digital whisper
Technologic Papers

אתגר השב"כ 2018

מאת Dvd848 ו-YaakovCohen88

הקדמה

בתחילת דצמבר 2018 שחרר השב"כ סדרת אתגרים כחלק מקמפיין גיוס עבור אנשי טכנולוגיה. במאמר זה נציג את הפתרון שלנו לשניים מתוך ארבעת המסלולים של סדרת האתגרים.

חשוב לנו להגיד: אנחנו אומנם לא מכירים את היוצרים, אך הצלחנו להשיג מהם אישור לפרסום הפתרונות לפני מועד סגירת האתגר. ללא אישורם - מסמך זה לא היה מתפרסם.

סיפור רקע

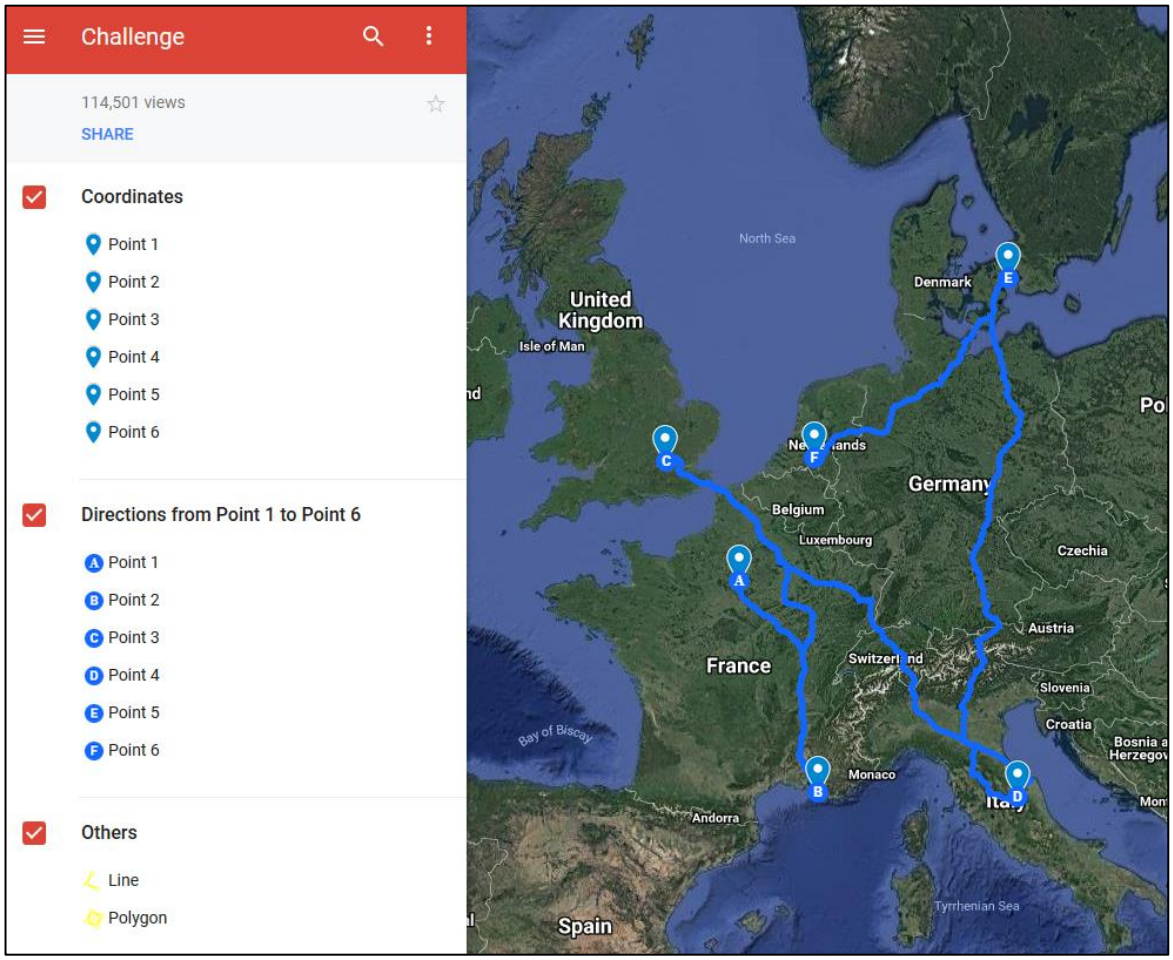
Hello Special agent A from the Israeli Security Agency (ISA) Technological Unit

"White September" (WS) is a group of arch-terrorists. They are connected to the global Jihadist movement, and are funded by Iran and Hezbollah. Several weeks ago, they used the darknet to declare their intentions of carrying out a mega terror attack in Israel. They nicknamed the operation "Israeli September 11th". These people are highly sophisticated and utterly merciless. We at the ISA have received a tip that some of the terrorists have already infiltrated the country. Our agents have launched an operation to halt them before they can carry out their plot.

