חייכו, אכלתם אותה - איך התקנתי Ransomware על המצלמה הדיגיטלית שלכם

מאת אייל איטקין

מבוא

מצלמות. הן מלוות אותנו לכל אירוע חשוב בחיינו, אנחנו לוקחים אותן איתנו לחופשות, ושומרים עליהן בתיקים מרופדים כדי להגן עליהן במהלך הנסיעה. מצלמות הן יותר מסתם כלי עבודה או צעצוע, אנחנו מפקידים בידיהן את הזכרונות שלנו, ועל כן הן חשובות מאוד עבורנו.

מאמר זה יתאר את המסע אליו יצאנו בכדי לבדוק האם תוקפים יכולים לפגוע בנו בדיוק בנקודה הרגישה הזו. שאלת המחקר שלנו הייתה: "האם תוקפים יכולים להשתלט על המצלמות שלנו, אלו שאמונות על שמירת הזכרונות שלנו, ולהדביק אותן ב-Ransomware?". והתשובה היא: **כן**.

קצת רקע: מצלמות דיגיטליות הן כבר לא אותן מצלמות פילם גדולות ומסורבלות שהיו בשימוש של ההורים שלנו. מצלמות כיום הן מוצרי embedded דיגיטליים אשר מתחברים למחשב באמצעות USB, והדגמים החדישים אפילו תומכים בחיבור WiFi אל-חוטי. אולם, חיבורי ה-USB וה-WiFi אשר משמשים אותנו בכדי לייבא את התמונות מהמצלמה אל המחשב, או אל הפלאפון הנייד, גם חושפים את המצלמה אל העולם החיצון.

המחקר שלנו מראה כיצד תוקף הנמצא בטווח קצר מהמצלמה (WiFi), או תוקף שכבר השתלט על המחשב ונגיש אל המצלמה דרך USB, יוכל להתפשט הלאה אל המצלמה בכדי להדביק אותה ברושעה (Malware). רק דמיינו כיצד תגיבו אם תוקף ידביק ב-Ransomware גם את המחשב שלהם **וגם** את המצלמה, כך שכל התמונות שלכם יוחזקו כבנות ערובה עד שלא תשלמו את הכופר.

כטעימה לקראת ההמשך, להלן לינק לסרטון ההדגמה של התקיפה הנ"ל:

https://www.youtube.com/watch?v=75fVog7MKgg

Picture Transfer Protocol (PTP)

היות ומצלמות דיגיטליות כבר מזמן לא עושות שימוש בפילם, איגוד תעשיות הצילום הבינלאומי (International Imaging Industry Association) הגדיר פרוטוקול מתוקנן להעברת תמונות בצורה דיגיטלית מהמצלמה אל המחשב. פרוטוקול זה נקרא "פרוטוקול העברת תמונות" או Picture Transfer Protocol (PTP). למרות שהחל כפרוטוקול להעברת תמונות, הפרוטוקול תומך כיום בעשרות פקודות שונות הכוללות כל דבר החל משליחת פקודה לצילום חי ללא מגע יד אדם במצלמה, ועד לעדכון תוכנה למצלמה.

למרות שרוב האנשים עדיין משתמשים בכבל ה-USB בכדי לחבר את המצלמה שלהם אל המחשב, דגמים חדשים של מצלמות תומכים כיום גם בחיבור WiFi. המשמעות היא שמה שהחל כפרוטוקול PTP/USB, שודרג עם השנים כך שיתמוך גם ב-PTP/IP. שינוי זה הפך את המצלמה לנגישה לכל מי שנמצא בקרבה יחסית אל המצלמה, ללא הצורך לגישה פיזית באמצעות USB.

במחקר קודם בשם "Paparazzi over IP" (פפארצי מעל IP), אשר הוצג בכנס HITB 2013, חוקר בשם דניאל מנדה (Daniel Mende) הציג את כל תקיפות הרשת השונות על הפרוטוקולים שנתמכו באותה התקופה במצלמות דיגיטליות מסדרת ה-EOS של חברת Canon. במחקר זה, דניאל דן גם בפרוטוקול ה-PTP/IP והראה כי כל שתוקף צריך לעשות בכדי לתקשר עם המצלמה זה להסניף הודעת תקשורת יחידה ולהעתיק ממנה מזהה ייחודי (GUID). מזהה זה נוצר בעת החיבור הראשוני בין המחשב והמצלמה, ומאוחר יותר נשלח מחדש בכל הודעה בכדי ששני הצדדים ידעו שהם מתקשרים זה עם זה. היות ופרוטוקול ה-PTP תומך במגוון רחב של פקודות, והוא אינו כולל אימות חזק (מעבר לצימוד המזהה הייחודי) או צופן כלשהו, דניאל הדגים איך הוא ניצל לרעה את הפקודות הקיימות בפרוטוקול בכדי לרגל אחר בעלי המצלמה.

במחקר זה אנחנו נשאף לקחת את המחקר של דניאל צעד נוסף קדימה. אנחנו נרצה למצוא בעיות מימוש בפרוטוקול במטרה לנצל אותן ולהגיע עד לכדי הרצת קוד מלאה על המצלמה. השתלטות שכזו על המצלמה תאפשר לנו לעשות ככל העולה על רוחנו, ללא ההגבלות בנוגע לפונקציונאליות המקורית בה תומך הפרוטוקול. הדבקת המצלמה ב-Ransomware היא רק אופציה אחת מני רבות, אך לדעתנו היא האופציה שמפחידה את המשתמשים יותר מכל אופציה אחרת ולכן בחרנו לממש אותה במהלך מחקר זה.

מנקודת מבטו של תוקף, שכבת ה-PTP היא מטרה נהדרת:

- מדובר בפרוטוקול ללא אימות או הצפנה, אשר תומך בעשרות פקודות שונות ומורכבות.
- חולשה בשכבת ה-PTP תהיה נגישה דרך USB ודרך WiFi.
- הנגישות דרך ה-WiFi הופכת את החולשה לנגישה "מרחוק", או לפחות לא דורשת נגיעה פיזית.

