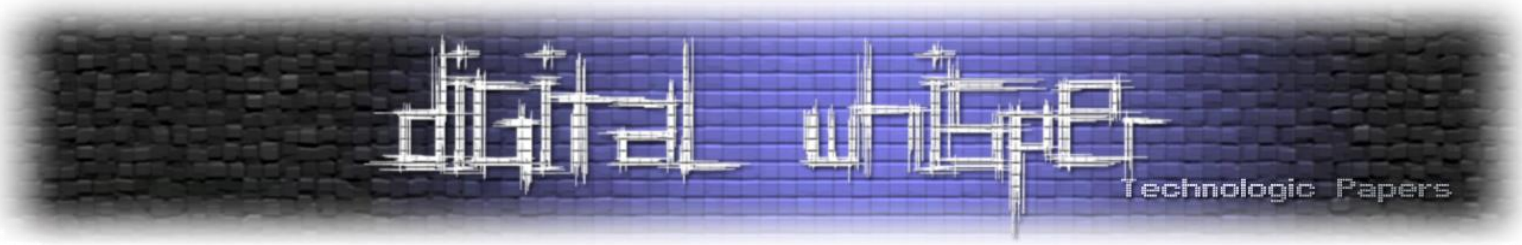


Digital Whisper

גליון 110, ספטמבר 2019

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	אריאל ז., אייל איטקין, שחף גרגיר ודביר גולן

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 העורכים

ברוכים הבאים לגליון ה-110 של DigitalWhisper, גליון ספטמבר!

החודש, מפאת חוסר זמן אנו נאלצים לשלוח אתכם לדרככם ללא דברי פתיחה, בתקווה שחודש הבא נחפור לכם מספיק כדי לפצות ☺

ואיך אפשר בלי - ברצוננו להגיד תודה לכל מי שניצל את הזמן שלו החודש בצורה איכותי ביותר וישב לכתוב לנו מאמרים שלא נראו כמותם מזה עידן ועידנים! תודה רבה ל**אריאל ז.**, תודה רבה ל**אייל איטקין**, תודה רבה ל**שחף גרגיר** ותודה רבה ל**דביר גולן**.

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

דבר העורכים	2
תוכן עניינים	3
Windows Kernel Fuzzing With Intel PT	4
חייכו, אכלתם אותה - איך התקנתי Ransomware על המצלמה הדיגיטלית שלכם	22
RTP - כיצד מזרימים מדיה בזמן אמת	40
דברי סיכום	58

Windows Kernel Fuzzing With Intel PT

מאת אריאל ז.

מבוא

במסגרת המאמר אציג שיטה חדשה לביצוע coverage guided fuzzing בקרנל של Windows תוך שימוש ב-WinAFL כמנוע ה-fuzzing וב-Intel PT כתשתית code trace שמחליפה את תשתית האינסטרומנטציה הסטנדרטית DynamoRIO.

ההנחה היא שהקורא מכיר ברמה סבירה את עולם ה-fuzzing ואת AFL בפרט, ואת עולם הרברסינג ודיבוג בקרנל של Windows. אך ראשית נתחיל בהגדרת מושגים בסיסיים והכרת הכלים איתם נעבוד.

Fuzzing, Coverage guided fuzzing

Fuzzing היא שיטה אוטומטית לגילוי באגים בתכנה במסגרתה מרעישים את הקלט לתכנה במטרה לגלות באגים, קריסות ומקרי קצה לא מטופלים.

ניתן לסווג כלי fuzzing ל-White, Grey, Black box:

- White box fuzzing - מכיר את קוד המקור של התכנית ומשתמש בשיטות ניתוח כמו symbolic execution ו-constraint solving במטרה לייצר קלטים שמגדילים כיסוי קוד או מגיעים לאזורי קוד קריטיים. כלים אלו יכולים להגיע לכיסוי קוד גדול אך דורשים מחיר חישובי גבוה.
- Black box fuzzing - משתמש בהכי פחות מידע על התכנית, מה שמאפשר לו ליצור קלטים באופן מהיר על חשבון העובדה שהם מייצרים הכי פחות כיסוי קוד.
- Grey box fuzzing או Coverage guided fuzzing - משתמש באינסטרומנטציה של ריצת התכנית על מנת למדוד כיסוי קוד שמייצר הקלט. כיסוי הקוד משמש לייצור קלטים שמטרתם להביא לכיסוי קוד מקסימלי. כלים אלו מהווים איזון טוב בין שני הסוגים הקודמים.

Intel PT

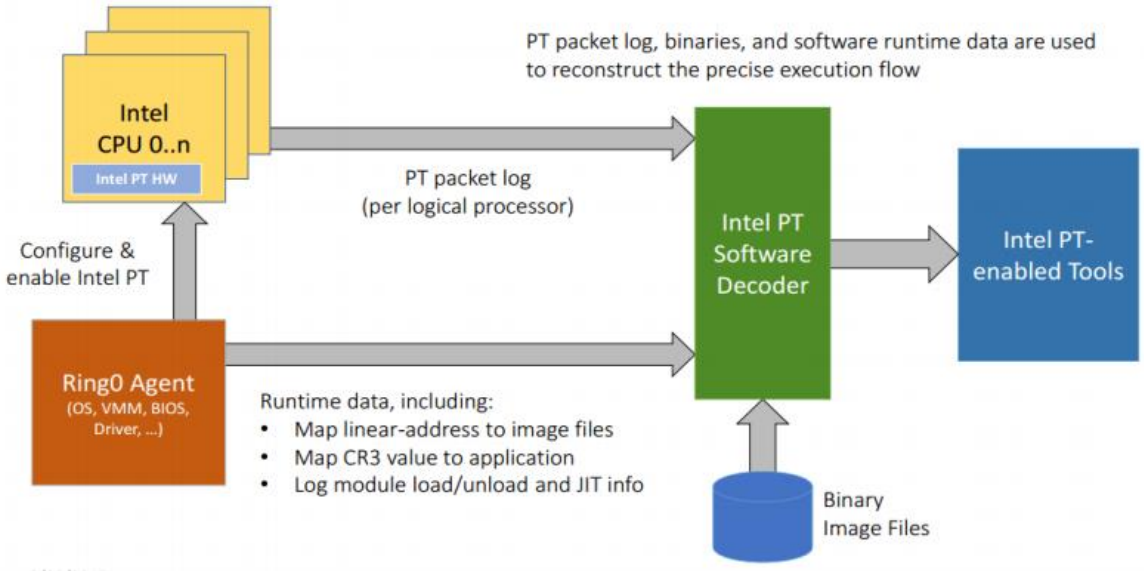
Intel PT (Processor Trace) הוא פיצ'ר חומרתי במעבדי אינטל המאפשר ביצוע tracing על ריצת קוד.

PT מייצר פאקטות מידע על הריצה ברמת ה-hardware thread (ליבה פיסית), מידע זה מאפשר בניה מחדש של רצף הפקודות של התכנית.

מה השימושים של PT?

- דיבוג - gdb ו-windbg משתמשים ב-PT על מנת לאפשר הקלטה והרצה קדימה ואחורה ב-trace התכנתי
- profiling
- performance analyzing - למשל linux perf
- fuzzing

איך PT עובד?



דרייבר שיושב במערכת ההפעלה (Ring0 Agent) מקנפג ומפעיל את PT באמצעות כתיבה וקריאה ל-MSR יעודיים. ה-trace נכתב בצורה מקודדת לזיכרון הפיסי.

ה-Decoder מפענח את ה-trace המקודד באמצעות תמונת זיכרון (image) של מודולים רלוונטים (כגון exe, dll, sys). נחזור לנקודה זו בהמשך המאמר.

אינטל מספקת ספרית decoding בשם libipt:

https://github.com/intel/libipt

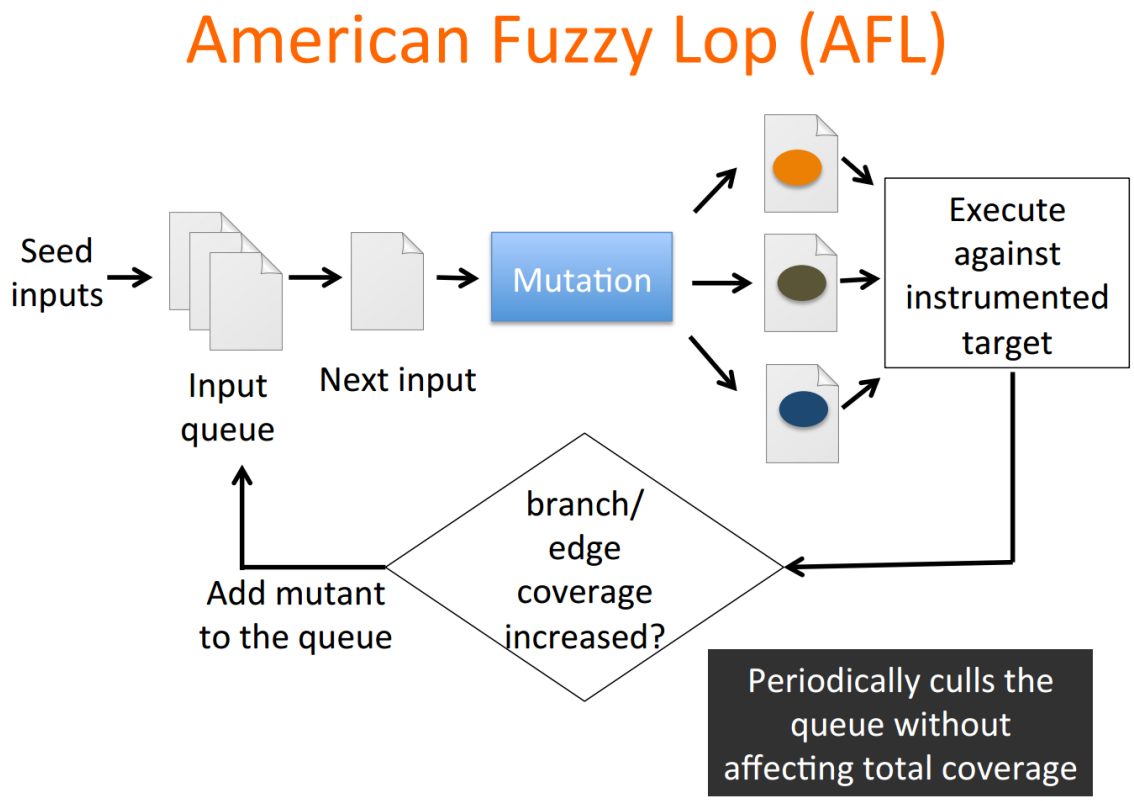
בקצה השרשרת נמצא רכיב היעד (Intel PT-enabled Tool), למשל debugger, profiler או fuzzer שמתממשק לרכיב המפענח ומשתמש ב-trace המפוענח לצרכיו.

WinAFL

WinAFL הנו פורט ל-windows של ה-fuzzer הלינוקסי AFL.

AFL הוא coverage guided fuzzer שמשתמש בהיוריסטיקות הרעשת קלט שהוכחו כיעילות מאד במציאת באגים.

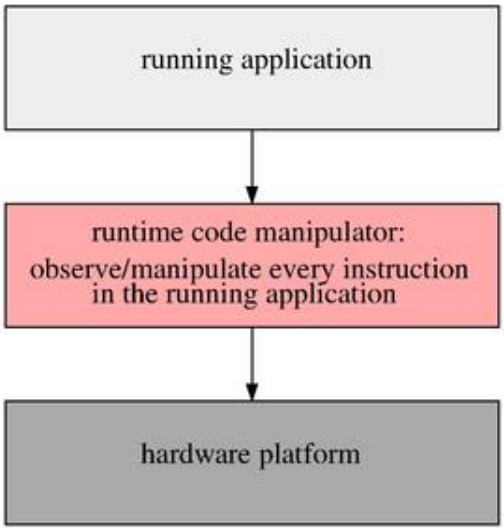
איך AFL עובד?



AFL מרעיש את הקלט של תכנית שרצה עם אינסטרומנטציה ומודד את כיסוי הקוד. קלט שמגדיל את הכיסוי (מביא למסלולים חדשים בקוד) מסווג כ"מעניין", נשמר בצד ומשמש בסיס להמשך הרעשה של הקלט.

איך AFL מבצע אינסטרומנטציה לקוד?

בלינוקס ניתן לקמפל את קוד התכנית עם אינסטרומנטציה מובנית במידה וקוד המקור זמין, או להשתמש ב-QEMU כשמדובר על fuzzing של אפליקציה closed source. ב Windows משתמשים בתשתית האינסטרומנטציה DynamoRIO. איך היא עובדת?



DynamoRIO רץ כשכבה חוצצת בין אפליקצית היעד למערכת ההפעלה, יש לה שליטה מלאה על ריצת האפליקציה וכזאת יודעת לנתח ולספק מידע על רצף הקוד של התכנית.

הבעיה באינסטרומנטציה היא הפגיעה בביצועים. לפי התיעוד QEMU ו-DynamoRIO מביאים לפגיעה של לפחות פי 2 בזמן הריצה של התכנית.

לאחרונה נוסף ל-WinAFL מוד trace חומרתי המבוסס על Intel PT שמחליף את השימוש ב-DynamoRIO.

יש לציין שמכיוון שה trace מבוצע בחומרה נצפה לקבל שיפור בביצועים אבל יש לקחת בחשבון ששלב הפענוח של ה-trace המקודד הנו כבד יחסית. כיום לפי הבנתי הביצועים פחות או יותר דומים ל-DynamoRIO אבל יש עוד מקום לאופטמיזיציה של שלב הפענוח.

שאלת המחקר

WinAFL הוא user mode fuzzer אבל PT מאפשר ביצוע trace של כל הקוד הרץ על המעבד[1] ובפרט הקוד של מערכת ההפעלה, לכן השאלה המתבקשת היא **האם ניתן להרחיב את WinAFL לתמיכה ב-kernel mode fuzzing ב-Windows**.

כלים קיימים

לפני שנתחיל לבחון את הבעיה, נבדוק אילו כלי coverage guided fuzzing קיימים כיום ל-Windows kernel והאם ניתן להשתמש בהם.

אז כיום ישנם שני כלים, שניהם משתמשים ב-AFL כמנוע ה-fuzzing וב-PT כתשתית code trace.

הכלים אמנם פומביים אבל לא ניתן להשתמש בהם כמוצר מדף מכיוון שהכותבים לא פרסמו גרסה מלאה שלהם.