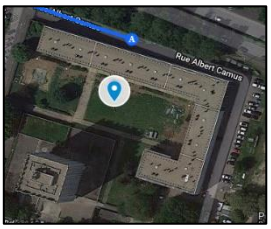
כניסה לאתגר

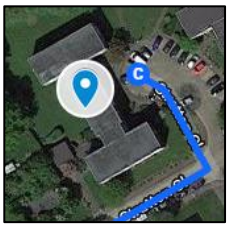
בדומה למה שראינו בשנים קודמות באתגרי המוסד, גם אתגר השב"כ הנוכחי נפתח עם תרגיל חימום שרק לאחריו ניתן להגיע אל רשימת האתגרים:

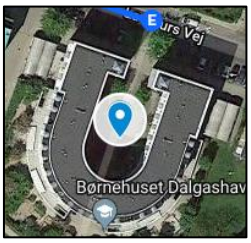
לחיצה על הלוגו מובילה אל מפה ב"גוגל מפות" שעליה שש נקודות ציון, ומסלול מהנקודה הראשונה אל האחרונה:

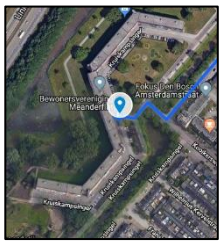


נבקר בנקודות השונות:



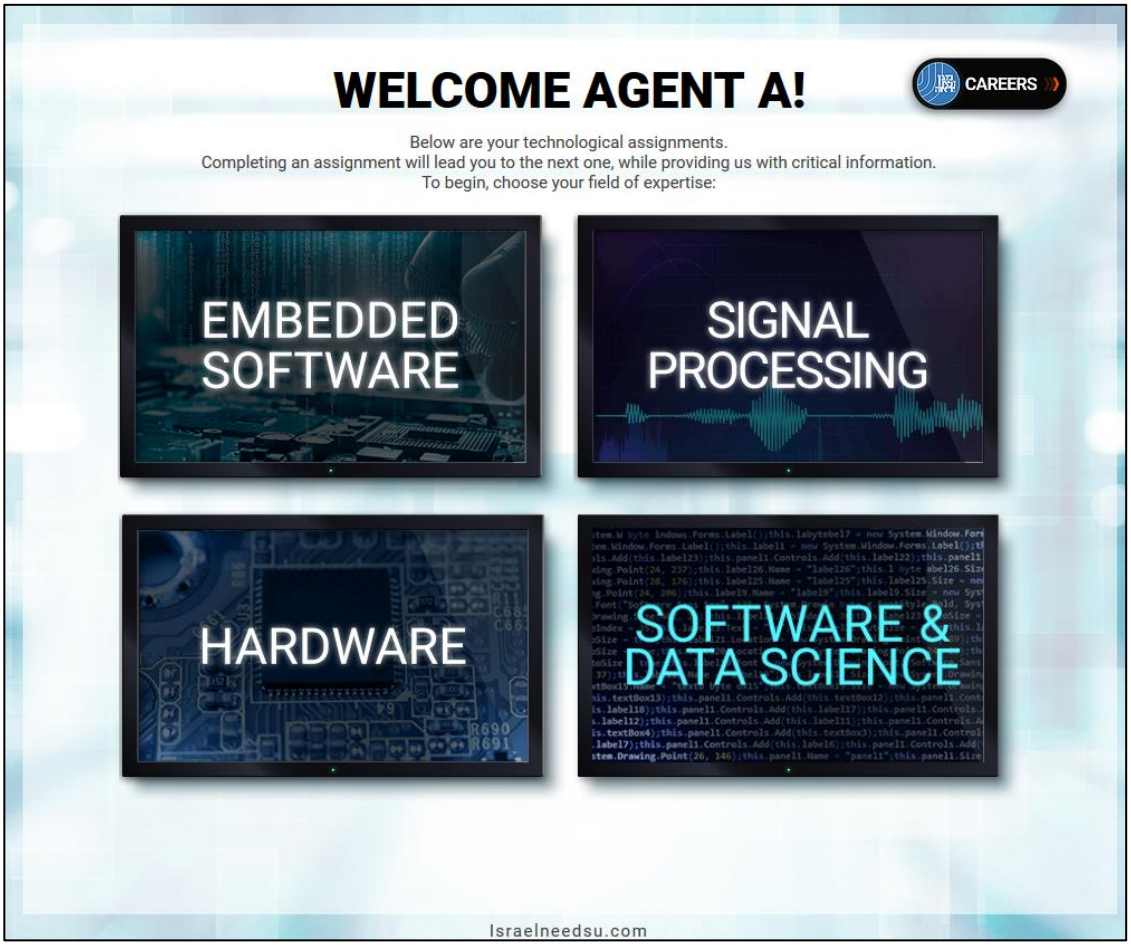






כל נקודה מציינת מבנה או סימן-נוף בצורה של אות באנגלית. אם נחבר את האותיות נקבל את התשובה: **joinus**.

התשובה מובילה אל ארבעת מסלולי האתגר:



שאלה #1: Cat and Mouse

תיאור האתגר:

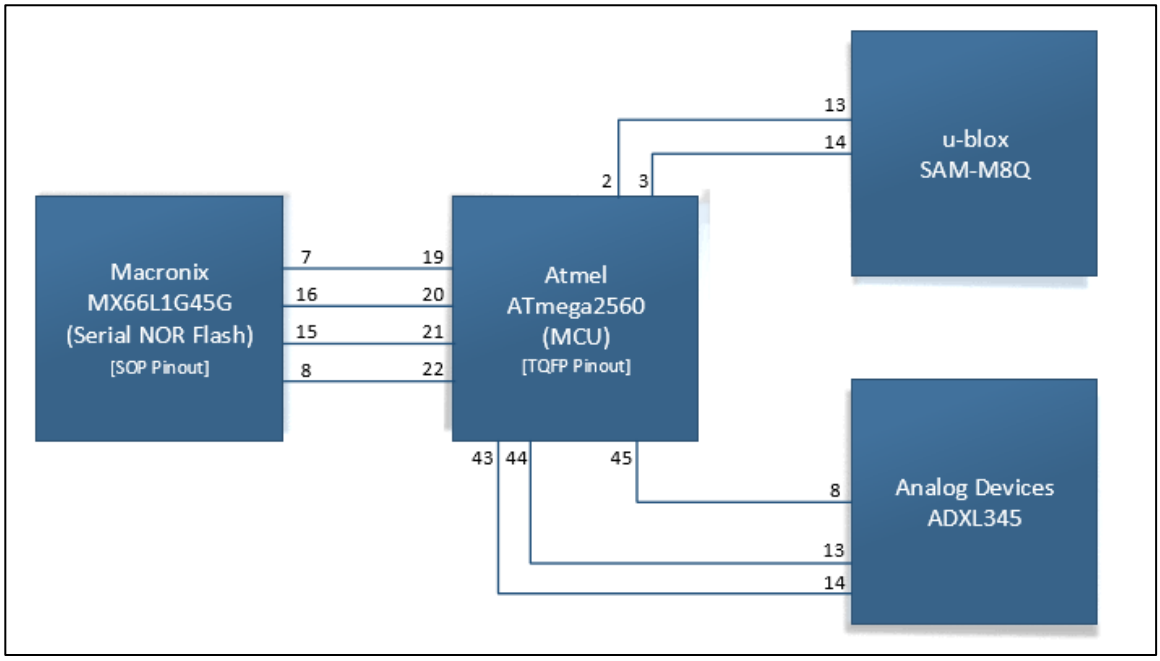
A routine counter-surveillance check of a senior Minister of Defense's vehicle revealed an electronic device in the undercarriage. We suspected it was a tracking device, and sent it to the Technological Department for an in-depth analysis.
After reverse-engineering the product, the following information was uncovered:

1. A partial scheme of the electrical circuit and its components (see the attached electronic_scheme.pdf file).
2. A disassembly of the code programmed to the micro controller (see the attached program.c file).
3. The memory dump of the external memory component (see attached external_mem_dump.bin file).

Earlier that week, we had intercepted a suspicious SMS sent by a suspected member of White September. The suspect's message read "package received".
The message was received on 30/10/18, at 01:21 AM UTC. Our analysis team analyzed the data and determined that the message had most likely been sent by the same WS member who had installed the device in the senior official's vehicle. The analysts suspect he retrieved it from a WS dead drop.
Our engineers believe that when the message was sent, the device was online, and that therefore the location of the dead drop could potentially be extracted from it.

Your mission:
Find the exact coordinates of the dead drop.

הקבצים המצורפים כללו, כאמור, תרשים של המכשיר:



קוד המקור:

```
#include <stdio.h>
#include <memory.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>

#define TRUE 1
#define FALSE 0

#define PARSER_INVALID 0
#define PARSER_TYPE_1 1
#define PARSER_TYPE_2 2

#define PARSER_PREFIX_1 "GPGGA"
#define PARSER_PREFIX_2 "GPRMC"
#define PARSER_PREFIX_LENGTH 5
#define MAX_UART_BUFFER 256
#define DATA_MAX_LENGTH 100
#define DATA_MIN_LENGTH 9
#define MAX_SAVE_BUFFER_SIZE 256

uint8_t should_save = FALSE;
uint8_t reset = TRUE;
uint8_t counter = 0;
uint8_t counter_max_val = 75;
uint8_t is_triggered = FALSE;

static uint32_t uart_read(uint8_t *buffer){/* Read data from UART0
until 0x00 terminator */};

static uint8_t parse(uint8_t *in_buffer, uint32_t length, uint8_t
**out_buffers)
{
  uint8_t i = 0;
  uint8_t res = PARSER_INVALID

  if (length < DATA_MIN_LENGTH || length > DATA_MAX_LENGTH ||
*in_buffer != '$' || in_buffer[length - 1] != '\n' || in_buffer[6] !=
',')
    return PARSER_INVALID;

  if (0 == strncmp(in_buffer + 1, PARSER_PREFIX_1,
PARSER_PREFIX_LENGTH))
    res = PARSER_TYPE_1;
  else if (0 == strncmp(in_buffer + 1,
PARSER_PREFIX_2,PARSER_PREFIX_LENGTH))
    res = PARSER_TYPE_2;
  else
      return PARSER_INVALID;

  in_buffer = in_buffer + PARSER_PREFIX_LENGTH + 2;

  out_buffers[i++] = in_buffer;
  while (NULL != (in_buffer = strchr(in_buffer, ',')))
  {
    *in_buffer = '\0';
    out_buffers[i++] = ++in_buffer;
  }
```

```
  return res;
}

static uint32_t format_save(uint8_t *in_buffer, uint8_t a, uint32_t
length, uint8_t *out_buffer)
{
  *out_buffer = a;
  *(uint32_t *)&out_buffer[1] = length;
  memcpy(&out_buffer[5], in_buffer, length);
  return length + 5;
}

static uint32_t format1(uint8_t **in_buffer, uint8_t *out_buffer)
{
  *(int *)out_buffer = atoi(in_buffer[0]);
  out_buffer += sizeof(float);
  *(int *)out_buffer = atoi(in_buffer[8]);
  return 2 * sizeof(float);
}

static uint32_t format2(uint8_t **in_buffers, uint8_t *out_buffer)
{
  char val1_str[10] = { 0 };
  for (uint8_t i = 0; i < 3; i += 2)
  {
    memset(val1_str, 0, sizeof(val1_str));
    uint8_t *pos = strchr(in_buffers[i], '.');
    memcpy(val1_str, in_buffers[i], pos - in_buffers[i] - 2);
    uint16_t val1 = atoi(val1_str);
    float val2 = atof(pos - 2);
    float res =  val1 + (val2 / 60);
    if (*in_buffers[i + 1] == 'W' || *in_buffers[i + 1] == 'S')
      res *= -1;
      *(float *)out_buffer = res;
      out_buffer += sizeof(float);
  }
  return 2 * sizeof(float);
}

static void configure1(void)
{
  cli();
  should_save = FALSE;
  counter = 0;
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1 = 0;
  OCR1A = 62499;
  TCCR1B |= (1 << WGM12);
  TCCR1B |= (1 << CS12) | (1 << CS10);
  TIMSK1 |= (1 << OCIE1A);
  sei();
}
static void configure2(void)
{
  cli();
  EICRA |= 0x01 << 2;
  EIMSK |= 0x01 << 2;
```

```
  sei();
}

ISR(TIMER1_COMPA_vect)
{
  if (++counter == counter_max_val)
  {
    should_save = TRUE;
    counter = 0;
    counter_max_val = (is_triggered == TRUE) ? 15 : 150;
  }
}

ISR(INT2_vect)
{
  static uint8_t interrupt_buffer[MAX_SAVE_BUFFER_SIZE];
  uint8_t *temp_buffer;
  uint8_t a;
  uint32_t length;
  if (PIND & 0x01)
  {
     is_triggered = TRUE;
     a = 2;
  }
  else
  {
     is_triggered = FALSE;
     a = 3;
  }
  length = format_save(temp_buffer, a, 0, interrupt_buffer);
  save_to_flash(interrupt_buffer, length);
}

static void save_to_flash(uint8_t *buffer, uint32_t length){ /* Save
buffer to flash at next empty address */ }

static void configure_usart0(void) { /* Configure USART 0 */ }

static void configure_spi(void) { /* Configure SPI */ }

static void configure_two_wire_serial_interface(void) { /* Configure
the two wire serial interface */ }

int main()
{
  uint32_t size = 0;
  uint8_t recv_buffer[MAX_UART_BUFFER];

  uint8_t *parsed_buffers[20];
  uint8_t save_buffer[MAX_SAVE_BUFFER_SIZE];
  uint8_t temp_buffer[MAX_SAVE_BUFFER_SIZE];
  uint32_t save_length = 0;
  uint8_t res;
  uint32_t formatted_length;
  uint8_t a = FALSE;

  configure_sysclk(); /* Assume system clock is 16 MHz */
  configure_usart0();
  configure_spi();
```

```
  configure_two_wire_serial_interface();
  configure1();
  configure2();

  while (1)
  {
    formatted_length = 0;
    size = uart_read(recv_buffer);

    if (size > 0)
    {
      res = parse(recv_buffer, size, parsed_buffers);
      if (reset == TRUE)
      {
         if(res != PARSER_TYPE_2) continue;

         formatted_length = format1(parsed_buffers, temp_buffer);

         reset = FALSE;
         a = TRUE;
      }
      else if (should_save == TRUE)
      {
        if(res != PARSER_TYPE_1) continue;

        formatted_length = format2(&parsed_buffers[1], temp_buffer);

        should_save = FALSE;

        a = FALSE;
      }

      if(formatted_length > 0)
      {
        save_length = format_save(temp_buffer, (a == TRUE) ? 0 : 1,
formatted_length, save_buffer);
        save_to_flash(save_buffer, save_length);
      }
    }
  }
}
```

וקובץ בינארי שמכיל Memory Dump של המכשיר:

```
root@kali:/media/sf_CTFs/shabak/Cat_and_Mouse/program# xxd -g1
external_mem_dump.bin
00000000: 00 08 00 00 00 48 e8 01 00 ba 49 04 00 02 00 00  .....H....I.....
00000010: 00 00 01 08 00 00 00 aa 05 00 42 a9 8a 0b 42 01  ..........B...B.
00000020: 08 00 00 00 43 05 00 42 0e 8a 0b 42 01 08 00 00  ....C..B...B....
00000030: 00 7a 05 00 42 18 89 0b 42 01 08 00 00 00 27 06  .z..B...B.....'.
00000040: 00 42 0a 88 0b 42 01 08 00 00 00 9b 06 00 42 40  .B...B........B@
...
000057f0: 42 ac 28 0b 42 01 08 00 00 00 96 4c 00 42 60 28  B.(.B......L.B`(
00005800: 0b 42 01 08 00 00 00 c8 4c 00 42 21 28 0b 42 01  .B......L.B!(.B.
00005810: 08 00 00 00 9e 4c 00 42 00 29 0b 42 9e 4c 00 42  .....L.B.).B.L.B
00005820: 00 29 0b 42                                      .).B
```

מעיון בקוד עצמו נראה שפונקציית main מחכה לקלט סריאלי, ואז מבצעת עיבוד ראשוני באמצעות פונקציית parse. פונקציה זו מצפה לקבל קלט לפי אחד משני פורמטים: פורמט GPGGA ופורמט GPRMC. מחיפוש מהיר נראה שהפורמטים הללו מייצגים מידע הקשור ל-GPS.

התוכנה מקבלת את הקלט בלולאה, ראשית רשומה אחת של GPGGA ולאחריה רצף אינסופי של רשומות GPRMC. את הרשומות התוכנה מייצגת באמצעות ייצוג פנימי שלה (הפורמט של הודעת GPGGA נבנה בפונקציית format1 והפורמט של הודעת GPRMC נבנה בפונקציית format2). לאחר מכן, הייצוג נכתב אל זיכרון הפלאש של המכשיר במידה ומורם דגל לעשות כן. הדגל עבור רשומת GPGGA מורם פעם אחת בתחילת כל תוכנית, בעוד הדגל עבור רשומת GPRMC נשלט על ידי טיימר.