הצגת המטרה - Canon EOS 80D

המדגים שבחרנו עבור המחקר הוא Canon's EOS 80D:

[מצלמת EOS 80D של חברת Canon]

הסיבות שהובילו אותנו לבחור במצלמה הזו היו:

- Canon היא היצרנית הגדולה ביותר של מצלמות דיגיטליות, והיא שולטת ביותר מ-50% משוק מצלמות ה-DSLR.
- דגם ה-EOS 80D תומך גם ב-USB וגם ב-WiFi.
- מסתבר של-Canon יש קהילת מפתחים גדולה אשר מפתחת התאמות למצלמה, ונקראת Magic Lantern.

Magic Lantern (או בקצרה "ML") היא מעין גרסה פתוחה של תוכנת המצלמה, אשר מתקנת באגים ומוסיפה תכולות חדשות על גבי התוכנה הקיימת של המצלמות של חברת Canon. מכאן שקהילת ML כבר חקרה חלקים רבים של תוכנת (וחומרת) המצלמה, וניתן למצוא בפורומים ובוויקי של הקהילה תיעוד מקיף ומועיל בנוגע לממשקים שונים של תוכנת המצלמה.

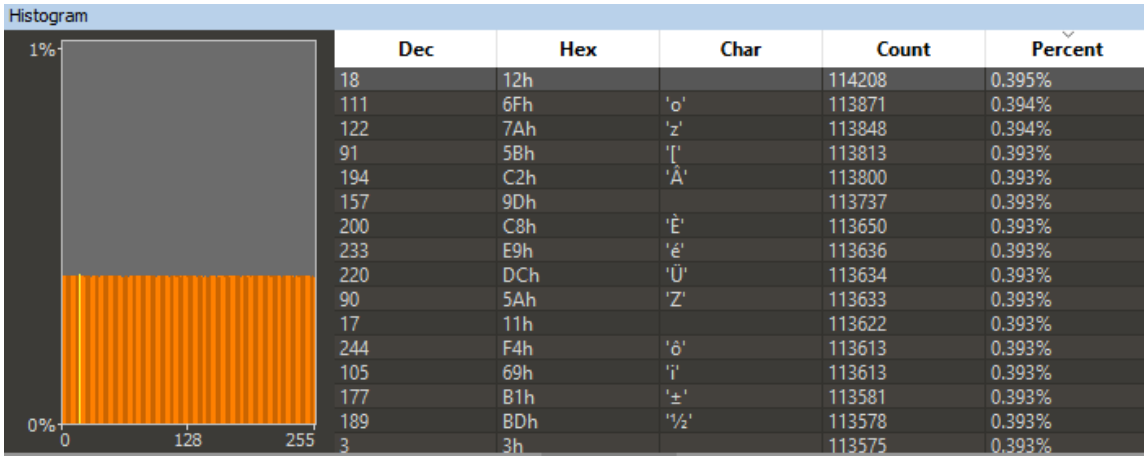
דוגמא לתיעוד של ML על פונקציית מערכת ההפעלה האחראית ליצירת Task חדש:

```
/** Create a new user level task.
 *
 * The arguments are not really known yet.
 */
extern struct task *
task_create(
        const char *            name,
        uint32_t                priority,
        uint32_t                stack_size,
        void *                  entry,
        void *                  arg
);
```

תוקפים הם בהוויתם ממקסמי רווחים, שזה ביטוי יפה ל"עצלנים, שחותרים לסיים מהר את הפרויקט". הרבה יותר קל להתחיל (ולסיים) מחקר כשאנחנו מגיעים אליו עם תיעוד מקיף אודות המוצר, ולא צריכים להתחיל הכל מאפס. במובן הזה, התיעוד הקיים של קהילת ML עזר לנו מאוד בתחילת המחקר.

השגת התוכנה (Firmware)

פעמים רבות זהו השלב המורכב ביותר בפרויקטי embedded. השלב הראשון הוא לבדוק האם יש עותק פומבי נגיש של עדכון תוכנה באתר של היצרן. כצפוי, די מהר מצאנו קובץ שכזה אחרי חיפוש קצת ב-Google. לצערנו, מיד לאחר שהורדנו אותו ופתחנו אותו בתוכנת עריכה בינארית, ציפתה לנו הפתעה:



Dec	Hex	Char	Count	Percent
18	12h		114208	0.395%
111	6Fh	'o'	113871	0.394%
122	7Ah	'z'	113848	0.394%
91	5Bh	'['	113813	0.393%
194	C2h	'Â'	113800	0.393%
157	9Dh		113737	0.393%
200	C8h	'È'	113650	0.393%
233	E9h	'é'	113636	0.393%
220	DCh	'Ü'	113634	0.393%
90	5Ah	'Z'	113633	0.393%
17	11h		113622	0.393%
244	F4h	'ô'	113613	0.393%
105	69h	'i'	113613	0.393%
177	B1h	'±'	113581	0.393%
189	BDh	'½'	113578	0.393%
3	3h		113575	0.393%

[התפלגות הבתים של קובץ עדכון התוכנה]

כפי שניתן לראות, התפלגות הבתים של הקובץ היא אחידה - ישנה כמות שווה של מופעים עבור כל בית. התפלגות שכזו מעידה לרוב כי הקובץ דחוס או מוצפן, כאשר אם מדובר בדחיסה הרי שזו דחיסה טובה למדי.

גם אחרי סריקה מהירה של תוכן הקובץ לא הצלחתי למצוא מקטע שיכול להיות קוד עלייה ראשוני (bootloader) שעשוי לעזור לנו לפתוח את הדחיסה (כפי שאכן קרה במחקר הפקס שלנו, אותו הצגנו בשנה שעברה). השלב הבא היה לנסות לפתוח את הדחיסה של הקובץ באמצעות מגוון תוכנות כגון 7Zip או Binwalk, אבל כל הנסיונות שלנו העלו חרס. המסקנה המתבקשת היא שהקובץ מוצפן ולא דחוס.