1. WinAFL IntelPT (TALOS)

הוצג בכנס BH 2017. זהו ה-coverage guided fuzzer הפומבי הראשון ל-windows kernel. הכותב לא פרסם את רכיב הממשק בין PT ל-AFL לכן אין למעשה גרסה מלאה של הכלי. כמו כן הכלי משתמש בדרייבר custom ל-PT אבל הוא לא עובד בגרסאות חדשות של WIN10 בגלל עדכונים ב-patchguard.

https://github.com/intelpt/winafl-intelpt

2. kAFL

הוצג בכנס usenix 17. זהו כלי לינוקסי המשתמש ב-KVM Hypervisor על מנת לבצע fuzzing עם AFL בקרנל של מ"ה שרצה ב-VM. הכלי תומך ב-fuzzing של מ"ה Linux, Windows, MacOS אבל ה-Agents של Windows ו-MacOS לא שוחררו פומבית.

https://github.com/RUB-SysSec/kAFL

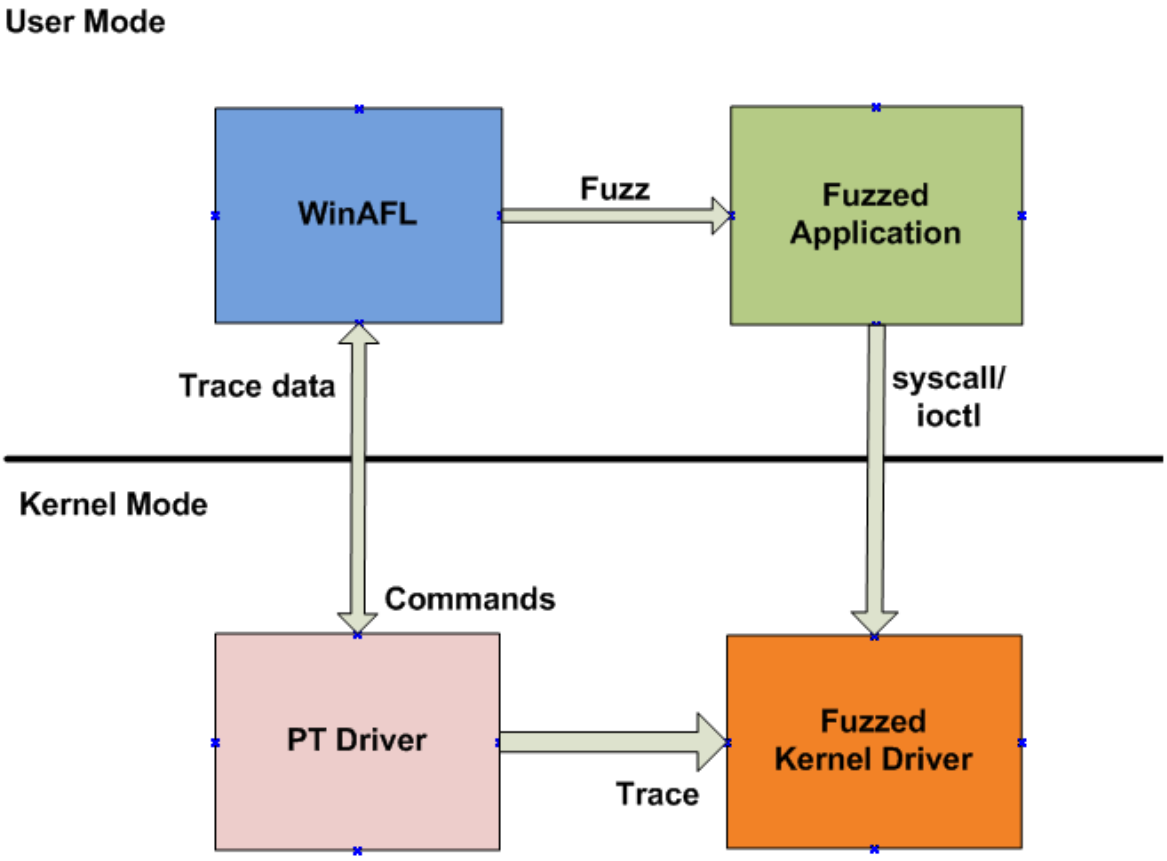
[1] למעשה ישנם איזורים עליהם לא ניתן לבצע trace: קוד SGX, וקוד SMM אלא אם עושים opt in ברמת ה-BIOS

מוטיבציה

לסיכום פרק המבוא, הרעיון הוא לכתוב אפליקציה שמדברת עם דרייבר ומטריגה בו פונקציונליות עליה רוצים לבצע fuzzing. WinAFL ירעיש את הקלטים לאפליקציה תוך מדידת כיסוי קוד קרנלי עם PT.

כלומר אנו למעשה מבצעים fuzzing על קוד הדרייבר כאשר האפליקציה משמשת כצינור להעברת הקלט.

הרעיון מתואר בדיאגרמה הבאה:



השמשה

השמשת WinAFL לעבודה עם PT

ראשית יש להוריד את הקוד של WinAFL ולקמפל אותו עם דגל INTELPT=1:

https://github.com/googleprojectzero/winafl

כדי לבצע fuzzing מבוסס PT במקום DynamoRIO יש להשתמש בפרמטר שורת פקודה -P במקום -D.

מומלץ גם לקרוא את ה-readme על השילוב של PT ב-WinAFL:

https://github.com/googleprojectzero/winafl/blob/master/readme_pt.md

PT ב-Windows

על מנת להבין את השינויים שנבצע בקוד יש להכיר את הדרייבר של PT ב-Windows.

הדרייבר Ipt.sys אחראי לממשק בין רכיבי user mode שזקוקים לשירותים של PT, כמו ETW או WinDbg, למנגנון PT החומרתי. הדרייבר חושף ממשק ioctl סטנדרטי לבקשות משתמש:

```
            return 0xC000000D;
          v38 = IptQueryCoreTrace(&v41);
        }
        else
        {
          v38 = IptQueryProcessTrace(*(a1 + 8), &v41);
        }
        v10 = v38;
        *v9 = v41;
        return v10;
      }
      v39 = IptResumeThreadTrace(*(a1 + 8), &v47, a3, v8);
    }
    else
    {
      v39 = IptPauseThreadTrace(*(a1 + 8), &v47);
    }
    v10 = v39;
    *v9 = v47;
    return v10;
  }
  v33 = IptStopProcessTrace(*(a1 + 8));
LABEL_62:
  v10 = v33;
  goto LABEL_63;
}
if ( v6 == 5 )
{
  v33 = IptStartProcessTrace(*(a1 + 8), *(a1 + 16));
  goto LABEL_62;
```

אפשור trace קרנלי

בקוד של WinAFL נבחן את הפונקציה run_target_pt שתפקידה להריץ בצורה מחזורית את ה- target function (הפונקציה עליה מבצעים את ה-fuzzing):

```
// start tracing
if ((!options.persistent_trace) || (fuzz_iterations_current == 0))
{
    IPT_OPTIONS ipt_options;

    memset(&ipt_options, 0, sizeof(IPT_OPTIONS));

    ipt_options.OptionVersion = 1;

    ConfigureBufferSize(options.trace_buffer_size, &ipt_options);

    ConfigureTraceFlags(0, &ipt_options);

    if (!StartProcessIptTracing(child_handle, ipt_options))
    {
        FATAL("ipt tracing error\n");
    }

    last_ring_buffer_offset = 0;
}
```

הפונקציה ConfigureTraceFlags מקנפגת את הדגלים השונים בבקשת ה-trace, למשל האם ה-trace הוא של קוד user או קרנל. הפרמטר הראשון שערכו 0 מסמן בקשת trace של קוד user.

נבחן דגלים נוספים:

```
#define IPT_TOOL_USE_MTC_TIMING_PACKETS     0x01
#define IPT_TOOL_USE_CYC_TIMING_PACKETS     0x02
#define IPT_TOOL_TRACE_KERNEL_MODE          0x04
#define IPT_TOOL_TRACE_ALL_MODE             0x08
```

שני הדגלים הראשונים מקנפגים אילו פאקטות מידע יש לכלול ב-trace.

הדגל IPT_TOOL_TRACE_KERNEL_MODE מסמן בקשת trace קוד קרנל בלבד.

הדגל IPT_TOOL_TRACE_ALL_MODE מסמן בקשת trace של קוד user וקרנל יחד.

אם כן, עלינו לשנות את בקשת הקנפוג ל-trace קרנלי:

```
// start tracing
if ((!options.persistent_trace) || (fuzz_iterations_current == 0))
{
    IPT_OPTIONS ipt_options;

    memset(&ipt_options, 0, sizeof(IPT_OPTIONS));

    ipt_options.OptionVersion = 1;

    ConfigureBufferSize(options.trace_buffer_size, &ipt_options);

    //ConfigureTraceFlags(0, &ipt_options);
    ConfigureTraceFlags(4, &ipt_options);

    if (!StartProcessIptTracing(child_handle, ipt_options))
    {
        FATAL("ipt tracing error\n");
    }

    last_ring_buffer_offset = 0;
}
```

לא השתמשנו בשם הדגל IPT_TOOL_TRACE_KERNEL_MODE מכיוון שהוא לא מוכר ב-scope באותה נקודה, כך WinAFL בנוי כרגע.

ננסה להריץ את WinAFL לאחר השינוי על תכנית לדוגמא:

```
Command Prompt
C:\Shared\winafl_intelpt\winafl-master\build64\bin\Release>afl-fuzz.exe -i in -o out -P -t 20000 -- -fuzz_i
terations 5000 -target_module harness.exe -target_offset 0x1390 -nargs 2 -nopersistent_trace -- harness.exe
 @@
WinAFL 1.16b by <ifratric@google.com>
Based on AFL 2.43b by <lcamtuf@google.com>
IPT service enebled
[+] You have 4 CPU cores and 0 runnable tasks (utilization: 0%).
[+] Try parallel jobs - see docs\parallel_fuzzing.txt.
[*] Checking CPU core loadout...
[+] Found a free CPU core, binding to #0.
[+] Process affinity is set to 1.

[*] Setting up output directories...
[+] Output directory exists but deemed OK to reuse.
[*] Deleting old session data...
[+] Output dir cleanup successful.
[*] Scanning 'in'...
[+] No auto-generated dictionary tokens to reuse.
[*] Creating hard links for all input files...
[*] Attempting dry run with 'id_000000'...

[-] PROGRAM ABORT : ipt tracing error

        Location : run_target_pt(), C:\Shared\winafl_intelpt\winafl-master\winaflpt.c:1503


C:\Shared\winafl_intelpt\winafl-master\build64\bin\Release>
```

קיבלנו שגיאה "ipt tracing error".

ניתן לראות בקוד שצירפנו לעיל שהשגיאה נובעת מכישלון בפונקציה StartProcessIptTracing שמבקשת מהדרייבר להתחיל להקליט trace.

נבחן את קוד הפונקציה:

```
BOOL
StartProcessIptTracing (
    _In_ HANDLE hProcess,
    _In_ IPT_OPTIONS ullOptions
    )
{
    BOOL bRes;
    HANDLE hIpt;
    IPT_INPUT_BUFFER inputBuffer;
    IPT_OUTPUT_BUFFER outputBuffer;

    //
    // Open the IPT Device
    //
    bRes = OpenIptDevice(&hIpt);
    if (bRes != FALSE)
    {
        //
        // Send a request to start tracing for this process
        //
        InitializeIptBuffer(&inputBuffer, IptStartProcessTrace);
        inputBuffer.StartProcessIptTrace.Options = ullOptions;
        inputBuffer.StartProcessIptTrace.ProcessHandle = hProcess;
        bRes = DeviceIoControl(hIpt,
                               IOCTL_IPT_REQUEST,
                               &inputBuffer,
                               sizeof(inputBuffer),
                               &outputBuffer,
                               sizeof(outputBuffer),
                               NULL,
                               NULL);
        CloseHandle(hIpt);
    }
    return bRes;
}
```

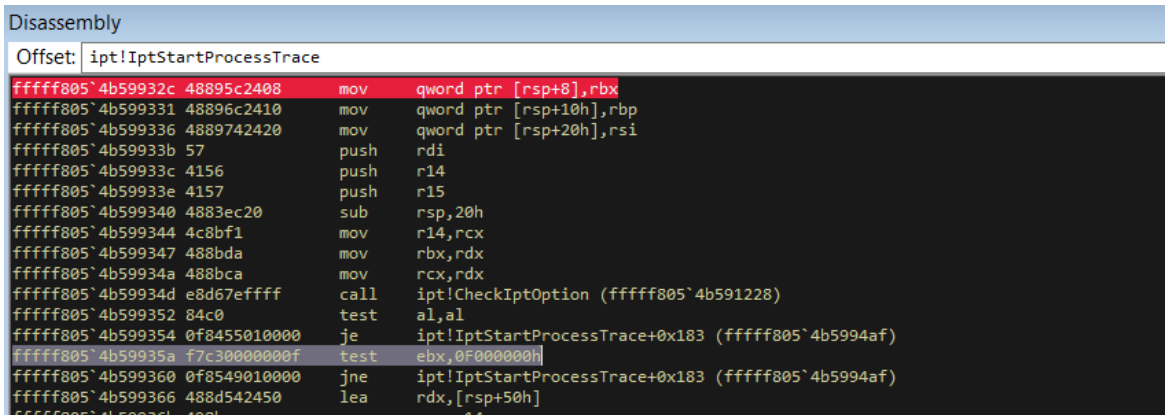
הפונקציה נכשלת בבקשת ה-DeviceIoControl לדרייבר. על מנת להבין את הסיבה לשגיאה עלינו לדבג את הדרייבר בקרנל.

בסופו של דבר בקשת ה-ioctl מגיעה לפונקציה IptStartProcessTrace בדרייבר:

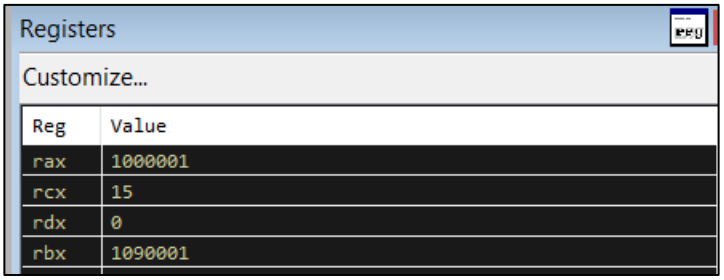
```
__int64 __fastcall IptStartProcessTrace(__int64 a1, __int64 a2)
{
  void *process_handle; // r14@1
  __int64 v3; // rbx@1
  __int64 result; // rax@3
  unsigned int v5; // er15@4
  __int64 pid; // rsi@4
  signed __int64 v7; // rax@6
  __int64 _process_context; // rdi@6
  volatile signed __int32 *process_context; // rax@13
  PVOID eprocess; // [rsp+50h] [rbp+18h]@3

  process_handle = a1;
  v3 = a2;
  if ( !CheckIptOption(a2) || v3 & 0xF000000 )
    return 0xC000000Di64;
  result = GetProcessFromHandle(process_handle, &eprocess);
  if ( result < 0 )
    return result;
  v5 = 0xC0000001;
  pid = PsGetProcessId(eprocess);
  KeEnterCriticalRegion();
  ExAcquirePushLockSharedEx(&push_lock, 0i64);
```

ניתן לראות כי יש בדיקות שמפלטרות בקשות trace מסוימות.