הטיימר קופץ בכל x יחידות זמן, מעלה משתנה-מונה באחת ומחליט לכתוב את הרשומה לפלאש במידה והמונה הגיע ל-counter_max_val.

מעבר למידע שמתקבל דרך הממשק הסריאלי, התוכנה מקבלת קלט גם דרך Pin D. כאשר ערך הסיגנל משתנה, מתקבל Interrupt והאירוע נכתב לפלאש. כמו כן, הערך של counter_max_val מתעדכן לפי האירוע הנ"ל.

הפורמט של השמירה לפלאש, כפי שניתן לראות מ-format_save, הוא בית אחד עבור סוג הרשומה:

- 0: הודעת GPGGA
- 1: הודעת GPRMC
- 2: סיגנל אשר מעיד על is_triggered = TRUE
- 3: סיגנל אשר מעיד על is_triggered = FALSE

לאחריו גודל המשך הרשומה (4 בתים) ומיד לאחר מכן תוכן באורך מספר הבתים שצוינו בשדה הקודם.

Type	Length	Payload
1 Byte	4 Bytes	<Length> Bytes

הן לרשומות GPGGA והן לרשומות GPRMC ישנו Payload של שמונה בתים: שני משתני Integer עבור GPGGA ושני משתני Float עבור GPRMC. מכאן אפשר לפרש את הרשומה הראשונה ב-dump שראינו קודם:

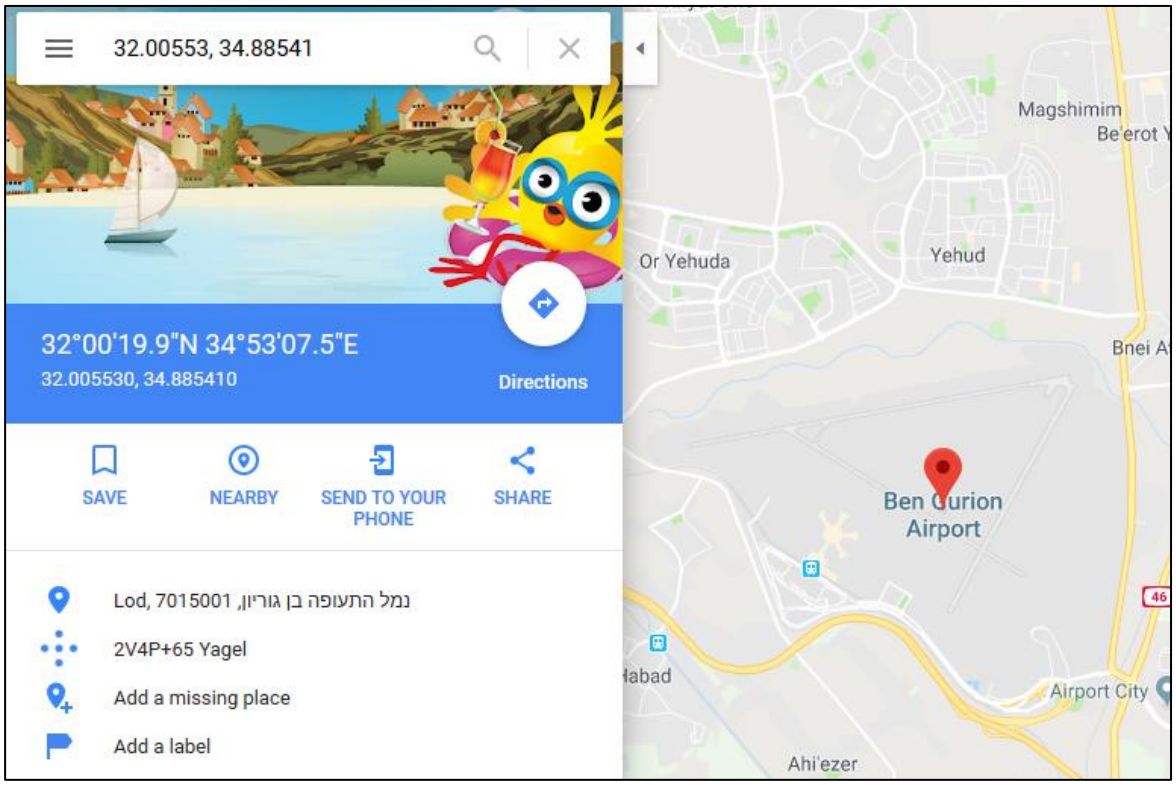
```
00 08 00 00 00 48 e8 01 00 ba 49 04 00
```

נראה שמדובר ברשומה מסוג 0 (GPGGA), באורך 0x00000008 בתים, עם הערכים 0x0001e848 ו-0x000449ba (שימו לב להיפוך שנגרם עקב השימוש בייצוג Little Endian). אם נתבונן בערכים בבסיס עשרוני נקבל 125000 ו-281018, מה שנראה כמו תאריך ושעה.