קבצי עדכון מוצפנים הם נדירים יחסית, אבל אחרי התייעצות קצרה עם מספר חברים הבנתי שלרוב אפשר למצוא את המפתח בחיפוש קצר ב-Google. אז ניסינו לחפש ב-Google, ולא מצאנו כלום ☹

מה שכן מצאנו, הוא ש-Magic Lantern היו צריכים גם הם להתמודד עם הבעיה הזו, ואכן מצאנו בוויקי שלהם את העמוד הבא, אשר מתאר את "הגנות העדכון" כפי שהן מופיעות בעדכונים השונים לאורך השנים. הדבר היחידי שמצאנו בעמוד הוא שהתוכנה מוצפנת באמצעות אלגוריתם הצופן AES, אבל לא מצאנו שום הסבר על המפתח. למרות ש-ML היא קהילת קוד פתוח, הם בכוונה לא משחררים את מפתח הפענוח לקבצי התוכנה. נראה שזהו עוד מבוי סתום.

השלב הבא היה לבדוק האם Magic Lantern סיימו לפתח את גרסת התוכנה לדגם המצלמה שלנו, מכיוון שהתוכנה שלהם כוללת תכולות דיבאג, ואולי הן יאפשרו לנו לקרוא את זכרון המצלמה. לצערנו, אפילו במועד כתיבת מסמך זה, עדיין לא קיימת גרסה נתמכת של התוכנה למצלמה שלנו. אבל, במהלך השיטוטים השונים שלנו בפורומים של ML, מצאנו משהו מעניין. נראה ש-ML פיתוח משהו שנקרא

Portable ROM Dumper (כלי להדלפת ה-ROM). זהו עדכון תוכנה ייעודי שהם פיתחו, אשר שופך את כל תוכן זכרון המצלמה אל כרטיס ה-SD.

להלן תמונה של המצלמה (EOS 80D) במהלך תהליך "עדכון התוכנה" להזלגת הקוד:

לא רק שהתהליך עבד בצורה חלקה, הפורום כולל גם הוראות לטעינה נכונה של הקובץ אל IDA ☺, עכשיו אנחנו יכולים בשעה טובה להתחיל לחפש חולשות במצלמה!

פריסת שכבת ה-PTP

מציאת שכבת ה-PTP במרחב הזכרון הייתה פשוטה יחסית, הודות לשילוב של שני מרכיבים שונים:

1. פרוטוקול ה-PTP מבוסס פקודות, ולכל פקודה יש מזהה מספרי יחודי.
2. התוכנה כוללת המון מחרוזות מועילות, אשר מקלות בצורה משמעותית על תהליך הפריסה.

```
ROM:FE56503A 82 46          MOV             R10, R0
ROM:FE56503C 8E B0          SUB             SP, SP, #0x38
ROM:FE56503E 0C 46          MOV             R4, R1
ROM:FE565040 03 21          MOVS            R1, #3
ROM:FE565042 1F 46          MOV             R7, R3
ROM:FE565044 4E A2          ADR             R2, aPtpOpensession ; "PTP_OpenSession"
ROM:FE565046 28 68          LDR             R0, [R5]
ROM:FE565048 DD E9 16 86    LDRD.W          R8, R6, [SP,#0x58+transaction]
ROM:FE56504C 40 69          LDR             R0, [R0,#0x14]
ROM:FE56504E C0 B2          UXTB            R0, R0
ROM:FE565050 97 F6 E0 E8    BLX             EI_dbg_printf_FE3FC214
```

[מחרוזת לדוגמא מתוך קוד ה-PTP]

הליכה אחורה מהפונקציה האחראית לטיפול בפקודת פתיחת החיבור (Open Session) הובילה אותנו אל הפונקציה הראשית אשר רושמת את **כל** פונקציות הטיפול השונות בפקודות הפרוטוקול. בדיקה מהירה של קבועי הפרוטוקול מול המחרוזות השונות בקוד, העידה שאכן יש לנו כעת מיפוי שלם בין כל פקודה לפונקציה שאחראית על הטיפול בה.

למעשה, רק בשלב הזה הבנו כמה הבחירה שלנו לחפש חולשות בשכבת ה-PTP הייתה מוצלחת. הפונקציה הראשית מגדירה 148 פונקציות **שונות**, דבר המעיד כי היצרן תומך בעשרות פקודות יעודיות שונות. עם תמיכה בכמעט 150 פקודות שונות, הסיכוי שנמצא חולשה קריטית בלפחות אחת מהפקודות הוא גבוה מאוד.

ממשק הטיפול בפקודות ה-PTP

כל פונקציה המטפלת בפקודת PTP יורשת מאותו הממשק. ממשק זה עושה שימוש נרחב באובייקט ה-ptp_context, אשר מתועד בחלקו הודות ל-ML. האיור הבא מראש שימוש לדוגמא בפונקציונאליות שמציע האובייקט:

```
EI_PTP_FillUpResponse(&msg, &opcode);
v15 = 0;
v14 = 0;
input_size = (ctx->get_data_size_ptr)(ctx->handle);
if ( input_size == 144 )
{
  v12 = (ctx->recv_data_ptr)(ctx->handle, local_buffer, 144);
  if ( v12 << 31 )
    EI_dbg_printf_FE3FC214(global_ptp_memory_context->some_mem_id, 6, "PTPReceiveDataOneTime (%#x)", v12);
  sub_FE38864A(local_buffer, &v15);
  sub_FE562286(0xC1FDu, v15, 0xE8u);
}
else
{
  EI_dbg_printf_FE3FC214(global_ptp_memory_context->some_mem_id, 6, "PTP_SendTimeSyncInfo: Size = %d", input_size);
}
if ( EI_PTP_SendResponse(ctx, &msg) << 31 )
  EI_dbg_printf_FE3FC214(global_ptp_memory_context->some_mem_id, 6, "PTP_SendTimeSyncInfo USB Send Error");
```

[להלן קוד לדוגמא (לאחר דה-קימפול) אשר אחראי על טיפול בפקודת PTP]

כפי שניתן לראות, האובייקט מכיל מצביעים שונים לפונקציות, אשר משמשים למטרות הבאות:

- בירור גודל ההודעה הנכנסת, טרם הטיפול בה.
- קריאת תוכן ההודעה הנכנסת.
- שליחת התשובה לאחר הטיפול בהודעה.

