלגל ב X

ב X-זה התגלגל thread-כ דרמטי (של Charles Guillemet, _P3b7@) - גוגל ייעלה את Shor, ממשל ארה"ב חסם את פרסום המאמר המלא, אז גוגל פרסמה הוכחת אפס-ידע במקום. ואז מישהו פתח תחרות לשחזר את התוצאה בעזרת AI. מודל שפה סורק מרחב עצום של מעגלים קוונטיים, כל אחד מועמד לאופטימיזציה של שור ובודק אם הוא מנצח את השיא הקודם. החלק החכם: השתמשו במאמת ה-ZKP של Google כפונקציית התגמול. אין תוצאות שווא, ומסתבר שזה אות יעיל מאוד. לפי ה-thread, תוך פחות מיומיים הקהילה שחזרה את התוצאה של גוגל וחמישה-עשר ימים אחר כך המודלים כבר 44% מעבר לה.

סיפור מצוין. הוא גם דוחס שני אירועים שונים מאוד לקשת אחת חלקה. נפריד אותם.

מה Trail of Bits באמת עשו

החלק שתפס כותרות הוא ש-Trail of Bits "ניצחו" את ההוכחה של גוגל. רק שזה לא היה הישג קוונטי, הם מצאו באגים של בטיחות-זיכרון ולוגיקה בקוד המאמת של גוגל שכתוב ב-Rust, רץ כ-guest בתוך zkVM מסוג SP1 , וזייפו הוכחה שמדווחת מספרים טובים יותר משל גוגל כולל אפס שערי Toffoli. הניצול לא נגע במעגלים קוונטיים בכלל: הוא נשען על deserialization עם access_unchecked ועל באג של register aliasing. גוגל תיקנה את המאמת (גרסה v2 של המאמר), והטענות המדעיות שלה לא נפגעו.

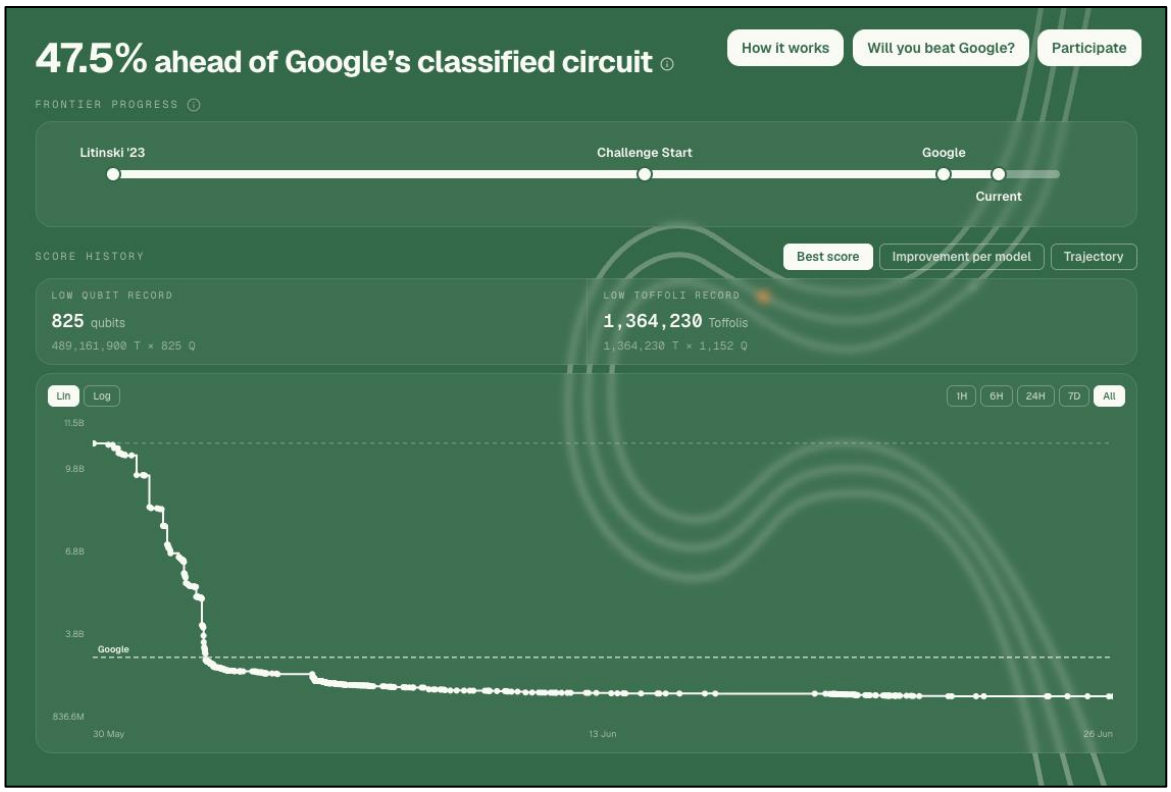
המסקנה כמו ש-Trail of Bits מנסחים: ZKP לא מבטל אמון, הוא מעביר אותו. במקום לסמוך על "האם המומחים מאמינים לזה", אתם עכשיו סומכים על המימוש של המאמת, על הקומפיילר ועל מערכת ההוכחה. זה משטח תקיפה של אבטחת יישומים, לא ערובה מתמטית.