באותו אופן, נפרש את רשומת ה-GPRMC הראשונה שנפגוש (מתחילה ב-0x12):

```
01 08 00 00 00 aa 05 00 42 a9 8a 0b 42
```

זוהי רשומת GPRMC, באורך 8, עם הערכים 0x420005aa ו-0x420b8aa9. נפרש כ-Float ונקבל: 32.00553, 34.85541. אם נתייחס לנתונים אלו בתור קואורדינטות אורך ורוחב, נקבל מיקום בלב נמל התעופה בן גוריון:



אפשר להוציא כך את כל הקואורדינטות, ולקבל מפה של המסלול המדויק בו טייל המכשיר:

הקואורדינטות נותנות לנו את המסלול, אבל אנחנו צריכים להצליב את המסלול הזה עם הזמן המדויק שבה נשלחה ההודעה המפלילה. לכן אנחנו צריכים להבין מתי בדיוק כל רשומה מסוג GPRMC נכתבה אל הפלאש.

אנחנו יודעים שרשומת GPRMC נכתבת לפלאש ברגע שהטיימר מרים דגל של should_save, וזה קורה ברגע שהוא משלים counter_max_val קריאות. counter_max_val הוא 15 או 150, בהתאם לערך הנוכחי של is_triggered (למעט בתחילת התוכנה, שם הוא מתחיל כ-75). אנחנו יודעים את המצב הנוכחי של is_triggered כי כל שינוי שלו נכתב לפלאש, ולכן אנחנו יכולים להסיק את הערך של counter_max_val. כל מה שנשאר זה להבין כל כמה יחידות זמן קופץ הטיימר, ונוכל לחשב את הזמן שעובר בין כל כתיבה לכתיבה.

הקוד כולל את השורה הבאה:

```
configure_sysclk(); /* Assume system clock is 16 MHz */
```

שמעידה על שעון של 16 MHz. עובדה זו מסתדרת גם עם ה-spec של השבב שצויין בתרשים המצורף.

בנוסף, מחיפוש בגוגל, נראה שההגדרות הבאות קשורות גם הן לקביעת זמן ההתעוררות של הטיימר:

```
    OCR1A = 62499;
    TCCR1B |= (1 << WGM12);
    TCCR1B |= (1 << CS12) | (1 << CS10);
    TIMSK1 |= (1 << OCIE1A);
```

מצאנו את ההסבר המצוין הבא באתר הזה:

```
Timer1 is set to interrupt on an overflow, so if you are using an
ATmega328 with a 16MHz clock. Since Timer1 is 16 bits, it can hold a
maximum value of (2^16 - 1), or 65535. At 16MHz, we'll go through one
clock cycle every 1/(16*10^6) seconds, or 6.25*10^-8 s. That means 65535
timer counts will pass in (65535 * 6.25*10^-8s) and the ISR will trigger
in about 0.0041 seconds.
To control this you can also set the timer to use a prescaler, which
allows you to divide your clock signal by various powers of two, thereby
increasing your timer period.  For example, if you want the LED blink at
one second intervals. In the TCCR1B register, there are three CS bits to
set a better timer resolution.  If you set CS10 and CS12 using:
TCCR1B |= (1 << CS10); and TCCR1B |= (1 << CS12);, the clock source is
divided by 1024. This gives a timer resolution of 1/(16*10^6 / 1024), or
0.000064 seconds (15625 Hz). Now the timer will overflow every (65535 *
6.4*10^-5s), or 4.194s.
```

במקרה שלנו, הטיימר סופר עד 62499, ולכן הוא יקפוץ כל 3.999936 = 0.000064 * 62499 שניות, או בקירוב כל 4 שניות. ולכן, אם counter_max_val מוגדר ל-15, הטיימר יקפוץ כל דקה, ואם הוא מוגדר ל-150, הטיימר יקפוץ כל עשר דקות.

בעזרת המידע הזה נוכל לייצר את הסקריפט הבא:

```
import os
import struct
from datetime import datetime, timedelta

ticks = 0
add_ticks = [75]

with open("external_mem_dump.bin", "rb") as f:
    while True:
        type = struct.unpack('B', f.read(1))[0]
        if type == "":
            break

        length = struct.unpack('I', f.read(4))[0]

        if type == 0:
            assert(length == 8)
            value1, value2 = struct.unpack('ii', f.read(8))
            print ("0\t{}\t{} - reset".format(value1, value2))
            ticks = 0
            add_ticks = [75]
            the_time = datetime.strptime("{} {}".format(value1, value2),
"%H%M%S %d%m%y")
        elif type == 1:
            assert(length == 8)
            if len(add_ticks) == 1:
                ticks_to_add = add_ticks[0]
            else:
                ticks_to_add = add_ticks.pop(0)
            ticks += ticks_to_add
            the_time += timedelta(seconds=ticks_to_add*4)
            value1, value2 = struct.unpack('ff', f.read(8))
```

```
            print ("1\t{:.5f}\t{:.5f}\t{}\t{}".format(value1, value2,
ticks, the_time))
        elif type == 2:
            assert(length == 0)
            print ("2\tis_triggered = TRUE (Log every 15 ticks)")
            add_ticks = [add_ticks[0], 15]
        elif type == 3:
            assert(length == 0)
            print ("3\tis_triggered = FALSE (Log every 150 ticks)")
            add_ticks = [add_ticks[0], 150]
        else:
            assert(False)
```

הסקריפט מממש את כל מה שהזכרנו, יחד עם נקודה אחת אחרונה שצריך לשים לב אליה: אם הערך של is_triggered משתנה, מה שיגרום לשינוי של counter_max_val, הטיימר הבא עדיין יקפוץ לפי הערך הישן, והערך של counter_max_val יתעדכן רק לאחר מכן לערך החדש.