מסתבר כי מרבית הפקודות הן פשוטות יחסית. הן מקבלות רק מספר פרמטרים כערכים מספריים, היות והפרוטוקול תומך בקבלת עד 5 פרמטרים שונים לכל פקודה. סריקה מהירה של כל הפקודות העלתה כי מתוך 148 פקודות נתמכות, רק 38 פקודות מקבלות בנוסף לפרמטרים גם חוצץ מידע. זה יחסית הגיוני כשחושבים על זה, היות ורוב הפקודות בפרוטוקול נועדו כדי לתשאל מידע מהמצלמה, ולכן אין הגיון בכך שאנחנו נצטרך לשלוח לה כמויות גדולות של מידע.

מנקודת מבטו של תוקף, יש לנו שליטה מלאה בכל התוכן של חוצץ המידע שנשלח בתוך ההודעה שלנו אל המצלמה. לכן, בחרנו להתמקד תחילה רק ב-38 הפקודות אשר מטפלות במידע שמגיע מאיתנו. למזלנו, קוד הטיפול בהודעות הוא קוד C פשוט למדי, כך שקל לבחון אותו ולחפש בו חולשות. ואכן, מהר למדי מצאנו את החולשה הראשונה.

CVE-2019-5994 - Buffer Overflow in SendObjectInfo - 0x100C

- **שם פקודת PTP:** SendObjectInfo
- **מזהה פקודת PTP:** 0x100C

פנימית, הפרוטוקול מתייחס לתמונות ולקבצים בתור "אובייקטים", כאשר הפקודה הנ"ל מאפשרת למשתמש לעדכן את המידע (metadata) של אובייקט ספציפי. קוד הטיפול בהודעה כולל חולשת דריסת זכרון (Buffer Overflow) כאשר מועתק מה שאמור להיות שם הקובץ ב-Unicode. האיור הבא מציג גרסה מפושטת מעט של הקוד הפגיע:

```
/* Query the size of the incoming message */
input_size = ctx->get_data_size_ptr(ctx->handle);
if ( !input_size )
    return 0;
/* Allocate memory for the incoming message */
pMessage = PTP_AllocateMemory(input_size);
if ( !pMessage)
{
    ... // error handling
    return 1;
}
/* Receive all of the message */
if (ctx->recv_data_ptr(ctx->handle, pMessage, input_size) < 0)
{
    ... // error handling
    return 0;
}
ObjectFormat = pMessage[4] | (pMessage[5] << 8);
if (ObjectFormat != Script && ObjectFormat != UNDOCUMENTED_writable)
{
    goto label_error_resp;
}
pMemoryContext = global_ptp_memory_context;
global_ptp_memory_context->some_handle = some_handle;
/* Read a 1-byte length field from the packet */
filename_len_1_byte = pMessage[52];
/* EI-DBG: Controlled memcpy into a buffer of size 0x40 */
memcpy(pMemoryContext->unicode_filename, pMessage + 53, 2 * filename_len_1_byte);
...
/* EI-DBG: Free-Where primitive - using a pointer we can overwrite using our overflow */
if ( pMemoryContext->pKeywordsStringUnicode)
{
    PTP_FreeMemory(pMemoryContext->pKeywordsStringUnicode);
    global_ptp_memory_context->pKeywordsStringUnicode = 0;
}
```

[קטע הקוד הפגיע מתוך פונקציית הטיפול בפקודת SendObjectInfo]

דריסת הזכרון עצמה נמצאת בתוך שדות שונים של אובייקט גלובאלי כלשהו. מבלי אבחון מעמיק לשדות השונים של האובייקט, מצאנו כי ההשלכה הישירה של הדריסה היא השגת פרימיטיב מסוג "שחרור נשלט" (Free Where) שממוקם מיד לאחר ההעתקה הפגיעה. הדריסה שלנו יכולה לשלוט בתוכנו של השדה pKeywordsStringUnicode, ומאוחר יותר שדה זה משוחרר באמצעות קריאה לפונקציה free().

נראה כמו התחלה טובה למחקר, אבל בחרנו להמשיך לחפש חולשות נוספות שבשאיפה יהיו קלות יותר להשמשה. שווה להזכיר כי בשלב הזה יש לנו רק מצלמה ואין לנו דיבאגר, כך שאנחנו אפילו לא יודעים עדיין איך עובד מנגנון ה-heap של המצלמה.

CVE-2019-5998 - Buffer Overflow in NotifyBtStatus - 0x91F9

- **שם פקודת PTP:** NotifyBtStatus
- **מזהה פקודת PTP:** 0x91F9

למרות שהמצלמה שלנו כלל לא תומכת ב-Bluetooth (וכן, בדקנו את זה כמה וכמה פעמים ברגע שראינו את הפקודה הזו), נראה כי חלק מהפקודות הקשורות ל-Bluetooth הושארו מאחור ולא הוסרו מהקוד. לימדו אותי לא לשאול שאלות כשיש הפתעות לטובה בפרויקטים, כך שלא הופתעתי גם לגלות דריסת מחסנית קלאסית בפונקציה הזו ☺

```
input_size = ctx->get_data_size_ptr(ctx->handle);
/* Preparing a buffer of size 0x120 */
bzero(local_buffer, 0x120);
PTP_FillUpResponse(&msg, args);
/* EI-DBG: Using input_size, with no checks against the buffer's size */
if (ctx->recv_data_ptr(ctx->handle, local_buffer, input_size) >= 0)
{
    COM_NotifyBtStatus(input_size, param1, local_buffer);
}
some_fptr = global_ptp_memory_context->adapterStatusCallback;
if ( some_fptr )
{
    some_fptr(global_ptp_memory_context->fptr_108, 12, param1);
}
result = PTP_SendResponse(ctx, &msg) < 0;
if ( result )
{
    ... // error handling
    result = 1;
}
return result;
```

[קטע הקוד הפגיע מתוך פונקציית הטיפול בפקודת NotifyBtStatus]

הפעם, השמשת החולשה הנ"ל נראית קלה למדי, מה שהופך אותה למועמדת העיקרית שלנו עבור תרחיש התקיפה. למרות שנהוג לעצור בשלב הזה, בחרנו להמשיך בסריקת יתר הפקודות מכיוון שכבר היינו יחסית קרובים לסוף הרשימה.