מה בדיוק מפולטר?



ניתן לראות כי את בדיקת CheckIptOption עוברים בהצלחה:



הבדיקה השניה מוודאת שביטים 24-27 ב-ebx מאופסים, אבל ניתן לראות שביט 24 דולק לכן נכשלים בבדיקה והבקשה נדחית.

מהו אותו ערך 0x1090001 שמגיע לדרייבר? זהו מבנה IPT_OPTIONS של הדגלים המועברים לדרייבר בבקשת הקלטת trace:

```
typedef union _IPT_OPTIONS
{
    struct
    {
        DWORDLONG OptionVersion : 4;    // Must be set to 1
        DWORDLONG TimingSettings : 4;   // IPT_TIMING_SETTINGS

        DWORDLONG MtcFrequency : 4;     // Bits 14:17 in IA32_RTIT_CTL
        DWORDLONG CycThreshold : 4;     // Bits 19:22 in IA32_RTIT_CTL

        DWORDLONG TopaPagesPow2 : 4;    // Size of buffer in ToPA, as 4KB powers of 2 (4KB->128MB)
        DWORDLONG MatchSettings: 3;     // IPT_MATCH_SETTINGS
        DWORDLONG Inherit : 1;          // Children will be automatically added to the trace

        DWORDLONG ModeSettings : 4;     // IPT_MODE_SETTINGS
        DWORDLONG Reserved : 36;
    };
    DWORDLONG AsULonglong;
} IPT_OPTIONS, *PIPT_OPTIONS;
```

בפונקציה ConfigureTraceFlags שהצגנו לעיל מקנפגים את הדגלים:

```
//
// Choose the right mode setting
//
if (dwFlags & IPT_TOOL_TRACE_KERNEL_MODE)
{
    pOptions->ModeSettings = IptCtlKernelModeOnly;
}
else if (dwFlags & IPT_TOOL_TRACE_ALL_MODE)
{
    pOptions->ModeSettings = IptCtlUserAndKernelMode;
}
else
{
    pOptions->ModeSettings = IptCtlUserModeOnly;
}
```

(enum _IPT_MODE_SETTINGS)IptCtlKernelModeOnly = 1
Sets BranchEn[2000], ToPA[100], OS[4]

ניתן לראות שבבקשת trace קרנלי נכתב הערך 1 לשדה ModeSettings שמאכלס את ביטים 24-27 כלומר מדליק את ביט 24 וזה מה שמכשיל את הבדיקה שכן ראינו שביטים אלו חייבים להיות מאופסים. כזכור ראינו שבקשת user mode trace כותבת ערך 0 לשדה זה.

כלומר למעשה אנחנו רואים **שהדרייבר של מיקרוסופט לא מאפשר ביצוע trace קרנלי**. סיבה אפשרית להגבלה היא הגנה מפני מעקף KASLR. כידוע החל מ-WIN 8.1 תהליכים שרצים ב-sandbox/low il אינם יכולים לקרוא את כתובות הבסיס של מודולים בקרנל כדי להקשות על השמשת חולשות קרנל. Trace קרנלי עלול לשמש כמעקף.

מניסיוני בגרסת Win10 RS5 תהליך שרץ ב-low il אכן יכול לפנות לדרייבר PT ולהקליט ולקרוא trace.

על מנת לעקוף את ההגבלה הזו עלינו לפצ'פצ' את קוד הבדיקה בדרייבר:

```
fffff805`4b59934a 488bca          mov     rcx,rdx
fffff805`4b59934d e8d67effff      call    ipt!CheckIptOption (fffff805`4b591228)
fffff805`4b599352 84c0            test    al,al
fffff805`4b599354 0f8455010000    je      ipt!IptStartProcessTrace+0x183 (fffff805`4b5994af)
fffff805`4b59935a f7c30000000f    test    ebx,0F000000h
fffff805`4b599360 90              nop
fffff805`4b599361 90              nop
fffff805`4b599362 90              nop
fffff805`4b599363 90              nop
fffff805`4b599364 90              nop
fffff805`4b599365 90              nop
fffff805`4b599366 488d542450      lea     rdx,[rsp+50h]
fffff805`4b59936b 498bce          mov     rcx,r14
fffff805`4b59936e e84dffffff      call    ipt!GetProcessFromHandle (fffff805`4b5992c0)
```

יצוין כי אותה הגבלה קיימת גם בפונקציה המחזירה את ה-trace לקורא (IptGetProcessTrace) לכן יש לפצ'פצ' גם את קוד הפונקציה הזו:

```
v8 = 0xC0000001;
KeEnterCriticalRegion();
ExAcquirePushLockSharedEx(&push_lock, 0i64);
if ( !byte_1C0005381 && !byte_1C0005380 )
{
  v9 = PsGetProcessId(Object);
  v10 = IptLookupProcess(v9);
  v11 = v10;
  if ( v10 )
  {
    // !!! kernel trace flag check !!!
    if ( !(*(v10 + 48) & 0xF000000) && 2 == _InterlockedCompareExchange((v10 + 64), 3, 2) )
    {
      RunOnEachThread(*(v10 + 40), GetTrace, &v15);
      v12 = a5;
      v13 = v18;
      v14 = v17;
      *v5 = v18;
      if ( v12 )
        *v12 = v14;
      if ( v15 && v13 > v14 )
      {
        v8 = 0xC0000023;
        if ( v14 )
          v8 = 0x80000005;
```

נציין בנקודה זו שבאופן כללי פיצ'פוצ' קוד של מודולים קרנליים מנוטר ע"י Patchguard ועלול לגרום ל-BSOD, לכן ההמלצה היא לרוץ במוד Debug שבו Patchguard מנוטרל. מניסיוני VMWare/VBox עדיין לא מבצעים אמולציה של PT לכן כרגע יש לדבג מכונה פיסית.

לאחר פיצ'פוצ'י הקוד נוסיף הדפסות debug בפונקציה analyze_trace_full_reference שתפקידה לפענח את ה-trace ולעדכן את כיסוי הקוד, וננסה להריץ שוב את ה-fuzzer:

```
Command Prompt
[+] Try parallel jobs - see docs\parallel_fuzzing.txt.
[*] Checking CPU core loadout...
[+] Found a free CPU core, binding to #0.
[+] Process affinity is set to 1.

[*] Setting up output directories...
[+] Output directory exists but deemed OK to reuse.
[*] Deleting old session data...
[+] Output dir cleanup successful.
[*] Scanning 'in'...
[+] No auto-generated dictionary tokens to reuse.
[*] Creating hard links for all input files...
[*] Attempting dry run with 'id_000000'...
decoder->ip: FFFFF8060DCF8E5D status: -13
decoder->ip: FFFFF80601863214 status: -13
decoder->ip: FFFFF809DFC024F2 status: -13
decoder->ip: FFFFF80601900D05 status: -13
decoder->ip: FFFFF806018C0957 status: -13
decoder->ip: FFFFF809DFC0CA84 status: -13
decoder->ip: FFFFF8060DA5D220 status: -13
decoder->ip: FFFFF806019B9232 status: -13

[-] PROGRAM ABORT : No instrumentation detected
         Location : perform_dry_run(), C:\Shared\winafl_intelpt\winafl-master\afl-fuzz.c:3047

C:\Shared\winafl_intelpt\winafl-master\build64\bin\Release>
```

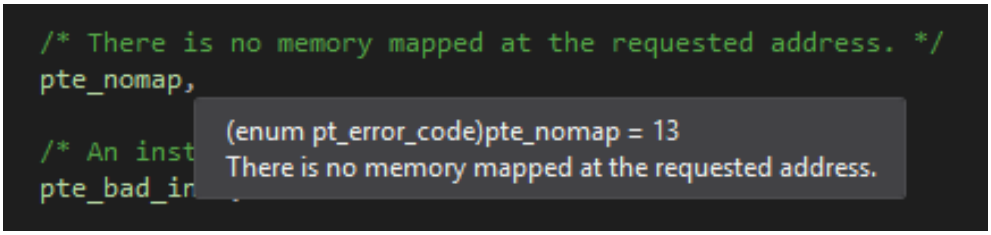
ניתן לראות שהצלחנו לקבל trace קרנלי - הכתובות ...0xFFFFF80.

מצוין, סיימנו את החלק הזה.

הוספת תמונת זיכרון ל-Decoder

ניתן לראות בצילום המסך לעיל שכעת מקבלים שגיאה אחרת "No instrumentation detected" והפונקציה pt_blk_next שתפקידה להחזיר את ה-basic block הבא ב-traceמחזירה שגיאה 13-.

נחפש את השגיאה 13- בקוד:



משמעות השגיאה pte_nomap היא שה decoder לא מצא תמונת זיכרון (image) שמתאימה לכתובות שהתקבלו ב-trace.

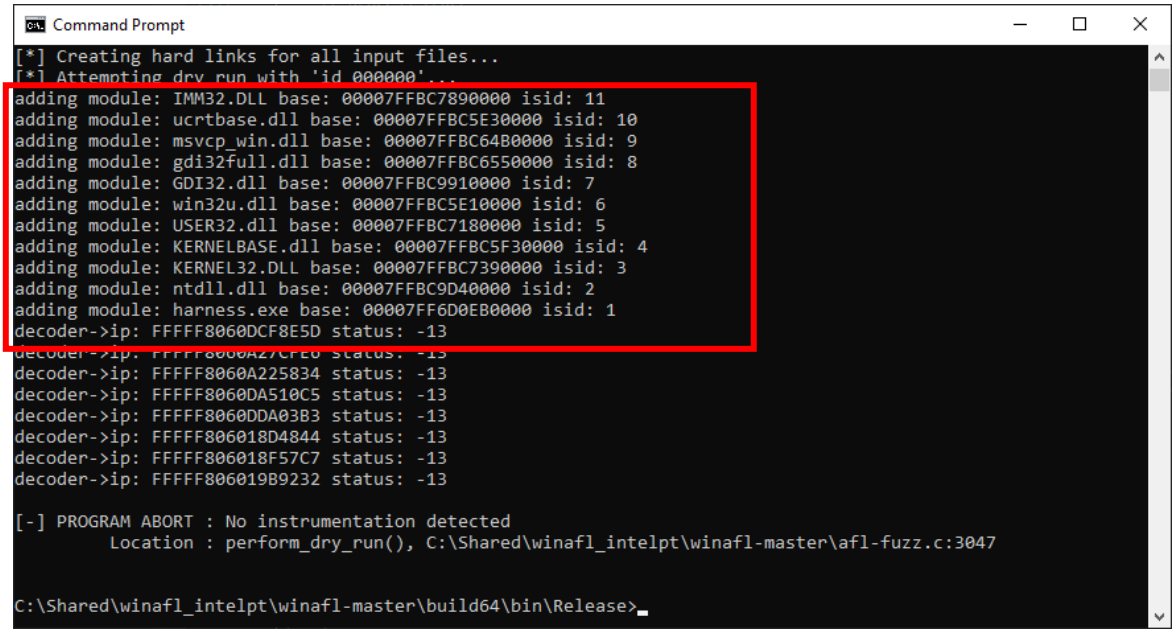
למה בעצם ה-decoder זקוק ל-image? מכיוון שה trace הגולמי אינו מכיל פקודות מעבד אלא רק "meta data" על ריצת הקוד כמו יעדי branch או האם branch נלקח או לא. ה-decoder משתמש ב-image על מנת לפענח את פקודות ה-assembly בכתובות שמתקבלות ב-trace, על מנת למשל להרכיב את כתובת הפקודה הבאה לביצוע או להרכיב basic blocks ברצף הקוד.

איך WinAFL מתמודד עם הדרישה הזו? הוא מפרסר ובונה לעצמו רשימה של המודולים הטעונים במרחב ה-user mode של התכנית.

בקטע הקוד הבא בפונקציה run_target_pt ניתן לראות שרצים על רשימת all_modules שמחזיקה descriptors של המודולים ב-user mode ומוסיפים אותם למבנה image של ה-decoder של אינטל. ה-descriptor מכיל את שם המודול, כתובת בסיס, גודל ואת ה-image של המודול הטעון.

```
struct pt_image *image = NULL;
if ((options.decoder == DECODER_FULL_FAST) || (options.decoder == DECODER_FULL_REFERENCE))
{
    image = pt_image_alloc("winafl_image");
    module_info_t *cur_module = all_modules;

    while (cur_module)
    {
        if (cur_module->isid > 0)
        {
            int ret = pt_image_add_cached(image, section_cache, cur_module->isid, NULL);
        }

        cur_module = cur_module->next;
    }
}
```

נוסיף הדפסות debug לקטע הקוד:



ניתן לראות שאכן כל ה-user mode modules נוספו ל-decoder.