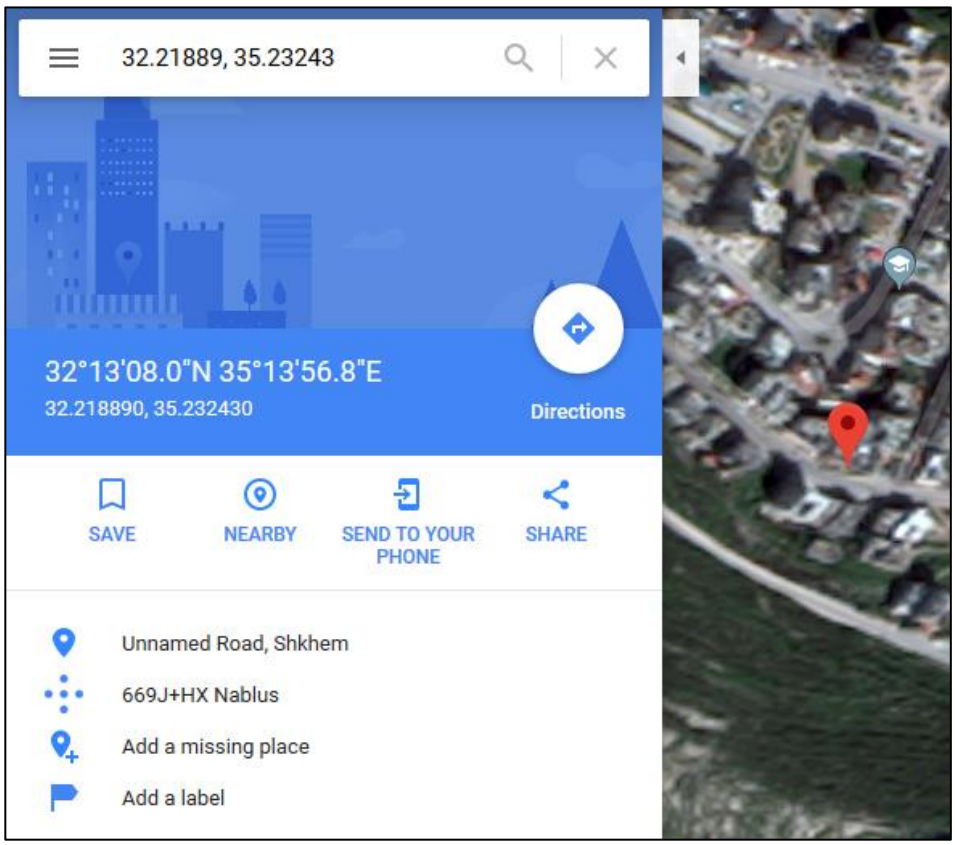
נריץ ונקבל:

```
0     125000      281018 - reset
2     is_triggered = TRUE (Log every 15 ticks)
1     32.00553          34.88541          75    2018-10-28 12:55:00
1     32.00514          34.88482          90    2018-10-28 12:56:00
1     32.00535          34.88388          105   2018-10-28 12:57:00
1     32.00601          34.88285          120   2018-10-28 12:58:00
1     32.00645          34.88208          135   2018-10-28 12:59:00
1     32.00696          34.88113          150   2018-10-28 13:00:00
1     32.00732          34.88019          165   2018-10-28 13:01:00
1     32.00666          34.87967          180   2018-10-28 13:02:00
…
1     32.21888          35.23234          29085 2018-10-30 00:36:00
1     32.21895          35.23225          29235 2018-10-30 00:46:00
1     32.21903          35.23241          29385 2018-10-30 00:56:00
1     32.21892          35.23242          29535 2018-10-30 01:06:00
1     32.21889          35.23243          29685 2018-10-30 01:16:00
1     32.21893          35.23226          29835 2018-10-30 01:26:00
1     32.21908          35.23232          29985 2018-10-30 01:36:00
1     32.21906          35.23253          30135 2018-10-30 01:46:00
1     32.21887          35.23255          30285 2018-10-30 01:56:00
…
```

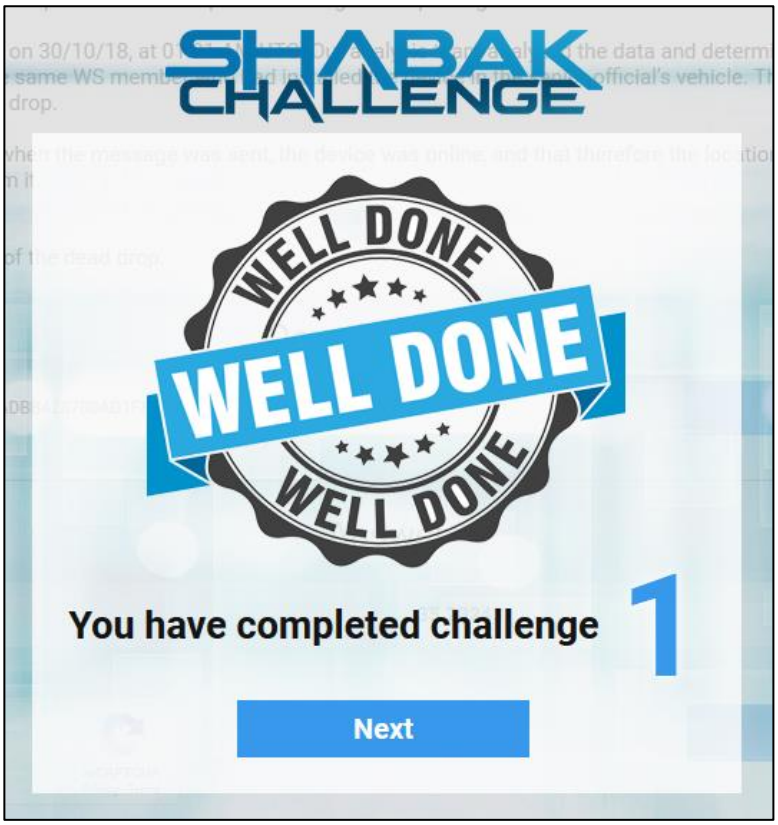
ההודעה נשלחה ב-30.10.2018 01:21, מה שאומר שהרשומה הקרובה ביותר היא:

```
1     32.21889          35.23243          29685 2018-10-30 01:16:00
```