מה הקהילה שחזרה מתוך ידע קיים

במקביל, חוקרים עצמאיים שחזרו את האופטימיזציה האמיתית מתוך ספרות פומבית, תוך מספר ימים. העבודה של André Schrottenloher על חיבור נקודות, ו-preprint על היפוך מודולרי חסכוני בזיכרון של Luo ואחרים, נשענים על תוצאות בנות עשורים: שיתוף רגיסטרים של Proos-Zalka מ-2003, והשיטה של Kaliski להיפוך. ה"סוד" של גוגל התברר כניתן לשחזור מתוך ה-prior art. תחרות בסגנון Shor-at-home המשיכה לגזום את המעגל הלאה.

לכן כשאתם רואים "הקהילה כבר X אחוז מעבר לגוגל", שווה לזכור שהמספר מערבב הוכחה מזויפת עם קריפטאנליזה אמיתית, ושהאחוז זז ותלוי באיזו מטריקה בוחרים. תתייחסו למספרים המדויקים כלא-רשמיים.

ומה זה אומר בפועל? שום דבר כאן לא שובר את ה-TLS שלכם השנה. עדיין צריך מחשב קוונטי רלוונטי-קריפטוגרפית עם מספר קיוביטים גדול, והוא לא קיים. מה שהשתנה הוא בדיוק מה שטענו בפוסט המקורי: הערכות המשאבים ממשיכות לרדת, ועכשיו אנחנו גם יודעים שהן קלות לשחזור וקשות לשמירה בסוד. תכננו לפי המגמה, לא לפי המספר של היום:



[מקור: ecdsa.fail]

סיכום

לפני שנה, "אל תיכנסו לפאניקה" הייתה העצה הנכונה: התקנים לא היו מוכנים והחישוב לא היה שם. זה השתנה. התקנים נחתו באוגוסט 2024, מחסום הקיוביטים נפל פעמיים ב-18 חודשים, והחלון לתכנון בדחיפות נמוכה נסגר. תתחילו את האינוונטר הקריפטוגרפי. תעדפו את ML-KEM להחלפת מפתחות. שימו מעבר חתימות על מפת הדרכים. ואם חוויתם אירוע עם הוצאת מידע מוצפן, הכניסו את הקוונטים למודל הסיכון ארוך-הטווח שלכם. היריבים שלכם כבר עושים את זה.

על המחבר

גיא ברנהרט-מגן הוא סמנכ"ל הטכנולוגיות של Profero וחוקר אבטחה עם רקע במתמטיקה, קריפטוגרפיה ובינה מלאכותית. הוא כותב על הנקודות שבהן קריפטוגרפיה, תגובה לאירועים ומחשוב קוונטי נפגשים. פרופרו היא חברה המתמחה בסיוע לחברות בהכנה לאירועי סייבר ותגובה לאירועים. ניתן לצור קשר דרך guy@profero.io.

מקורות מידע

- Babbush & Neven, Google Research - Safeguarding cryptocurrency by disclosing quantum vulnerabilities responsibly (https://research.google/blog/safeguarding-cryptocurrency-by-disclosing-quantum-vulnerabilities-responsibly/)
- Cain et al. - Shor's algorithm is possible with as few as 10,000 reconfigurable atomic qubits, arXiv:2603.28627 (https://arxiv.org/abs/2603.28627)
- NIST Post-Quantum Cryptography - FIPS 203 / 204 / 205 (https://csrc.nist.gov/projects/post-quantum-cryptography)
- Babbush et al. - Securing Elliptic Curve Cryptocurrencies against Quantum Vulnerabilities, arXiv:2603.28846 (https://arxiv.org/abs/2603.28846)
- Trail of Bits - We beat Google's zero-knowledge proof of quantum cryptanalysis (https://blog.trailofbits.com/2026/04/17/we-beat-googles-zero-knowledge-proof-of-quantum-cryptanalysis/)
- Schrottenloher - Optimized Point Addition Circuits for Elliptic Curve Discrete Logarithms, arXiv:2606.02235 (https://arxiv.org/abs/2606.02235)
- Luo et al. - Space-Efficient Quantum Algorithm for Elliptic Curve Discrete Logarithms, arXiv:2604.02311 (https://arxiv.org/abs/2604.02311)
- Proos & Zalka - Shor's discrete logarithm quantum algorithm for elliptic curves, arXiv:quant-ph/0301141 (https://arxiv.org/abs/quant-ph/0301141)
- Roetteler et al. - Quantum resource estimates for computing elliptic curve discrete logarithms (Kaliski inversion), arXiv:1706.06752 (https://arxiv.org/abs/1706.06752)
- Shor-at-home challenge - ecdsa.fail (https://ecdsa.fail)
- אלגוריתם Shor - ויקיפדיה (https://he.wikipedia.org/wiki/)
- Harvest Now, Decrypt Later - ויקיפדיה (https://en.wikipedia.org/wiki/Harvest_now,_decrypt_later)
- הפוסט המקורי Understanding Quantum Cryptography: Separating Fact from Fiction (Profero) (https://profero.io/blog/understanding-quantum-cryptography-separating-fact-from-fiction/)
- עדכון עמדה על PQC Filippo Valsorda (https://words.filippo.io/crqc-timeline/)
- הדיון ב-X (@P3b7_) (https://x.com/P3b7_/status/2066540230442692805)

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-187 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב: למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה רבות כדי להביא לכם את הגליון.

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

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

www.DigitalWhisper.co.il

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

הגליון הבא בתקווה ביום האחרון של חודש יולי!

אפיק קסטיאל, ספיר פדרובסקי

30.06.2026