CVE-2019-5999- Buffer Overflow in BLERequest - 0x914C

- **שם פקודת PTP:** BLERequest
- **מזהה פקודת PTP:** 0x914C

שוב פקודת Bluetooth. נראה שפקודות אלו פגיעות יותר בהשוואה לפקודות אחרות, מה שעלול להעיד כי צוות הפיתוח הנ"ל היה פחות מנוסה מיתר הצוותים שעבדו על המוצר. הפעם, כדי להשלים סדרה, מצאנו דריסת זכרון על ה-heap, כפי שניתן לראות בקטע הקוד הבא:

```
...
input_size = ctx->get_data_size_ptr(ctx->handle);
if ( !input_size )
{
    ... // error handling
    return 0;
}
/* Allocating a fixed size buffer of 24 bytes */
pBuffer = PTP_AllocateMemory(global_ptp_memory_context, 24);
/* EI-DBG: receiving data into &buffer[4], without size checks */
ctx->recv_data_ptr(ctx->handle, pBuffer + 4, input_size);
((unsigned int *)pBuffer)[0] = param2;
res = COM_BLERequest(7, pBuffer, 24);
PTP_FreeMemory(global_ptp_memory_context, pBuffer);
...
```

קטע הקוד הפגיע מתוך פונקציית הטיפול בפקודת BLERequest.

עד כה מצאנו 3 חולשות דומות:

- דריסת זכרון על משתנים גלובאליים.
- דריסת זכרון על המחסנית.
- דריסת זכרון על ה-heap.

כפי שהזכרנו קודם לכן, אנחנו ננסה להשמיש את החולשה על המחסנית, בתקווה שהיא תהיה הקלה ביותר לניצול.

הגיע הזמן להריץ קוד

הדבר הראשון שעשינו הוא לחבר את המצלמה אל המחשב שלנו באמצעות USB. עוד קודם לכן חיברנו את המצלמה למחשב באמצעות כבל USB ותקשרנו איתה באמצעות תוכנת "EOS Utility" של היצרן, כך שזה הרגיש לנו טבעי להתחיל לנסות לנצל את החולשה מעל USB. בחיפושינו אחרי ספריית קוד פתוח בפייתון שתתמוך בפרוטוקול ה-PTP, מצאנו את ptpy. לצערי, כמו רוב פרויקטי הקוד הפתוח, הספריה לא עבדה בנסיון ראשון. את הבאגים תיקנו מהר יחסית ובסופו של יום הספריה כן חסכה לנו זמן עבודה יקר.

לפני שנכתוב השמשה מלאה שתריץ קוד על המצלמה, בחרנו להתחיל בלכתוב הוכחת היתכנות קצרה (PoC) לכל אחת מהחולשות שמצאנו.

כלל חשוב: עד שלא כתבתם קוד שמטריג חולשה, אתם לא יכולים לדעת שהחולשה אכן קיימת. תמיד יכול להיות איזשהו מרכיב נסתר שיחסום את החולשה.

במקרה שלנו, נקווה לראות בכל מקרה כי המצלמה תקרוס. התמונה הבאה מראה כיצד נראית המצלמה כשהיא קורסת, מה שמתואר על ידי היצרן לרוב בתור "Err 70":

[מסך הקריסה שקיבלנו כאשר בדקנו את הוכחות ההיתכנות לחולשות שמצאנו]

עכשיו כשאנחנו בטוחים שכל החולשות שלנו אכן עובדות, הגיע הזמן להפשיל שרוולים ולהתחיל לכתוב את התקיפה המלאה על המצלמה.

סיכום קצר לפני שנתחיל: למצלמה שלנו **אין** דיבאגר, וגם אין לנו גרסת ML עבור המצלמה. המצלמה עדיין שלמה, ולא פתחנו את הקופסא שלה (חבל להרוס מצלמה יקרה) כך שגם אין לנו שום יכולות דיבוג חומרתיות. בנוסף אנחנו גם לא יודעים שום דבר על מרחב הזכרון של המצלמה, למעט הכתובות שראינו דרך IDA. בשורה התחתונה, אנחנו מחוברים אל קופסא שחורה (המצלמה שלנו) באמצעות כבל USB, אנחנו עיוורים לחלוטין, ועדיין אנחנו מנסים להשמיש דריסה על המחסנית.

בואו נתחיל.

התוכנית שלנו היא לקרוא לפונקציה Sleep() שתשמש בתור breakpoint, כך שבתקווה נוכל לבדוק אם המצלמה קרסה אחרי מספר מסויים של שניות, או שהיא קרסה ישר. בדיקה שכזו תוכל להעיד האם תהליך הניצול של החולשה הצליח להגיע בהצלחה אל הקריאה ל-Sleep() או שאולי הייתה לנו טעות וקרסנו עוד לפני נקודת הבדיקה. התוכנית נראתה מאוד מוצלח על הנייר, אבל נראה שלמצלמה היו תוכניות אחרות בשבילנו. ברוב הפעמים, נראה שה-Task הפגיע מת מבלי להקריס את המצלמה, מה שבפועל גרם למצלמה להיתקע ולא להגיב. מיותר לציין כי אין לנו שום דרך להבדיל בין היעדר תגובה ובין שינה ואחריה היעדר תגובה, מה שהופך את שיטת הדיבוג שלנו לחסרת תועלת.

במקור, המטרה הייתה לבדוק עד לאן הגענו בתהליך הניצול. זאת כדי שנוכל לדבג ולחפש טעויות, כאשר אנחנו יוצאים מנקודת הנחה שהסיכוי שהכל יעבוד בנסיון הראשון הוא נמוך למדי. לשם כך אנחנו צריכים דרך כלשהי לדעת האם הגענו עד לנקודה מסויימת בריצה, וכפי שהבנתם אנחנו לא בררניים, אנחנו נשתמש בכל שיטה שתעבוד.

אחרי לא מעט ניחושים, הצלחנו לבסוף למצוא כתובות שתמיד תגרום למצלמה לקרוס עם Err 70. למקרה שתהיתם, הכתובת הייתה: 0x00022B60. כן, גם הכתובות 0x0 ו-0xFFFFFFFF לא עבדו לנו, כך שהיינו די שמחים כשמצאנו כתובת שאשכרה עובדת בסוף.