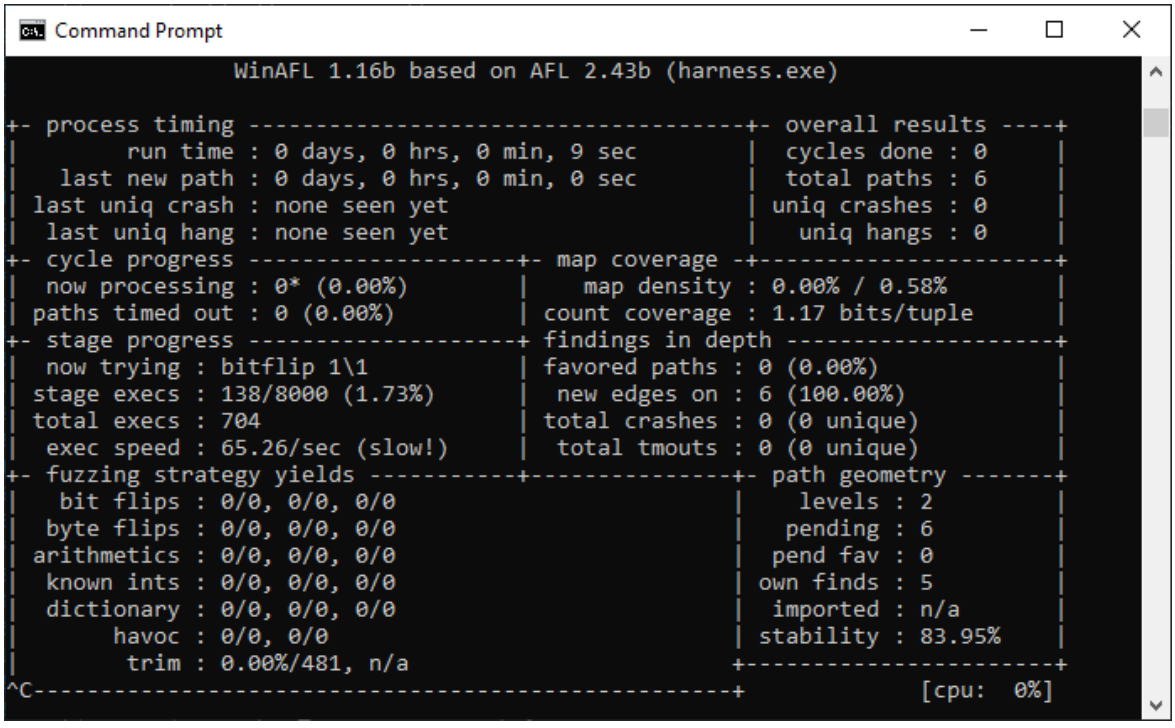
לכן על מנת לפתור את שגיאת pte_nomap עלינו להוסיף את ה-image-ים המתאימים **ממרחב הקרנל** לרשימת המודולים. לשם כך ניתן לשלוף את כתובת הבסיס, גודל ולהוריד dump של הדרייבר באמצעות windbg. לאחר מכן נוסיף פונקציה שמוסיפה את הדרייבר שהורדנו לרשימת המודולים:

```
void add_module_section_cache(char* modulename, void* base, size_t size)
{
    module_info_t *loaded_module;

    loaded_module = (module_info_t *)malloc(sizeof(module_info_t));

    strcpy(loaded_module->module_name, modulename);
    loaded_module->base = base; //module_info.lpBaseOfDll;
    loaded_module->size = size; //module_info.SizeOfImage;

    // todo put these files in a separate directory and clean it periodically
    char tmpfilename[MAX_PATH];
    sprintf(tmpfilename, "%s", modulename);

    loaded_module->isid = pt_iscache_add_file(section_cache, tmpfilename, 0, size, base);

    if (loaded_module->isid <= 0) {
        FATAL("Error adding file to pt cache.");
    }

    loaded_module->next = all_modules;
    all_modules = loaded_module;
}
```

ונקרא לה:

```
struct pt_image *image = NULL;
if ((options.decoder == DECODER_FULL_FAST) || (options.decoder == DECODER_FULL_REFERENCE)) {

    ////////////////////////////////////////////////////////////////////

    // 07/2019 pt kernel trace

    add_module_section_cache(options.driver_image, options.driver_base, options.driver_size);

    ////////////////////////////////////////////////////////////////////

    image = pt_image_alloc("winafl_image");
    module_info_t *cur_module = all_modules;

    while (cur_module) {
        if (cur_module->isid > 0) {
            int ret = pt_image_add_cached(image, section_cache, cur_module->isid, NULL);
        }
        cur_module = cur_module->next;
    }
}
```

נריץ שוב את ה-fuzzer ונבחין כי כעת איננו מקבלים שגיאות ו-WinAFL מתחיל לעבוד תוך כדי הצגת מסך הסטטוס המוכר שלו:



לסיכום שלב זה, נתקלנו בשגיאה "No instrumentation detected" שנבעה מכך שה decoder לא הצליח לפענח את ה-trace מכיוון שה image של הדרייבר היה חסר.

פתרנו את השגיאה באמצעות dump של הדרייבר והוספה של descriptor מתאים לרשימת המודולים.

בעית יציבות ה-Fuzzer

ניקח לדוגמא תרחיש שבו הדרייבר עליו מבצעים fuzzing יוצר כמה thread-ים בזמן טעינה. ה-trace שיוקלט יכיל גם את מסלולי הקוד של ה-thread-ים וגם את המסלולים שהתקבלו כתוצאה מביצוע ה-fuzzing.

מצב כזה יפגע בביצועי AFL. מדוע?

כדי להבין את משמעות הפגיעה בביצועים יש להכיר את מדד ה-Stability של AFL שאומר עד כמה ה-fuzzing "יציב". כאשר נאסף כיסוי קוד שלא נוצר מביצוע ה-fuzzing, ה-Stability קטן וביצועי ה-fuzzing נפגעים מכיוון שההחלטות ש-AFL מקבל לא בהכרח מובילות אותו להגדלת כיסוי הקוד.

בתרחיש שתיארנו מתווספים מסלולי קוד שאינם קשורים למסלולים שמתקבלים במסגרת ה-fuzzing, מסלולים אלו יטעו את AFL לחשוב שהקלטים שהוא חולל מגדילים את כיסוי הקוד.

פתרון אפשרי הוא מנגנון ב-PT שמאפשר פילטור ה-trace פר תהליך, כלומר איסוף כיסוי קוד רק בקונטקסט של תהליך ה-fuzzing.

PT מממש זאת באמצעות קינפוג ערך רגיסטר CR3 שכידוע הנו ייחודי לכל תהליך. ברגע שקובעים ערך CR3 של תהליך יעד, ה-trace יקליט רק קוד שרץ בקונטקסט של אותו תהליך. הדרייבר של windows לא תומך במנגנון זה אבל ניתן לממש אותו באמצעות הוספת shellcode בזמן ריצה.

נציין שפיתרון זה לא מטפל בכל התרחישים.

תרחיש אחר הוא דרייבר שמטפל בפסיקות ורושם callback שנקרא ע"י מערכת ההפעלה בקבלת פסיקה. סטטיסטית ייתכן שה-callback ירוץ בקונטקסט של תהליך ה-fuzzing מכיוון ש-DPC של פסיקה רץ ב-arbitrary user thread context, כלומר אנו שוב בבעיה של קוד שרץ על הדרייבר ואינו חלק מתהליך ה-fuzzing.

על מנת לפתור את הבעיה הזו מהשורש יש לפלטר את ה-trace ברמת ה-kthread או לשבת בצומת שמאפשר פילטור trace בקונטקסט של DPC. ישנם כמה כיוונים לפיתרון כזה אך הם חורגים מה scope של המאמר ולכן לא נתאר אותם.

לסיכום, השיטה שתיארנו עובדת כרגע על דרייברים שרץ עליהם קוד שמוטרג מתהליך ה-fuzzer בלבד. כמו כן תיארנו כיוונים נוספים להרחבת התמיכה גם לדרייברים אחרים, "מורכבים" יותר.

סיכום

במאמר זה הוצגה שיטה לביצוע coverage guided fuzzing בקרנל של Windows באמצעות יכולת tracing חומרתית של מעבדי אינטל. לשם כך בוצעה הרחבה לקוד של WinAFL ושינויי קוד דינמיים בדרייבר של PT ב-Windows. כמו כן הוצגו תרחישים בעייתיים שעלולים לפגוע בביצועי AFL ופתרונות אפשריים.

יובהר כי המתואר הנו POC בלבד וכי יש מקום להרחבות נוספות שיהפכו את התהליך לאוטומטי יותר.

על המחבר

אריאל ז. עובד בחברת רפאל כחוקר אבטחת מידע.

arielze[at]rafael[dot]co[dot]il

מקורות מידע נוספים

1. Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3, Chapter 36, INTEL PROCESSOR TRACE:

https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3c-part-3-manual.pdf

2. Harnessing Intel Processor Trace on Windows for Vulnerability Discovery, HITB 2017

https://conference.hitb.org/hitbsecconf2017ams/materials/D1T1 - Richard Johnson - Harnessing Intel Processor Trace on Windows for Vulnerability Discovery.pdf

3. Evolutionary Kernel Fuzzing, Black Hat USA 2017

https://moflow.org/Presentations/Evolutionary Kernel Fuzzing-BH2017-rjohnson-FINAL.pdf

4. Intel Processor Trace on Linux, Tracing Summit 2015

https://halobates.de/pt-tracing-summit15.pdf

5. Hardware Tracing with Intel Processor Trace

https://hsdm.dorsal.polymtl.ca/system/files/10Dec2015_0.pdf

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.

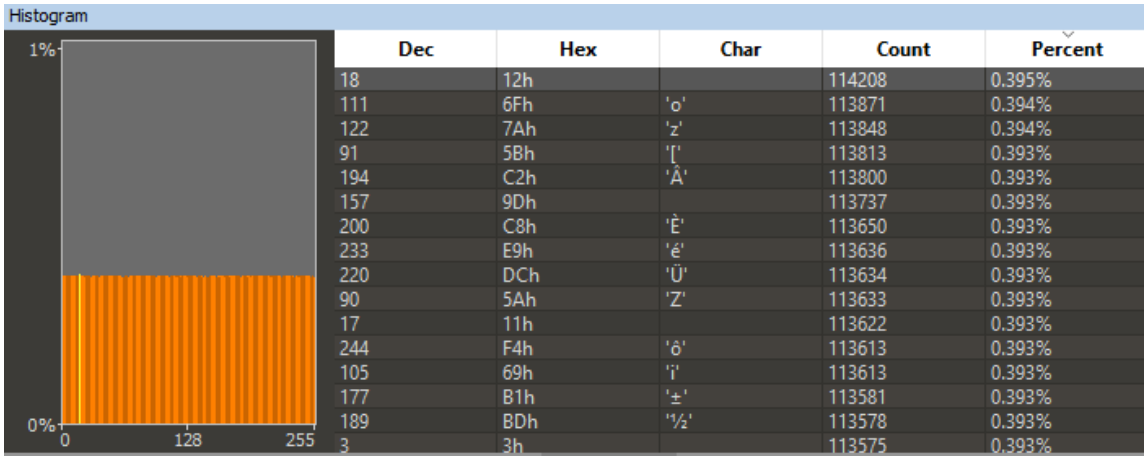
דוגמא לתיעוד של ML על פונקציית מערכת ההפעלה האחראית ליצירת Task חדש:

```
/** 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.

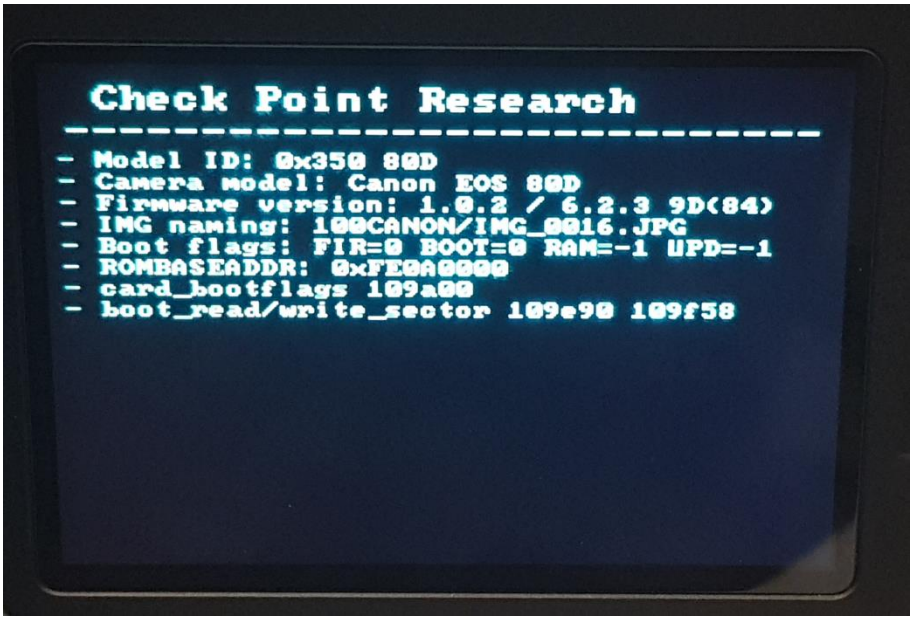
זמן לקצת צופן

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:



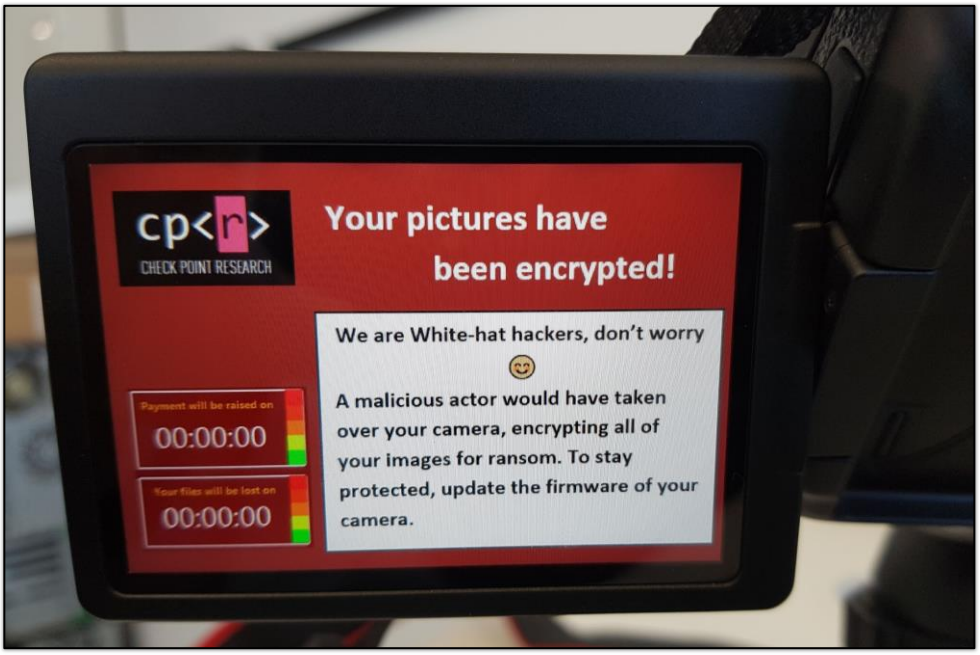
תוצאות הרצת "עדכון התוכנה" העוין שלנו, שכעת עושה שימוש בכותרת שלנו ☺

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/

RTP - כיצד מזרימים מדיה בזמן אמת

מאת שחף גרגיר ודביר גולן

מבוא

יישומים רבים משתמשים בשיחות והעברת מידע בזמן אמת דרך האינטרנט, כדוגמת WhatsApp, Skype שבהן מבוצעות בעזרת האינטרנט שיחות וידאו ורגילות ועוד (חשוב להדגיש כי אפליקציות אלו הן דוגמאות להמחשה של השימוש בהעברת מידע בזמן אמת ואינן עושות שימוש בפרוטוקולים המתוארים במאמר).

מאמר זה, יעסוק בפרוטוקול Real-time Transport Protocol, או בקיצור RTP, שלרוב פועל מעל UDP בפורטים 16,384 - 32,767[2], חשוב לציין כי קיימים מקרים בהם נעשה שימוש גם ב-TCP, כאשר אמינות המידע חשובה. לדוגמה בהעברת קבצים בזמן אמת.

כמו כן, נסביר אודות הפרוטוקול, האופן בו הוא מתבצע, מבנה חבילות המידע (ה-Headers והערכים המופיעים בו). בנוסף, נציג דוגמאות למימושים שונים של הפרוטוקול ונתאר בקצרה כיצד מצפינים את המידע המועבר וכיצד מבטיחים את מהימנות המידע. יתר על כן, נציג שני רכיבים: Mixers ו-Translators, המשמשים להעברת מידע ב-RTP.

במהלך המאמר ניתקל רבות בפרוטוקול Real-time Transport Control Protocol, RTCP, הקרוב מאוד לפרוטוקול RTP, לכן נסביר גם אודותיו, ונסביר בקצרה על ניתוח RTP באמצעות התוכנה Wireshark.

קיים מאמר נוסף ב-DigitalWhisper בגיליון מספר 60, בשם "הצפנת VoIP למתחילים" המתעסק ב-RTP אך הוא מתמקד בעיקר בהצפנת חבילות המידע, ואנו במאמר זה מרחיבים ומתעסקים בכל הפרוטוקול, כפי שציינו למעלה. העמקה בנושא ההצפנה ניתן לקרוא במאמר לעיל.

[2] נלקח מ-https://www.voip-info.org/rtp-ports

מושגים חשובים

לפני שנתחיל להסביר אודות הפרוטוקול וכל מה שמלווה בכך, נציין מספר מושגים שכדאי להכיר ולהבין לפני קריאת המאמר, ונסביר על כל אחד מהם בקצרה:

- **RTP Payload** - המידע המועבר על ידי RTP בחבילת המידע. למשל, דגימות אודיו או וידאו דחוס.

- **עיקרון קצה לקצה** - (End-to-End Principle) עיקרון עיצובי זה קובע כי, פעולות מסוימות עם מטרות ספציפיות ליישום, אמורות לקרות אצל משתמש הקצה ולא בצומת אמצעי כלשהו ברשת (למשל בנתב וכו'), כל עוד ניתן לממש אותה אצל המשתמש הנ"ל בצורה מלאה ונכונה. ברור כי פחות משתלם להוסיף פונקציות לרשת, ובפרט במקרים בהם משתמשי הקצה צריכים לממש אותן בכל מקרה. עיקרון זה ממומש לרוב על מנת לשמור על רמת סיבוכיות נמוכה של הרשת ואופן הפעולה שלה, וזמן ביצוע "סביר יחסית" לכלל המשתמשים.

- **ריצוד** - באלקטרוניקה, ריצוד הוא סטייה ממחזוריות של אותות, הגורם ל"הפרעה" מסוימת ביישום. ריצוד יכול לגרום לצג התצוגה להבהב, להשפיע על הביצועים של מעבדים במחשבים אישיים, להציג השפעות לא רצויות אחרות באותות שמע, ולגרום לאובדן נתונים המועברים בין התקני רשת, וכדומה.

- **חבילות מידע אבודות (Packet Loss)** - חבילות מידע נתונים אשר "מטיילות" ברשת מחשבים ואינן מגיעות ליעדן (הן לא מסתובבות לעד ברשת האינטרנט בעקבות ה-TTL בפרוטוקול IP, שהוא מספר מוגדר המציין כמות של רכיבים רשתיים בהם חבילת המידע יכולה לעבור. לאחר שה-TTL מגיע ל-0, חבילת המידע "נופלת"). אובדן זה נגרם עקב שגיאות בהעברת הנתונים. הפסד חבילות מידע נמדד לפי אחוז חבילות המידע שאבדו ביחס חבילות מידע שנשלחו. TCP למשל, מגלה אובדן כזה ומבצע שידורים חוזרים של חבילות המידע כדי להבטיח העברת הודעות מהימנות. בהקשר של הפרוטוקול שלנו, Real-Time Transport Protocol, לרוב אין שימוש ב-TCP מכיוון שהמידע צריך לעבור בזמן קצר ככל הניתן, ולא רצוי "לבזבז" זמן על שליחת חבילות המידע הפגומות/האבודות מחדש, לכן RTP פועל מעל UDP. בהמשך נראה כי ניתן לסדר את הבעיות נ"ל באמצעות אלגוריתם לתיקון שגיאות.

- **Error Concealment** - הסתרת שגיאות הינה טכניקה המשמשת בעיבוד אותות שמטרתה למזער את הידרדרות האותות הנגרמת בעקבות נתונים חסרים, כלומר אובדן חבילות מידע. אות הוא בעצם מסר שנשלח ממשדר למקלט במספר חבילות מידע קטנות.

- **RTP Session** - קבוצת משתמשים המעבירה בינה לבין עצמה מידע בעזרת RTP, וכל משתמש מדווח לכל המשתמשים האחרים על המידע שקיבל בעזרת RTCP (החלק האחראי על השליטה והבקרה בפרוטוקול). כל משתתף מגדיר Transport Address עבור כל Session שהוא משתתף בו. לכל משתתף ב-Session מסוים יש מספר מזהה שונה, על מנת להבדיל בין מקורות שונים. כל משתמש יכול להיות חלק מכמה Sessions בו זמנית. לרוב, בתקשורת מולטימדיה כל סוג מידע בתקשורת

מועבר ב-RTP Session נפרד ומשתמש בחבילות מידע RTCP משלו. למשל, בשיחות וידאו, לרוב יפתח Session לשידור הקול ו-Session אחר לשידור הוידאו.

- **Mixer** - היא מערכת שמקבלת חבילות מידע RTP ממקור אחד או יותר, כלומר זרמי מידע, ומאחדת את חבילות מידע אלו לחבילת מידע של RTP אחת, כלומר לזרם מידע אחד, ושולחת אותה. כמו כן, ייתכן שינוי בשיטת קידוד המידע. למשל, קידוד המידע באופן חסכוני לשידור על קווים בעלי קיבולת נמוכה, או איחוד של מספר תמונות של משתתפים בשיחה לכדי תמונה אחת ובכך לחסוך במידע כפול. ה-Mixer מציב מספרי חבילות מידע וזמנים חדשים בכל חבילות מידע ה-RTP החדשות שנוצרות. כלומר, ל"תחנת" Mixer יש SSRC (קיצור של Synchronization source Identifier) משלה המוכנס לשדה המתאים בחבילות מידע ה-RTP שהיא יוצרת (משפט זה יתבהר בהמשך המאמר).

- **Translator** - זוהי מערכת תיווך שמטרתה להעביר חבילות מידע RTP ליעדן מבלי לשנות את שדה ה-SSRC של חבילות המידע. למשל, השינוי בחבילות מידע יכול להיות בפורמט המידע. דוגמה לשימוש ברכיב Translator: נניח שתחנה A שולחת חבילות מידע RTP לתחנה B. חבילות מידע ה-RTP מכילות מידע וידאו בפורמט 2-MPEG (לינק לקריאה נוספת). על מנת לשדר באיכות הנדרשת בפורמט זה דרוש רוחב פס גדול יותר ממה שהקו בין A ל-B מסוגל לספק. נניח שבין A ל-T (שהיא תחנת ה-Translator) ישנו רוחב פס מספיק לשידור וידאו בפורמט 2-MPEG. במקרה זה A יכולה לשדר את המידע ל-T, ו-T תמיר את המידע לפורמט MPEG-4 (לינק לקריאה נוספת), שהוא חסכוני הרבה יותר, ותשלח את המידע ל-B. נשים לב שמאחר ו-T לא שינתה את שדה ה-SSRC בחבילות ה-RTP שקיבלה מ-A, היא בעצם "שקופה" מבחינתה של תחנת B, כלומר B "חושבת" שהמידע הגיע ישירות מתחנה A.

להלן שרטוט המציג את הדוגמה לעיל:

- **End System** - החלק באפליקציה שתפקידו לייצר את המידע שבחבילות מידע ה-RTP, ובכיוון השני לצרוך את המידע בחבילות מידע ה-RTP שהגיעו.

הסבר כללי

פרוטוקול RTP פועל בעיקרון קצה לקצה ומזרים מדיה בזמן אמת. הפרוטוקול מספק אמצעים כדי לפצות על ריצוד וגילוי של חבילות מידע אבודות ולסדר חבילות מידע שהגיעו בסדר שונה משליחתן[3]. פרוטוקול RTP מאפשר העברת מידע למספר יעדים שונים דרך שידורי IP ותומך גם ב-IPv4 וגם ב-IPv6. פרוטוקול RTP נחשב לתקן הראשי והעיקרי להעברת שמע וידאו ברשתות IP, על פי RFC 3550.

תוכנות התומכות בהזרמת מולטימדיה צריכות לבצע שיקולים וחישובים למען רצף ההזרמה למשתמש, לכן, כדי שחבילות מידע המתווספות כל הזמן יגיעו בזמן המדויק למקום הנכון, אחרות עלולות ללכת לאיבוד. אובדן זה עלול ליצור קטיעות למשך שברירי שניות, אך ניתן לתקן זאת באמצעות אלגוריתם אשר נמצא תחת הקטגוריה "Error Concealment" (ראו דוגמה לאלגוריתם כנ"ל בשם LNCS 2849).

RTP ניתן למימוש גם באמצעות פרוטוקול TCP, אך זה לא נפוץ מכיוון שבאפליקציות הבנויות על TCP, יש עדיפות גבוהה של אמינות לעומת זמן, מה שעלול לשבש או לעוות את המידע (כלומר, המידע אשר יוצג למשתמש יהיה לא רציף, ולכן יהיו עיוותים ושיבושים במידע הכולל המועבר), לכן רוב היישומים המשתמשים בפרוטוקול RTP, בנויים על ה-UDP.

RTP פותח על ידי קבוצת הוידאו/שמע של (Internet Engineering Task Force) IETF, ה-RTP ממומש בשיתוף עם פרוטוקולים נוספים כמו ה-H.323[4] ו-RTSP[5]. פרוטוקול RTP עובד בשיתוף פרוטוקול נוסף, RTCP, אשר הוא החלק האחראי על בקרה, שתפקידיו לשלוח פקודות או לספק מידע על המשתתפים בשיחה, ועל השליטה הכללית.

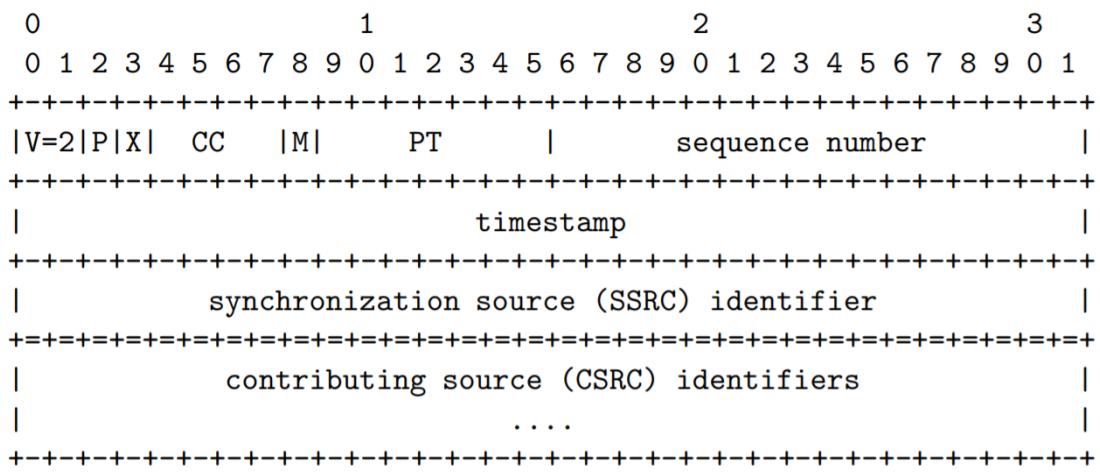
פרוטוקול ה-**RTP** מכיל נתונים בזמן אמת ולכן המידע המסופק על ידי פרוטוקול זה עובר תהליך של החתמה (Time-stamping), בה מצויין הזמן המדויק על מנת ליצור סנכרון מקסימלי. בדומה לחבילות מידע אחרות, גם חבילות מידע ה-RTP הן בעלות מספר רצף (Sequence Numbers), שנועד לזיהוי חבילות מידע אבודות, סידור וארגון מחדש של כל חבילות המידע ביחד כאשר הן מגיעות ליעד. כמו כן, חבילות מידע אלו הן בעלות Payload Format ("פורמט מטענים"), אשר מציין את הפורמט המקודד של הנתונים.

פרוטוקול ה-**RTCP** (קיצור של **RTP Control Protocol**, RFC 3605) אחראי לאיכות השירות, למשוב ולסינכרון בין זרמי המדיה השונים. יתר על כן, רוחב הפס של ה-RTCP מהווה (בדרך כלל) כחמישה אחוזים[6] בפרוטוקול לעומת ה-RTP.

---

[3] דבר חשוב במיוחד בקרב העברות UDP על רשתותIP
[4] לקריאה נוספת: https://tools.ietf.org/pdf/rfc3508.pdf
[5] לקריאה נוספת: https://tools.ietf.org/pdf/rfc7826.pdf
[6] לקוח מ: https://en.wikipedia.org/wiki/RTP_Control_Protocol#Protocol_functions

מבנה חבילות המידע



[השרטוט לקוח מ-https://tools.ietf.org/pdf/rfc3550.pdf]

כפי שניתן לראות, בטבלה הנ"ל המחשה טובה יותר של החלקים בכל חבילת מידע. כעת, נסביר על כל חלק וחלק וכיצד הוא משתלב בתפקיד של הפרוטוקול:

- **Version** - מציין באיזו גרסת RTP מדובר.
- **P (שמקורה במילה Padding - ריפוד)** - אם הביט P דולק, חבילת ה-RTP מכילה מידע נוסף בסוף חבילת המידע שאינו חלק ממנה. הבית האחרון במידע הנוסף, מכיל את מספר הבתים של המידע הנוסף, כולל עצמו. שימוש אפשרי ל-Padding הוא לטובת אלגוריתמים להצפנה שדורשים שגודל המידע יהיה כפולה של קבוע מסוים כלשהו (למשל 128 ביטים).
- **Extension - (הרחבה) X** - אם ערכו של הביט X הוא 1, לחבילת המידע יתווסף בלוק נוסף, בלוק ההרחבה, הסיומת, עליו נדון בהמשך.
- **CC** - או בשמו המלא, CSRC Count, מצביע על גודל המערך שנמצא בשדה ה-CSRC (עליו נדון בהמשך), כלומר, מספר ה-CSRC-ים שיופיעו בהמשך חבילת המידע.