השיטה שלנו מעכשיו תהיה כמעט הפוכה מהשיטה הקודמת. קריסה תעיד כי הגענו עד ל-breakpoint שלנו, ואילו "כלום" (כלומר מצלמה שנתקעה ללא תגובה) יעיד כי לא הצלחנו להגיע עד ל-breakpoint. בשיטה הזו הצלחנו בהדרגה לבנות את הניצול שלנו, עד שלבסוף הצלחנו להריץ קטע אסמבלי לשליטתנו - הצלחנו להריץ קוד ☺

טעינת דיבאגר - Scout

דיבאגר ברירת המחדל שלי במחקרי embedded הוא Scout. זהו דיבאגר מבוסס הוראות (או פיקודים) שפיתחתי עבור מחקר הפקס, ואשר הוכיח את עצמו גם במהלך המחקר הזה. אבל, לרוב אני טוען את הדיבאגר אל היעד באמצעות מודול טעינת TCP, אשר דורש קישוריות רשת אל המטרה. זהו למעשה קטע הקוד הראשוני (לרוב יקרא shellcode) שישלח כחלק מהתקיפה. בעוד שאנחנו יכולים להשתמש במודול טעינה מקובץ שימצא על כרטיס ה-SD, מאחור יותר בכל מקרה נצטרך קישוריות רשת עבור הדיבאגר עצמו. לכן, מוטב שכבר נצליח להתגבר על בעיית הרשת כעת, במקום לדחות אותה לאחר כך.

אחרי ששיחקתי מעט עם ההגדרות השונות של המצלמה, הבנתי לצערי כי אי אפשר להשתמש ב-WiFi בזמן שכבל ה-USB מחובר. ההסבר לכך הוא ששניהם נועדו כדי לשמש שכבת תעבורה עבור ה-PTP, ולכן אין תמיכה לכך ששניהם יהיו פעילים בו זמנית. נראה שהגיע הזמן להתקדם ולעבור מ-USB ל-WiFi.

קשה לי לומר שהמעבר אל חיבור ה-WiFi היה פשוט. השילוב של ממשק WiFi רעוע בצד המצלמה, יחד עם תכולות חסרות מצד ptpy לא הקלו עלינו, אבל בסופו של יום הצלחנו להגיע למצב שבו הסקריפט שלנו שולח בדיוק את אותה התקיפה, והפעם מעל WiFi. לרוע מזלנו, הסקריפט שלנו הפסיק לעבוד. בדיקה מעמיקה העלתה שהמצלמה קורסת עוד לפני שהיא חוזרת מהפונקציה הפגיעה, ולכן החולשה שניצלנו כבר לא נצילה יותר ☹

אין לנו באמת מושג למה המצלמה קורסת הפעם, אבל הניחוש שלנו הוא שלהודיע למצלמה משהו בנוגע לחיבור Bluetooth בזמן שהיא גם ככה מחוברת באל-חוט דרך WiFi פשוט היה הקש ששבר את גב הגמל. זה שהיא לא באמת תומכת ב-Bluetooth בטוח לא עזר.

הגיע הזמן לחזור ללוח השרטוטים. אנחנו יכולים לנסות להשמיש אחת מהחולשות האחרות, אבל אחת מהן גם היא קשורה ל-Bluetooth, כך שזה לא נראה מבטיח. לחלופין, החלטנו שעדיף לעבור פעם נוספת על רשימת פקודות ה-PTP, בתקווה למצוא חולשות נוספות שאולי פספסנו בסריקה הקודמת. הפעם המזל כן העיר לנו פנים, כי מצאנו עוד כמה חולשות.

CVE-2019-6000- Buffer Overflow in SendHostInfo - 0x91E4

- **שם פקודת PTP:** SendHostInfo
- **מזהה פקודת PTP:** 0x91E4

גם מבט חטוף על קטע הקוד הפגיע יסביר איך פספסנו את החולשה בפעם הראשונה שעברנו על הפונקציה:

```
input_size = ctx->get_data_size_ptr(ctx->handle);
/* Size check - expecting a fixed size message */
if ( input_size != 100 )
{
    dbg_printf(
      global_ptp_memory_context->some_mem_id,
      3,
      " PTP_SendHostInfo ReceiveSizeError [%x]",
      input_size);
    /* EI-DBG: No return? illegal packets will be logged, and that's it! */
}
ctx->recv_data_ptr(ctx->handle, local_msg_buffer, input_size);
COM_SendHostInfo(local_buffer);
if ( PTP_SendResponse(ctx, &msg) < 0 )
{
    dbg_printf(
      global_ptp_memory_context->some_mem_id,
      6,
      "PTP_SendHostInfo USB Send Error");
}
return result;
```

[קטע הקוד הפגיע מתוך פונקציית הטיפול בפקודת SendHostInfo]

הפעם נראה כי המפתחים זכרו לבדוק כי אורך ההודעה תואם לאורך המצופה, בדיוק 100 בתים. אולם, הם שכחו משהו די חשוב. הודעות לא חוקיות ירשמו ללוג, **אבל לא יופלו.** בדיקה מהירה במודל אימתה את המסקנה שלנו, פונקציית הלוג לא מקריסה את המצלמה, ולכן שום דבר לא יעצור אותנו מלשלוח הודעה ארוכה מדי ששוב תוביל לדריסת זכרון על המחסנית ☺

למרות שהחולשה הזו היא בדיוק מה שחיפשנו, גם הפעם בחרנו להמשיך לחפש חולשות נוספות. זאת ביחוד משום שיש סיכוי גבוה שנמצא חולשות נוספות כמו זו שבדיוק מצאנו.