- **M** - או בשמו המלא, Marker, הוא הביט שמסמן לאפליקציה על אירוע משמעותי כלשהו שמציינת חבילת המידע בה הוא דולק. כלומר, הוא דגל המסמן חשיבות על המידע המועבר, אם הוא דולק, פירוש הדבר שלמידע זה יש עדיפות מיוחדת ברמת האפליקציה (בדומה לשדה ה-TOS בחבילת IP). יהיה יותר נוח להבין זאת בעזרת דוגמה, למשל, דגל זה דולק כאשר יש תחילה או חזרה אל דיבור על ידי משתתף בשיחת אינטרנט.
- **PT (Payload Type) - סוג המידע** - שדה זה מיועד לזיהוי הפורמט של המידע/השמע שיועבר בהמשך. למשל, עקב בעיות בתקשורת מחליטה אפליקציה כלשהי ששולחת אודיו לשנות את הקידוד לפורמט חסכוני יותר על חשבון איכות. מקבל המידע יתעלם מחבילות מידע RTP שסוג המידע המופיע בשדה זה לא מוכר לו.

- **Sequence_Number** - מכיל מספר סידורי של חבילת RTP, כלומר מייצג את המיקום היחסי של חבילת המידע ב-Session נתון. המספר עולה ב-1 בכל חבילת מידע שנשלחת, ומאפשרת לשמור על סידור חבילות מידע (זה נכון שגם ב-TCP קיים Sequence Number אך מאחר שלרוב הפרוטוקול רץ על UDP לא ניתן להסתמך על כך וחשוב שתהיה לו בקרה עצמית), סדר ניגון המידע שהתקבל ולזהות חבילות מידע שאבדו. המספר שבו מתחילה הספירה של חבילות המידע נבחר באופן אקראי וכך תורם לסיום והפרדת השיחה באופן מלא, לא יוכל להיווצר בלבול עם שיחה חדשה כי היא תתחיל ממספר שונה לגמרי.

- **Timestamp** - בדומה ל-Sequence Number, הערך הראשוני של שדה זה נבחר באופן אקראי ובהמשך הערך עולה ביחס אליו. הערך שנמצא בחלק זה הוא הזמן בו נוצר הבית הראשון בחבילת המידע (כמובן בתוספת הערך האקראי שנבחר לו מלכתחילה), כך שהזמן צריך להיות מדויק ומסונכרן עם ה"זמן האמיתי", האפליקציה בוחרת זמן מדויק שעובר בין כל יחידה ליחידה. למשל, אפשר להגדיר שכל יחידה בשדה היא ms10 (עשר מילי שניות), ולכן המספר המצוין בשדה הוא הזמן שעבר מתחילת הקשר (חבילת המידע הראשונה) ועד לבית הראשון ב-Sequence.

- באותה האפליקציה, ניתן להגדיר זמן שונה המציין עלייה של השדה באחד, אך, צריך לבדוק שניתן לבסוף לסנכרן את השניים ולתאם ביניהם. ישנו שימוש בשדה זה למרות שקיים כבר שדה דומה, ה-Sequence Number, על מנת לדעת מתי חבילת המידע נשלחה ביחס לתחילת השיחה, ולאו דווקא לדעת את סדר השליחה שלה ביחס לחבילות המידע האחרות.

- **SSRC מקור סנכרון - (Synchronization Source)** - השדה המזהה של יוצר חבילת המידע. מספר זה הוא אחד ויחיד בכל חבילת מידע (כלומר, לא מופיעים מספרים של שני session-ים), השדה נבחר באקראיות ומשתמש באלגוריתם שבודק שאין כפילות מזהים בין משתתפי השיחה. גודל השדה הוא 32 ביטים, ובעזרתו יודעים רכיבי הקצה להבדיל בין session-ים שונים.

- **CSRC (Contributing source) - מקור תורם)** - זהו מערך בגודל בין 0 עד 15 ביטים, אשר מכיל רשימות של SSRC שנוצרו. במקרה בו נדרש שילוב של מספר מקורות שמע (למשל בשיחת ועידה), שדה זה יכיל את רשימת מזהי ה-SSRC של כל המקורות (שכמותם בהתאם לשדה ה-CC), וכך יאפשר את שילוב השמע בצורה מסונכרנת.

- **Extension Header**[7] - בפרוטוקול RTP, מאופשרות הרחבות ייעודיות שונות. כמו שכתבנו קודם, כאשר הדגל X דולק, פירוש הדבר שנעשה שימוש בהרחבה כלשהי, ובשדה זה מפורטים כל פרטי ההרחבה (רכיבי קצה שלא מכירים הרחבה מסוימת יתעלמו ממנה). ההרחבה מיועדת להעברת מידע שאיננו תקני ושאפליקציות משתמשות בו לצורך ניסוי/בדיקה, תוך שמירת תאימות עם תקן RTP. בלוק ההרחבה יתווסף מיד אחרי השדה SSRC או CSRC (אם קיים), והוא מכיל 3 שדות: שדה בגודל 16 ביט

[7] לקריאה נוספת: https://tools.ietf.org/pdf/rfc8285.pdf

לשימוש האפליקציה המנסה (למשל, ציון סוגים שונים של הרחבות), שדה Length בגודל 16 ביט שמכיל את גודל שדה ה-Extension (ללא השדה הקודם ושדה ה-Length עצמו), ושדה ה-Extension המכיל את המידע ה"ניסיוני".

- CSRC identifiers – שדה נוסף בפרוטוקול, שאינו חייב להופיע בחבילה.

מימושים לפרוטוקול

בשנים האחרונות השימוש בפרוטוקול זה גדל ומשתלב בנושאים רבים בחיים שלנו, כפי שציינו בהתחלה, יישומים רבים משתמשים בשיחות והעברת מידע בזמן אמת דרך האינטרנט (WhatsApp, Skype ועוד...).

פרוטוקול RTP משומש באופן נרחב במערכות מדיה ובידור, הכוללות Streaming (הזרמת מדיה), כגון טלפוניה, יישומי ועידות וידאו, שירותי טלוויזיה ותוכנות לחץ-כדי-לדבר (push-to-talk) המבוססות על האינטרנט.

נוסף על כך, RTP הינו אחד היסודות הטכניים של Voice over Internet Protocol (ובקיצור, VoIP). זהו שם כללי לטכנולוגיות ולשיטות שונות להעברת דיבור על גבי רשתות IP. טכנולוגיות אלו מהוות בסיס למגוון רחב של יישומים (לא בהכרח RTP, אלא טכנולוגיות מגוונות תחת השם VoIP):

- שיחות דיבור בתוכנות למסרים מיידיים (כדוגמת ICQ ו-Skype).
- שיחות טלפון על גבי האינטרנט באמצעות מכשירי טלפון ייעודיים.
- שיחות ממחשבים אל קווי טלפון "רגילים".
- שיחות בין שני קווי טלפון "רגילים" אשר חברת הטלפון בוחרת להעבירן על גבי רשתות IP.

טכנולוגיות ה-VoIP מעבירות דיבור על גבי רשת הנתונים בצורה דיגיטלית. הדיבור האנושי הוא אנלוגי במקורו, ועל מנת להעבירו על גבי רשתות נתונים יש צורך לקודד אותו בצורה דיגיטלית. המקודד (Voice Coder) הוא לב-ליבה של טכנולוגיית ה-VoIP. באמצעות קידוד מתאים של אותות הדיבור (ישנה דרך נוספת לעבד את המידע האנלוגי, בעזרת תדר נייקוויסט, אשר ניתן לקרוא עליו כאן), ניתן להתגבר על תכונות רשת ה-IP[8] ולהצליח להעביר עליה דיבור.

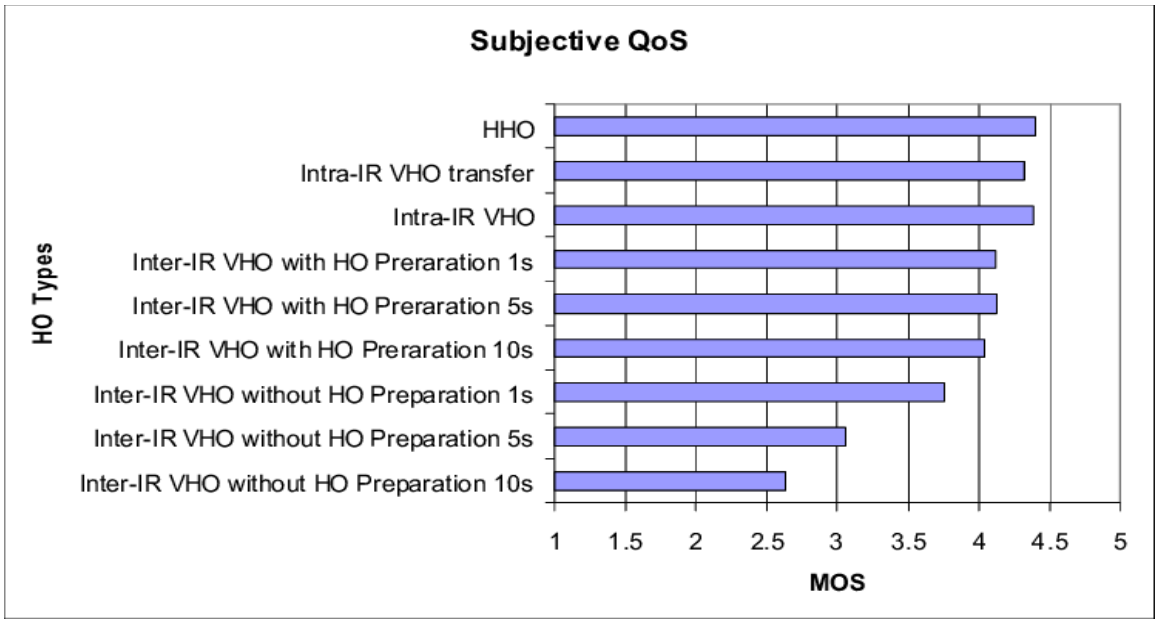
המקודד דוחס את השמע לקצבים נמוכים בהרבה מבטלפוניה רגילה, ובכך חוסך רוחב סרט רב. חלק מרוחב הסרט שנחסך משמש להעברת אותות בקרה להתמודדות עם התכונות הנ"ל (כמו למשל שידור מחדש של חבילות, מספור כל חבילה במספר סידורי, הוספת קודים לתיקון שגיאות ועוד). יתר על כן,

[8] התכונות הבעייתיות של רשת ה-IP: רוחב הסרט הזמין לשימוש מושפע מהעומס הרגעי על הרשת וממספר המשתמשים והיישומים הצורכים את משאבי הרשת בכל רגע ורגע, ולכן גם יכול להשתנות בכל רגע ורגע. בנוסף, המידע ברשתות נתונים לא נשלח בצורה רציפה אלא בחבילות מידע, אשר כל אחת מכילה נתח מסוים מתוך כלל המידע, והחבילה עוברת דרך מסלול שונה ברשת על פי העומס הקיים בכל רגע ורגע בנקודות שונות ברשת

המקודד מעביר רק דיבור אנושי, מסנן צלילים מיותרים ורעשי רקע ונמנע מהעברת השקט שבין המשפטים, המילים והאותיות. העמקה בנושא של טלפוניית IP ניתן למצוא כאן, מאמר מתוך גליון 20.

מושגים חשובים הקשורים ל-VoIP:

- **Jitter** - זהו מצב בו חבילות מידע מסוימות הגיעו לנמען בסדר שונה, או "נאבדו", מה שמוביל לשיחה פגומה ומעורפלת. עקב כך ש-VoIP מסתמך על חיבור האינטרנט, ה-Jitter ייגרם כתוצאה מחיבור אינטרנט לא יציב או תקלות רשת למיניהן.
- **Delay/Latency** - העיכוב בין מתן ההוראה להעברת נתונים לבין העברת הנתונים עצמם. השוואה פשוטה לעולם הגיימינג, עיכוב זה הוא מרווח הזמן בין פעולת השחקן לביצועה. התרחשות עיכוב כזה מכונה "Lag".
- **Mean Opinion Score** - בדומה לשמו, ה-MOS הוא ציון בין 1.0 ל-5.0 שמקבל חיבור בין מכשירים המתבסס על VoIP. באמצעות ניקוד/ציון זה ניתן למדוד את איכות השיחה את הביצועים, את השינויים והבעיות הלא רצויות שעלולות להיגרם. ה-MOS מושפע מחומרה, רוחב פס, Jitter, Latency, איבוד פקטות ועוד...
- **Quality of Service** - איכות השירות, או בקיצור QoS, הוא התיאור או המדידה של הביצועים הכוללים של השירות, ובמיוחד כיצד הביצועים נראים בעיני משתמשי השירות. על מנת למדוד את ה-QoS, ישנם מספר היבטים שלרוב נכללים בחישוב, כמו למשל אובדן חבילות מידע, עיכוב השידור (Latency), חשיבות ותעדוף חבילת המידע, זמינות ו-Jitter.



[לקוח מהאתר ResearchGate]

בטבלה הנ"ל דוגמאות למקודדים נפוצים וציוני ה-MOS שלהם.

לאחר שהסברנו לכם קצת על טכנולוגיית ה-VoIP שקשורה רבות ל-RTP, נראה לכם דוגמה למימוש של הפרוטוקול RTP בשיחת ועידה:

קבוצת עבודה כלשהי נפגשת לדיון, באמצעות שירותי IP Multicast באינטרנט בשביל תקשורת קולית. באמצעות מנגנון הקצאה כלשהו, יושב ראש קבוצת העבודה מקבל כתובת קבוצת שידור (Multicast Group Address) וזוג פורטים. פורט אחד משמש לנתוני שמע, בעוד שהפורט השני משמש עבור חבילות מידע בקרה (מסוג RTCP). כתובת זו ופרטי הפורט מופצים לשאר המשתתפים בדיון (המשתתפים המיועדים). אם רצויה סודיות, הנתונים וחבילות מידע הבקרה עשויים להיות מוצפנים (נפרט על הצפנת RTP בהמשך), ובמקרה זה יש ליצור ולהפיץ גם מפתח הצפנה (הפרטים המדויקים של מנגנוני ההקצאה וההפצה ההלו הם מעבר להיקף הפעילות של RTP. ניתן לקרוא כאן). התוכנה בה משתמש כל משתתף בוועידה, שולחת נתוני שמע בחתיכות קטנות, למשל במשך של ms20 (עשרים אלפיות השנייה). לכל נתח של נתוני שמע קודמת RTP Header.

ה-RTP Header מציין איזה סוג קידוד אודיו (כגון PCM, ADPCM או LPC) נכלל בכל חבילת מידע כך שהשולחים יכולים לשנות את הקידוד במהלך הועידה, לדוגמה, על מנת להתאים משתתף חדש שמחובר דרך קישור ברוחב-פס נמוך או להגיב לאינדיקציות של עומס ברשת. האינטרנט, כמו חבילות מידע רשת אחרות, מאבד מדי פעם חבילות מידע, משחזר אותן ומעכב אותן על פי כמויות זמן משתנות. בשביל להתמודד עם ליקויים אלו, ה-RTP Header מכיל מידע ומספר רצף (Sequence Number) המאפשר לקולטי חבילות המידע לשחזר את התזמון המיוצר על ידי המקור, כך שבדוגמה זו, נתחי שמע מושמעים ברציפות כל ms20 (שחזור תזמון זה מבוצע בנפרד עבור של מקור של חבילות מידע RTP בוועידה). קולט חבילת המידע יכול גם להשתמש במספר הרצף על מנת להעריך כמה חבילות מידע "הולכות לאיבוד".

מאז שחברי קבוצת העבודה הצטרפו ועזבו את הוועידה במשך הדיון, כדאי לדעת מי ממשתף בה בכל רגע נתון וכמה טוב הם מקבלים את נתוני השמע. לשם כך, כל מופע של יישום שמע בוועידה מקבל דוח קבלה בתוספת השם של המשתמש שלו על פורט ה-RTCP.

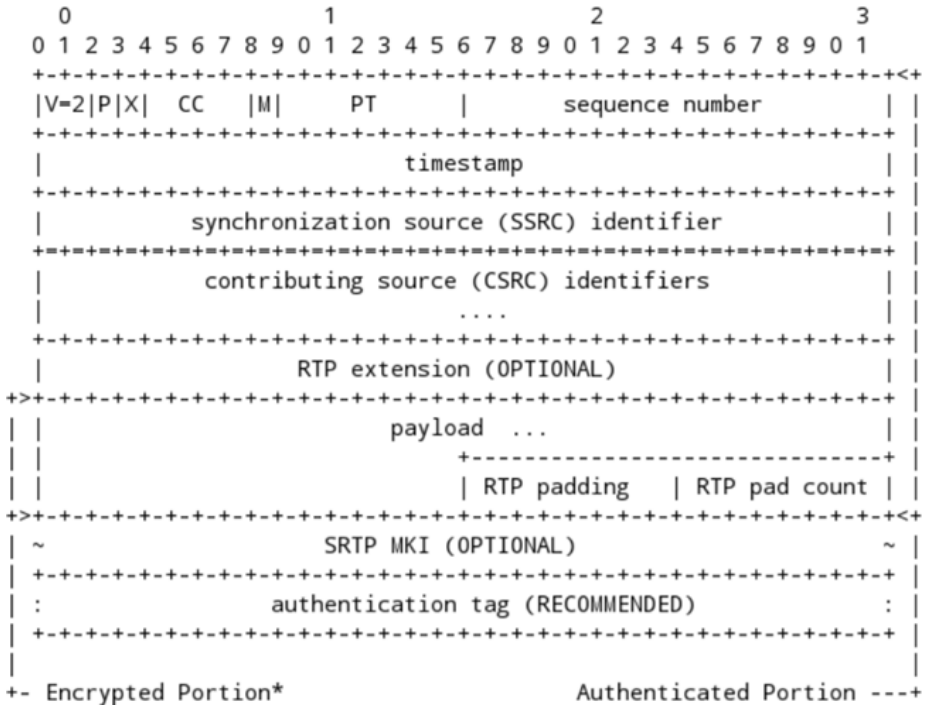
דוח הקבלה מציין עד כמה טוב הדובר הנוכחי מושמע ומתקבל על ידי המופע הספציפי. בנוסף לשם המשתמש, מידע מזהה אחר עשוי גם להיכלל בכפוף לבקרת מגבלות רוחב הפס. המופע שולח חבילת מידע מסוג RTCP BYE (שהיא הודעת סיום תקשורת של המשתמש הנשלחת על ידיו כאשר הוא עוזב את ה-Session) כאשר הוא עוזב את הוועידה.

הצפנה

ישנן שיטות מגוונות להצפין את המידע המועבר בעזרת RTP, בפרק זה נציג חלק מהן.

SRTP