CVE-2019-6001- Buffer Overflow in SetAdapterBatteryReport - 0x91FD

- **שם פקודת PTP:** SendAdapterBatteryReport
- **מזהה פקודת PTP:** 0x91FD

לא רק שמצאנו חולשה נוספת מאותו הסוג, גם מצאנו אותה בדיוק בפקודה האחרונה שברשימה, מה שהוביל אותנו לסיים בצורה מוצלחת.

```
input_size = ctx->get_data_size_ptr(ctx->handle);
/* Once again a fixed-size check */
if ( input_size != 28 )
{
    dbg_printf(
      global_ptp_memory_context->some_mem_id,
      3,
      " PTP_SetAdapterBatteryReport ReceiveSizeError [%x]",
      input_size);
    /* EI-DBG: Once again, there is a fall through after the log */
}
ctx->recv_data_ptr(ctx->handle, local_buffer, input_size);
COM_SetAdapterBatteryReport(&local_buffer[4]);
if ( PTP_SendResponse(ctx, &msg) < 0 )
{
    dbg_printf(
      global_ptp_memory_context->some_mem_id,
      6,
      "PTP_SetAdapterBatteryReport USB Send Error");
}
return result;
```

[קטע הקוד הפגיע מתוך פונקציית הטיפול בפקודת SendAdapterBatteryReport]

במקרה הזה, חוצץ המחסנית הוא קטן יחסית, כך שבחרנו לנצל את החולשה הקודמת שמצאנו במקום.

הערת צד: כשבדקנו את החולשה הנ"ל בסביבת WiFi מצאנו שגם היא מקריסה את המצלמה עוד בטרם החזרה מהפונקציה. את החולשה הזו הצלחנו להטריג רק מעל חיבור USB.

טעינת Scout - נסיון שני

מצויידים בחולשה החדשה שלנו, הצלחנו לסיים בהצלחה את כתיבת הניצול למצלמה, והצלחנו לטעון את Scout על המצלמה. כעת יש לנו דיבאגר רשתי, ונוכל להתחיל לקרוא זכרון מהמצלמה, דבר שיועיל מאוד בתהליך הפריסה.

אבל רגע, לא סיימנו כבר? המטרה שלנו הייתה להראות שאפשר להשתלט מרחוק על מצלמות באמצעות פרוטוקול ה-PTP, וזה בדיוק מה שהרגע עשינו. אבל השתלטות על המצלמה הייתה רק הצעד הראשון בתרחיש התקיפה שהצגנו, עכשיו הגיע הזמן לפתח Ransomware.

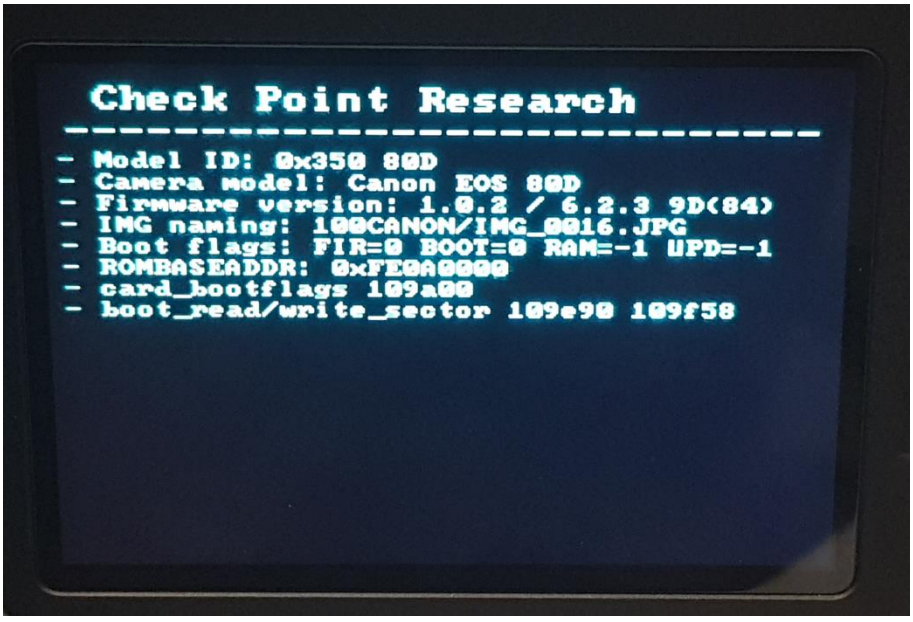
זמן לקצת צופן

כל Ransomware שמכבד את עצמו צריך פונקציות צופן בכדי להצפין את הקבצים / תמונות ששמורות על המצלמה. אם ניזכר, תהליך עדכון התוכנה הזכיר משהו על הצפנת AES. זאת יכולה להיות הזדמנות טובה לסיים את כל המטלות הפתוחות במכה אחת.

הפעם תהליך הפריסה היה הרבה יותר מוצלח ממה שציפינו. לא רק שמצאנו את כל פונקציות הצפנת ה-AES, גם מצאנו את **מפתחות הצופן** שאחראים על פענוח התוכנה ובדיקת החתימה שלה. ומכיוון ו-AES הוא צופן **סימטרי**, הרי שאותם המפתחות יכולים גם לשמש בכדי להצפין תוכנה משלנו ואפילו לייצר חתימה חוקית עבורה.

במקום לממש בעצמנו את כל הלוגיקה המורכבת של הצפנת התוכנה, נעזרנו שוב ב-Scout. מימשנו הרחבה שמדמה תהליך עדכון תוכנה, ושולחת בחזרה את החתימה הקריפטוגרפית שחושבה בחזרה אלינו. באמצעות ההוראה הנ"ל, יש לנו כעת את היכולת לייצר חתימה חוקית לכל קובץ שנרצה, בפרט לקובץ תוכנה עוין שנוכל לשלוח למצלמה בכדי להשתלט עליה גם ללא חולשות מימוש.