אנו נציג בפניכם כעת את פרוטוקול SRTP (ראשית תיבות של Secure Real-Time Protocol), שהיא שיטה נפוצה מאוד להצפנת המידע המועבר ב-RTP. הרעיון הוא לבצע הגנה לשידור המדיה במצבים של Unicast (כלומר שידור לצד אחד), או Multicast (כלומר שידור למספר גורמים ידועים במקביל, כדוגמת שיחת וועידה). הרעיון של SRTP הינו הצפנה בשני חלקים; חלק אחד הוא הצפנה של חבילת המידע עצמה, והחלק השני הוא הצפנה של חלקי הזרימה. הצפנת חבילת המידע עצמה היא סוג של "Block Cipher" (ובעברית: צופן בלוקים), אשר למעשה לוקחת את כל חבילת המידע, מחלקת לבלוקים, עושה Padding ומצפינה אותה. הצפנת חלקי זרימה, הוא הצפנת המידע (ה-payload) עצמו.



מבנה חבילת המידע מתואר בתרשים כאן למעלה (הלקוח מן RFC 3711), וניתן לראות כי הצפנת ה-block מתאורת בצד ימין למטה, והצפנת המידע (ה-payload) עצמו מתוארת בצד שמאל למטה.

בנוסף ל-SRTP, גם RTCP מכיל פרוטוקול הצפנה ששמו הוא SRTCP. הבעיה היא שלא חובה לבחור בשימוש של SRTP ושל SRTCP. כלומר, גם אם אחד מהפרוטוקולים הללו נכנס להצפנה, זה לא אומר שגם הפרוטוקול השני מוצפן. הפרוטוקול אפילו מאפשר להשתמש בהצפנה בשם "null cipher", שהיא למעשה רק מתנהגת כאילו ישנה הצפנה, אך המידע אינו באמת מוצפן. כאשר רוצים מידע מוצפן כמו שצריך, חייבים לקחת בחשבון את כל הנקודות האלו ולדאוג להן לפני שניתן לדעת שיש הצפנה בפועל של המדיה.

zRTP

זוהי גישת הצפנה מבוססת על עקרון Diffie-Hellman (פרוטוקול דיפי-הלמן) - פרוטוקול המאפשר לשני משתתפים שלא נפגשו מעולם ואינם חולקים ביניהם סוד משותף כלשהו מראש, להעביר אחד לשני מעל גבי ערוץ פתוח (שאינו מאובטח) סוד כלשהו כך שאיש מלבדם אינו יודע מהו (לקוח מתוך ויקיפדיה).

בהצפנה זו יש החלפת מפתחות הצפנה ושימוש ב-SRTP להצפנה עצמה. האות z מייצגת את אחד המתכננים העיקריים של הפרוטוקול בשם פיל צימרמן (Phil Zimmermann, שבנוסף יצר מיזם לפיתוח תוכנה המאפשר ביצוע הצפנה מקצה לקצה של מסרים בצורה קלה ופשוטה, שבעברית נקרא פרטיות טובה למדי, ובאנגלית Pretty Good Privacy - PGP. לקריאה נוספת:
https://en.wikipedia.org/wiki/Pretty_Good_Privacy)

zRTP עובד רק במצב של אחד לאחד, כלומר Unicast, ואינו מתוכנן לעבוד עם מצב של שיחות בין קבוצות. אך המימוש חייב לתמוך ברעיון של Multiplexing - רעיון בו ניתן להעביר יותר ממידע בודד באותו הצינור בדיוק, אותו רעיון של "קו טלפון" מסורתי, אשר כבל אחד יכול "להחזיק" מספר שיחות במקביל.

כמו כן, הפרוטוקול מנסה לבדוק האם הצד השני מסוגל לענות בפרוטוקול, ובמידה וכן, מצפין את המידע, ובמידה ולא, המידע עובר בצורה לא מוצפנת. גישה זו קיבלה את השם "הצפנה מזדמנת" (Opportunistic Encryption).

אחת הדרכים ש-zRTP (אגב, פיל גם יצר את התוכנה Zfone, שהיא מאפשרת להשתמש בשיחות קוליות על גבי האינטרנט בעזרת VoIP, והיא גם משתמשת בהצפנה אסימטרית) מבצע זאת, היא על ידי ביצוע סוג של ping, אשר כל מי שתומך ב-zRTP חייב לענות עליו. במידה ואין מענה, הוא יודע כי לא ניתן "להיכנס" למצב הצפנה.

למרות השימוש ב-Hellman-Diffie, אין צורך בהכרזה מוקדמת על מפתח ההצפנה בפרוטוקול, היות והמפתחות נוצרים בזמן ריצה בין שני הצדדים. גישה זו מאפשרת להגן מפני התקפת MiTM (התקפת אדם בתווך) במידה והמחשב שקיבל את הבקשה הראשונה אינו שייך לתוקף, ובכך קשה מאוד "לזהם" את ההצפנה כאשר זו התרחשה בין שני צדדים.

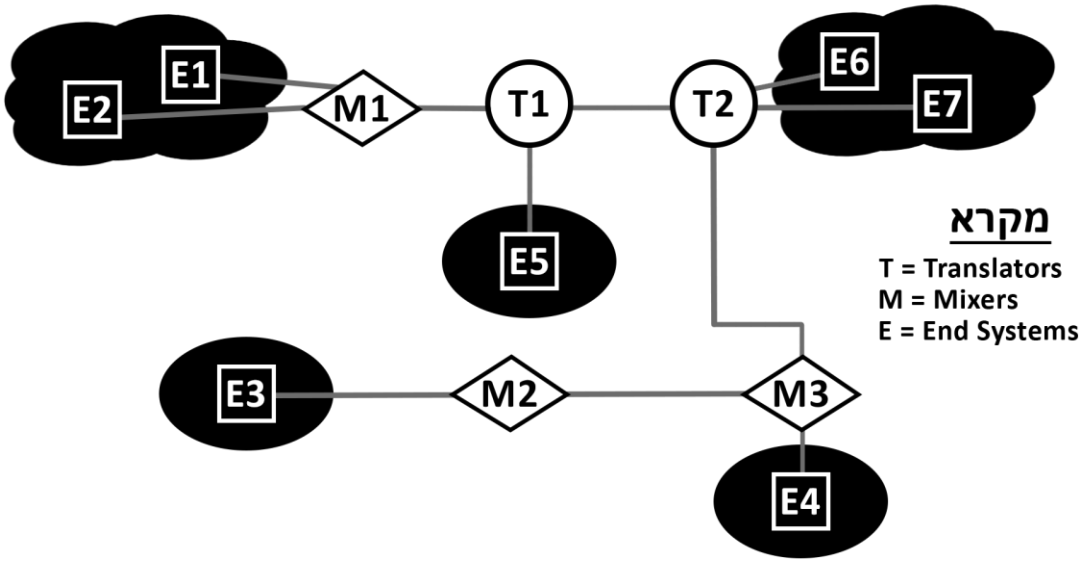
בקשות zRTP מכילות פקודות שונות ולמעשה מכילות תת פרוטוקול תקשורת, כאשר כל תת פקודה שכזו מתועדת ב-RFC ומכילה מבנה משל עצמה. לכן הפרוטוקול נחשב ל"בזבזני" יותר, אך מצד שני מספק יכולות רבות יותר, והגנה טובה יותר, והצפנת המדיה עצמה עדיין מתבצע עם SRTP, אשר zRTP מתערב בו.

להמשך קריאה ולהעמקה נוספת בנושא ההצפנה ל-RTP בפרט ול-VoIP בכלל ניתן לקרוא כאן

רכיבים אלו משמשים להעברה של מידע ב-RTP, עם שינויים אפשריים בו. הם מקשרים בין "ענים" שונים ברמת ה-RTP. למשל, "ענן" יוגדר כקבוצת משתמשים המשתמשת באותה ארכיטקטורת רשת ברמות ה-Network ומעלה, באותה כתובת רשת ובאותו פורט. לדוגמה: IP, UDP, Multicast Address, Port Number או IP, UDP, 2 unicast addresses, 2 Port Number.

ישנם מספר שימושים אפשריים ב-Mixers וב-Translators, אשר כוללים הסרת או הוספת הצפנה, שינוי הקידוד של המידע ושינוי כתובת היעד מ-Multicast למספר כתובות Unicast. ההבדל העיקרי בין שני הרכיבים הוא שה-Translator מעביר זרמי מידע של מקורות בנפרד, ללא איחוד ביניהם, ואילו ה-Mixer מאחד מספר זרמי מידע לכדי זרם מידע אחד.

כעת, נציג בפניכם דוגמה למבנה אפשרי של רשת RTP:



[השרטוט נלקח מ: http://math.haifa.ac.il/oran/adv/RTP.pdf]

ה-Mixer וה-Translator הינם רכיבים שמעברים חבילות מידע RTP. כדי למנוע לולאות בהעברה, כל "ענן" צריך להיות שונה מהשני בלפחות אחד מן הפרמטרים שמגדירים ענן.

כמו כן, Mixers ו-Transaltors לא יחברו במקביל, אלא אם כן יש חלוקה ביניהם של אילו SSRC כל אחד מעביר.

כל ה-SSRC-ים של מקורות באותו RTP Session שחבילות המידע שלהם מועברים על ידי Mixers/Transalotrs צריכים להיות שונים האחד מהשני בכל ה"ענים".

Translator - תחנת Translator מיועדת להעברת חבילות מידע RTP ללא שינוי שדה ה-SSRC שלהן. לפיכך, ה-Translator בעצם שקוף מבחינת הנמענים של חבילות המידע.

לרוב, משמש ה-Translator כאמצעי לשינוי סוג קידוד המידע בחבילות מידע RTP או למטרת איחוד בלוקים של Payload ממספר חבילות מידע שהגיעו מאותו שולח. במקרים אלו, חייב ה-Translator לעדכן את שדות ה-Timestamp וה-Sequence Number של חבילות המידע שעוברות דרכו, כך שיתאימו לחבילות מידע החדשות שייצר. במקרה ששונה הקידוד, יעדכן ה-Translator את שדה ה-Payload Type.

שימוש נפוץ נוסף של Translator הוא תחנת פענוח/הצפנה של מידע מוצפן/לא מוצפן שעובר דרכו.

Mixer - תחנת Mixer מקבלת חבילות מידע מידע ממקור אחד או יותר ומאחדת את זרמי המידע השונים לכדי זרם מידע אחד.

כאשר תחנת Mixer יוצרת חבילת מידע של RTP חדשה, היא דואגת לסנכרון נכון של המידע ממנו מורכבת חבילת המידע. היא עושה זאת בהסתמך על שדות ה-Timestamp וה-Sequence Number של חבילות המידע מהן מורכבת חבילת המידע החדשה.

מאחר ותחנת Mixer שולחת חבילת מידע עם מזהה SSRC משלה, היא מוסיפה לחבילות מידע ה-RTP שהיא יוצר שדה CSRC שמכיל את ערכי ה-SSRC-ים של כל המקורות מהם מורכבת חבילת המידע החדשה.

RTCP - הסבר כללי

מנגנון ה-RTCP מבוסס על שליחה מחזורית, אבל בפרקי זמן העשויים להשתנות, של חבילות מידע RTCP בין כל המשתתפים ב-Session, ובשימוש של אותו מנגנון הפצה כמו של חבילות מידע המידע (חבילות מידע RTP). למשל, שימוש באותו עץ הפצה של multicast.

RTCP מבצע ארבעה תפקידים עיקריים:

התפקיד העיקרי הוא לספק משוב בנוגע לאיכות הפצת המידע. המשוב יכול לספק מידע שיהיה יעיל לקביעת מאפייני הקידוד של המידע, ולבדיקה האם מנגנון הפצת המידע תקין. למשל, אם יש הרבה אובדנים, יש להשתמש בקידוד פחות חסכוני אך רגיש פחות לשגיאות. אם עץ הפצה ב-Multicast אינו מגיע לכל היעדים, ניתן יהיה לגלות זאת.

בנוסף לכך, שליחת משוב אל כל המשתתפים מאפשרת למקבל מסוים לאבחן האם בעיה היא לוקאלית או גלובלית (כלומר, האם מקבלים נוספים סובלים מאותה בעיה).

בתור שימוש נוסף, יכולים ספקי שירותי תקשורת (שאינם מעורבים ב-Session של לקוחותיהם) לנצל את הודעות ה-RTCP כדי לזהות בעיות בתשתית הרשת שלהם.

תפקיד נוסף הינו מנגנון RTCP המספק מזהה נוסף, קבוע, לכל RTP Source שנקרא המאפיין הקנוני (CNAME - שהוא מזהה ברמת התעבורה עבור נקודות קצה של RTP, אשר אמור להישאר ללא שינוי גם כאשר מתגלה התנגשות כלשהי, כך שיהיה ניתן לזהות נקודות קצה של RTP באופן ייחודי. ראו גם: רשומת CNAME בפרוטוקול DNS). מאחר ומזהה ה-SSRC נבחר באקראיות, תיתכן (גם אם בסבירות נמוכה) כפילות שתאלץ שינוי של המספר המזהה. המאפיין הקנוני מאפשר למקבל המידע להמשיך ולעקוב אחר המידע שנשלח על ידי שולח שערך ה-SSRC שלו השתנה.

כמו כן, מאפשר לשלב סוגי מידע שונים מאותו שולח המגיעים ב-RTP Sessions שונים, למשל אודיו ווידאו.
כמו כן, מאחר וכל משתתף שולח הודעות RTCP לכל שאר המשתתפים, יכול כל משתתף לדעת כמה משתתפים יש ב-Session ובעזרת מידע זה להתאים את הקצב שבו הוא שולח חבילות RTCP.

יתר על כן, ה-RTCP הינו שירות אופציונאלי: להעביר מידע בקרה מינימלי עבור השיחה, למשל השם של משתמש,לצורך הצגתו על המסך של המשתמשים האחרים. יש לכך יתרון בשיחות שבהן משתתפים יכולים להצטרף ולעזוב שיחות ללא הרשמה מוקדמת.

סוגי חבילות מידע RTCP[9]

חבילות מידע RTCP מחולקות למספר סוגים:

- **Sender Report (ובקיצור, SR):** דיווח מידע הקשור לשליחת מידע, ומידע סטטיסטי הקשור לקבלה של מידע. הודעות אלו נשלחות על ידי משתתפים המשמשים כשולחי מידע פעילים.
- **Receiver Report (ובקיצור, RR):** אלו הודעות המספקות סטטיסטיקה על קבלת מידע ממשתתפים שאינם שולחי מידע פעילים.
- **SDES (ראשי תיבות של Source Descriptions):** אלו הודעות הצגה של מקורות, זיהויים והעברת ה-CNAME שלהם.
- **BYE:** הודעות סיום תקשורת של משתמש. נשלחת על ידי משתמש שעוזב את ה-Session.
- **APP:** הודעות לשימוש ספציפי של אפליקציה שמשתמשת בפרוטוקול RTP.

חבילות מידע RTCP נשלחות בקבוצה של לפחות 2 חבילות מידע או יותר שיקראו "חבילות מידע מאוחדות". חבילת מידע מאוחדת מוכנסת לתוך אותה חבילת המידע ברמה נמוכה יותר, למשל לחבילת מידע של UDP. חבילת מידע מאוחדת מתאפיינת ב:

1. חבילת מידע של RTCP הראשונה בחבילת המידע המאוחדת חייבת להיות חבילת מידע למטרת דיווח מסוג SR או RR. אם עדיין לא נשלח או התקבל מידע בתחנה ששולחת את חבילת המידע, תישלח חבילת מידע של RR ריקה.
2. בכל משלוח של חבילת מידע מאוחדת תיכלל חבילת מידע של SDES המכילה את מזהה ה-CNAME של השולח שלה. זאת כדי שהמקבל ידע לשלב מהר מידע שהוא מקבל ב-RTP Sessions שונים מאותו משתמש.
3. במידה ומספר המקורות שעליהם נשלחות סטטיסטיקות של קבלת מידע עולה על 31, ישלח המידע הנוסף בחבילת המידע של RTCP הבאה בחבילת המידע המאוחדת.
4. חבילות מידע BYE או APP יופיעו אחרונות.

[9] להרחבה ניתן לקרוא ב-RFC 3550

ניתוח הפרוטוקול RTP באמצעות Wireshark

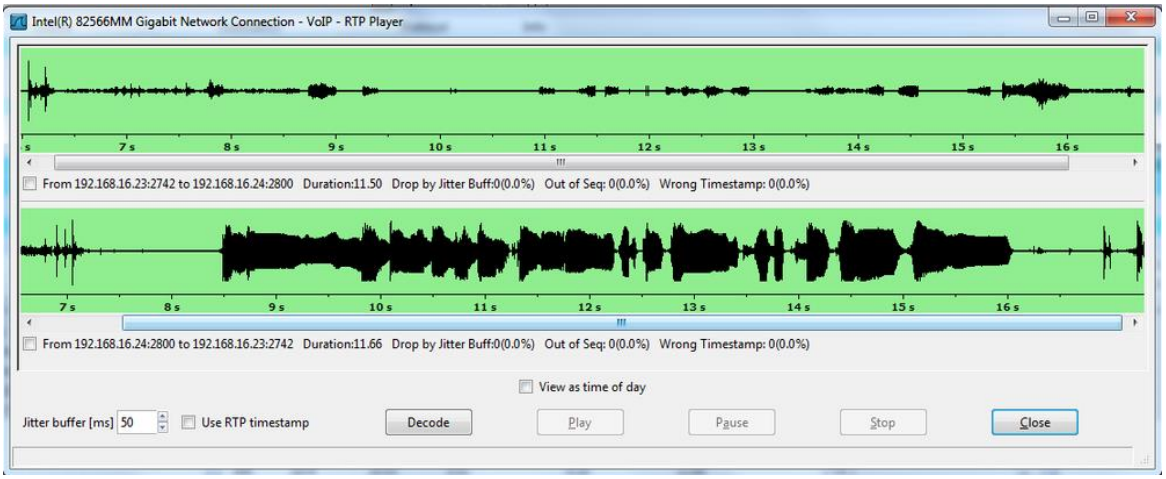
ראשית, Wireshark הינה תוכנת ניטור תעבורת תקשורת ברשת המקומית, אליה מחובר כרטיס הרשת המחובר למחשב (אמנם ניתוח תעבורת הרשת יכול להתבצע גם על פני קובץ PCAP ממחשב אחר). תוכנה זו משמשת בין היתר לאבחון בעיות, ניתוח תעבורת רשת, כלי עזר בתהליך הפיתוח של פרוטוקולי תוכנה ותקשורת, וכמובן למידה.

לפניכם תמונה של הסנפת חבילת מידע של RTP:

```
Internet Protocol Version 4, Src: 192.168.16.23 (192.168.16.23), Dst: 192.168.16.24
User Datagram Protocol, Src Port: tsb2 (2742), Dst Port: acc-raid (2800)
Real-Time Transport Protocol
⊞ [Stream setup by H245 (frame 597)]
  10.. .... = Version: RFC 1889 Version (2)
  ..0. .... = Padding: False
  ...0 .... = Extension: False
  .... 0000 = Contributing source identifiers count: 0
  0... .... = Marker: False
  Payload type: ITU-T G.711 PCMU (0)
  Sequence number: 11639
  [Extended sequence number: 77175]
  Timestamp: 998248329
  Synchronization Source identifier: 0x196d27c5 (426584005)
  Payload: cec4e14b60cb61f8684a70febfcd5f51494d70c1cdde3f4a...
```

[התמונה של ניתוח חבילת המידע נלקחה מכאן]

ה-Header המוצג הנ"ל מציג את הגרסה, האם קיים padding, האם קיימת הרחבה, מזהי מקור של חבילת המידע, האם דגל ה-Marker דולק, סוג המידע, מספר הרצף, חותמת זמן, סנכרון מזהה מקור, ולבסוף המידע עצמו, בהתאמה. הסברים על כל חלקי חבילת המידע הנ"ל מופיעים קודם לכן, תחת כותרת המשנה "מבנה חבילות המידע". כמו כן, התוכנה Wireshark מכילה בתוכה פיצ'ר מובנה הנקרא "VoIP calls" אשר מטרתו היא לפענח מידע של RTP לפורמט אודיו שניתן לניגון על ידי המשתמש.



[התמונה של השמע המוצג למשתמש נלקחה מכאן]

להסברים נוספים על התוכנה והכלי Wireshark ניתן לקרוא במאמר "עבודה עם Wireshark - חלק א'" שפורסם בגליון 75 במגזין זה.

סיכום

במאמר זה הכרנו מושגים רבים הקשורים לעולם התקשורת בכלל, ולפרוטוקול RTP בפרט, למדנו את בסיס הפרוטוקול ונגענו בנושאים רבים הקשורים לו. ראינו את מבנה חבילת המידע ולשם מה נחוץ כל חלק וחלק בו.

יתר על כן, ראינו את המימושים של הפרוטוקול כיום וכיצד השימוש בו גדל בשנים האחרונות, נגענו קצת בטכנולוגיות ה-VoIP ואפילו הראנו דוגמה לשיחת וועידה באמצעות RTP. כמו כן, דיברנו על שתי שיטות עיקריות להצפנת מידע העובר דרך פרוטוקול RTP, אחד הוא SRTP שהוא הנפוץ ביותר והשני הוא zRTP.

ראינו רכיבים המשמשים להעברת חבילות מידע של RTP: ה-Mixer וה-Translator, הבנו את ההבדל ביניהם ובנוסף לכך ראינו בעזרת תרשים דוגמה למבנה אפשרי של רשת RTP. הבנו מהו פרוטוקול RTCP, ראינו כמה שהוא חשוב ולמדנו מספר תפקידים עיקריים אותם הוא מבצע. ראינו 5 סוגים של חבילות מידע, והבנו את המושג "חבילות מידע מאוחדות" ואת המאפיינים שלו.

קצת עלינו

- **דביר גולן** - בן 15, גר בראשון לציון, לומד בתוכנית אודיסאה באוניברסיטת תל אביב במסלול סייבר בשנה א'. אוהב ללמוד כיצד תהליכים שונים במדעי המחשב קורים "מאחורי הקלעים", בין אם זה בחומרה או בתוכנה.
- **שחף גרגיר** - בן 15, גר בצורן, לומד בתוכנית אודיסאה באוניברסיטת תל אביב במסלול סייבר בשנה א'. אוהב לחקור דברים ולהבין כיצד הם עובדים באמת, לעומק, מה שלרוב לא מקבלים בבית ספר.

בחרנו לעסוק בפרוטוקול RTP עקב כך שהשימוש שלו כיום גדל והוא מתקדם בקצב רב בשנים האחרונות, היינו סקרנים לחקור עליו ועל מה שהוא עושה מאחורי הקלעים. המשימה הייתה לא פשוטה, ובמהלכה למדנו המון, החל מחיפוש מידע, הצגתו בצורה ידידותית ומעניינית, חלוקה וסיכום כאשר יש המון חומר, והרחבה ומחקר מעמיק כאשר יש מעט חומר.

המאמר נכתב תוך ליווי מצידו של שלומי בוטנרו כחלק מהפעילות והפרוייקטים בתכנית אודיסיאה

אנו שמחים שניתנה לנו האפשרות לכתוב את המאמר, דבר שהיה מאתגר ותרם לנו רבות.

קישור לאתר תכנית אודיסאה:

https://www.madaney.net/site/programs/odyssey

ריכוז חומרים

הסבר כללי על הפרוטוקול ועל האופן בו הוא מתבצע, מבנה חבילות המידע והמידע המועבר דרך הפרוטוקול:

https://www.rfc-editor.org/info/rfc4585

הסבר כללי על הפרוטוקול ועל האופן בו הוא מתבצע, מבנה חבילות המידע והמידע המועבר דרך הפרוטוקול:

https://www.rfc-editor.org/info/rfc3550

תיאור קצר והצפנה:

https://www.digitalwhisper.co.il/files/Zines/0x3C/DW60-2-VoIP.pdf

הצפנת העברת המידע, בקרה על המידע:

https://www.rfc-editor.org/info/rfc3711

מבנה חבילת המידע וקצת הסבר על מימושים:

https://en.wikipedia.org/wiki/Real-time_Transport_Protocol

הסבר וקצת מימושים (לא בהכרח מהימן)

https://www.youtube.com/watch?v=CttYdLpGeGA&index=3&list=PLOOa-TqFvRv_hyC1pxjHI86rhYMsQItks&t=0s

שימושים (לא בהכרח מהימן)

https://www.youtube.com/watch?v=Ll68ndJZW4g&list=PLOOa-TqFvRv_hyC1pxjHI86rhYMsQItks&index=17&t=0s

מבנה חבילת המידע:

https://www.youtube.com/watch?v=bHwTcoq7n7E&index=1&list=PLOOa-TqFvRv_hyC1pxjHI86rhYMsQItks

יתרונות וחסרונות:

https://answers.yahoo.com/question/index?qid=20090129044549AAbZQbI&guccounter=1

https://www.cs.columbia.edu/~hgs/rtp/faq.htmlb

יתוח תעבורת הפרוטוקול באמצעות Wireshark:

https://www.oreilly.com/library/view/packet-guide-to/9781449339661/ch04.html

הסבר כללי:

http://math.haifa.ac.il/oran/adv/RTP.pdf

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-110 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 אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין - Digital Whisper צרו קשר!

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

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

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא בסוף חודש ספטמבר 2019.

אפיק קסטיאל,

ניר אדר,

31.08.2019