מכיוון והייתה לנו רק מצלמה אחת, השלב הזה היה מסוכן מעט. אנחנו רוצים לבדוק את קובץ העדכון העוין שיצרנו בעצמנו, אבל אנחנו לא רוצים להרוס את המצלמה ולהפוך אותה ללבנה (brick). למזלנו, התמונה הבאה מראה את תוצאות הרצת ה-ROM Dumper המתוקן שלנו, שנוצר לאחר ריתוך העדכון המקורי של ML:



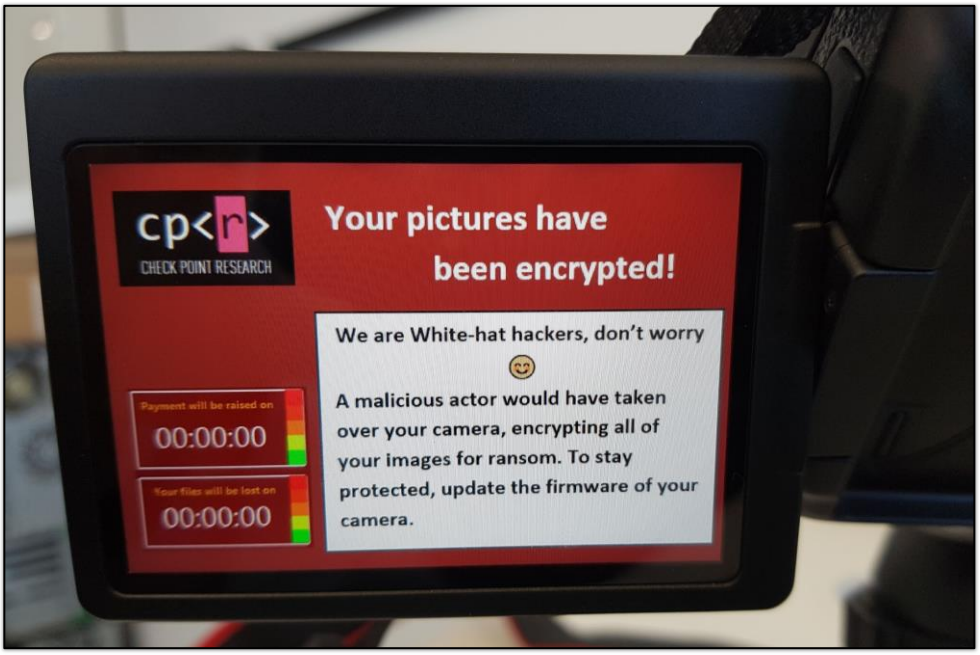
תוצאות הרצת "עדכון התוכנה" העוין שלנו, שכעת עושה שימוש בכותרת שלנו ☺

CVE-2019-5995 - עדכון תוכנה חרישי עוין

פרוטוקול ה-PTP כולל פקודת הרחבה שמומשה על ידי היצרן, ומאפשרת עדכון תוכנה מרחוק ש**לא** דורש מעורבות משתמש. המשמעות היא שגם אם כל חולשות המימוש שמצאנו יתוקנו, תוקף עדיין יוכל להשתלט על המצלמה באמצעות שליחת עדכון תוכנה עוין, בכפוף לכך שיוכל לייצר עדכון שכזה (כפי שאכן עשינו).

זמן לסיכום

אחרי שסיימנו להשתעשע עם תהליך עדכון התוכנה, חזרנו לסיים את תהליך כתיבת ה-Ransomware. בחרנו להצפין את התמונות באמצעות אותן פונקציות הצופן אשר משמשות את תהליך עדכון התוכנה, וזאת באמצעות קריאה ישירה לאותן הפונקציות כפי שהן ממומשות בתוכנת המצלמה. לאחר הצפנת כל התמונות על כרטיס ה-SD, הקוד שלנו יציג את הודעת הכופר על גבי מסך המצלמה:



להלן סרטון המציג את התקיפה המלאה על המצלמה:

https://www.youtube.com/watch?v=75fVog7MKgg

לו"ז תיקונים

- 31 למרץ 2019 - כל החולשות דווחו לחברת Canon
- 14 למאי 2019 - היצרן אישר את כל הממצאים ששלחנו לו
- החל משלב זה שתי החברות עבדו בשיתוף פעולה בכדי לתקן את החולשות
- 8 ליולי 2019 - אימתנו את התיקון של Canon ואישרנו אותו
- 6 לאוגוסט 2019 - Canon שחררו הודעה רשימת ללקוחות בקריאה שיתקינו את התוכנה המתוקנת
- 11 לאוגוסט 2019 - הצגה רשמית של המחקר מעל במת DEFCON 27

סיכום

במהלך המחקר שלנו מצאנו מספר חולשות מימוש (ואיפיון) קריטיות בפרוטוקול העברת התמונות (PTP), כפי שמומש על ידי חברת Canon. למרות שהמימוש שבחנו כולל מספר רב של פקודות ייעודיות ליצרן זה, הפרוטוקול עצמו מתוקנן, והוא משולב במרבית המצלמות הדיגיטליות הקיימות כיום. על סמך התוצאות שלנו, אנו מאמינים כי חולשות דומות עשויות להימצא במימושים נוספים של הפרוטוקול, ואף במצלמות של חברות נוספות.

המחקר שלנו מעיד כי כל מוצר "חכם", ובמקרה שלנו מצלמה דיגיטלית, הוא קורבן אפשרי לתקיפות. השילוב של מחיר גבוה, תוכן רגיש, ותפוצה רחבה בקרב צרכנים הופך את המצלמות הדיגיטליות למטרה יוקרתית עבור תוקפים.

הערה אחרונה בדבר הצפנת התוכנה. באמצעות השימוש בעדכון התוכנה העוין של Magic Lantern הצלחנו להדליף את תוכנת המצלמה, ומאוחר יותר עשינו שימוש בפונקציות עצמן של התוכנה בכדי לפתוח ולעקוף את שכבת האימות וההצפנה של עדכון התוכנה. זוהי דוגמא קלאסית לכך שהסתרה (obscurity) אינה בהכרח מספקת אבטחה (security), בייחוד כאשר שהזמן שלקח לנו לעקוף את שכבות ההגנה הנ"ל היה קצר למדי.

הפניות למידע נוסף

במקום להוסיף בכל מקום הפנייה ללינק חיצוני עם מידע נוסף, אנו מזמינים אתכם לעיין בבלוג המחקרי שפרסמנו באתר של קבוצת המחקר, והוא כולל את סרטון ההדגמה כמו גם את רשימת כל מקורות המידע בהם נעזרנו / אליהם אנחנו מפנים במהלך המאמר:

https://research.checkpoint.com/say-cheese-ransomware-ing-a-dslr-camera/