הקואורדינטות הן: 32.21889, 35.23242. אי שם במרכז שכם:



והתוצאה:



שאלה #2: Code Red

תיאור האתגר:

We have uncovered some suspicious online traffic and identified communication between two IP addresses. Both these addresses come from computers we believe belong to senior members of the White September organization (see attached file chat.pcap).

The Technological Department believes that the WS members are using an application designed for concealing messages.

Based previously gathered intel, our analysts team assesses that WS is planning a terror attack against Israel in the near future. They believe the exact date was disclosed in the aforementioned communication.

Your Mission:
Reveal the exact date and time of the planned attack.

לאתגר צורף קובץ תעבורת רשת בשם chat.pcap.

אחד הדברים הראשונים שכדאי לעשות כאשר מקבלים קובץ כזה הוא לעקוב אחרי ה-TCP Streams השונים שלו. ניתן לעשות זאת באמצעות הממשק הגרפי של WireShark, או באמצעות הסקריפט הבא שמייצא כל Stream לקובץ טקסט:

```
END=$(tshark -r chat.pcap -T fields -e tcp.stream | sort -n | tail -1);
for ((i=0;i<=END;i++)); do echo $i; tshark -r chat.pcap -qz
follow,tcp,ascii,$i > follow-stream-$i.txt;done
```

במקרה שלנו, הפקודה ייצאה 39 סטרימים שונים. באמצעות מעבר ידני עליהם, זיהינו שהסטרימים המעניינים הם 3 ו-11, אשר בניגוד לשאר הסטרימים לא כללו גישה לאתרי אינטרנט שגרתיים אלא מעין דו-שיח בין שתי נקודות ברשת המקומית.

תחילה, בוצעה בקשת POST ל-sessions/ שהניבה את התשובה הבאה:


```
{"Id":"8d671e57-4f6a-4a7c-a22b-
724578cecbfa","Urls":["https://static.wixstatic.com/media/57cf4c_afea1f0
bb82348d9bdc24653ea3208f9~mv2.png","https://static.wixstatic.com/media/5
7cf4c_9fa5cba479a24e73a24fb52163d9209b~mv2.png","https://static.wixstati
c.com/media/57cf4c_4080f95bf84349e5887042c3a06f7114~mv2.png","https://st
atic.wixstatic.com/media/57cf4c_0bf3bbad5f74409bad0a3a10b1dbd537~mv2.png
","https://static.wixstatic.com/media/57cf4c_0aa6e7ffcc024f7ba2b6611f72f
2432d~mv2.png","https://static.wixstatic.com/media/57cf4c_d9f88c5ddc9348
8d91ac03c56cc901ae~mv2.png","https://static.wixstatic.com/media/57cf4c_5
6a9ed0fd9c84c98935307aebb4783f7~mv2.png"]}
```


לאחר מכן, מספר בקשות POST ל-messages/ שהניבו תוצאות מהסוג הבא:


```
{"SessionId":"8d671e57-4f6a-4a7c-a22b-724578cecbfa","Content":"<base64
encoded data>","Counter":0}
```

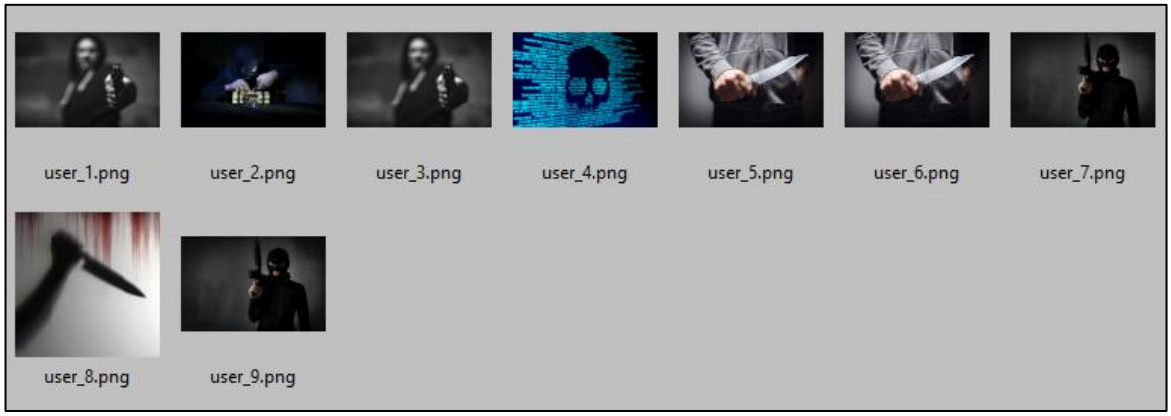

בכל תוצאה, Content הכיל מחרוזת Base64, ו-Counter הכיל מספר כלשהו.

עוד כדאי לציין שתגובת ה-HTTP הייתה chunked, כלומר, הפורמט של התשובה היה מספר שמציין את אורך ה-chunk, ולאחריו מספר בתים באורך זה. למשל:

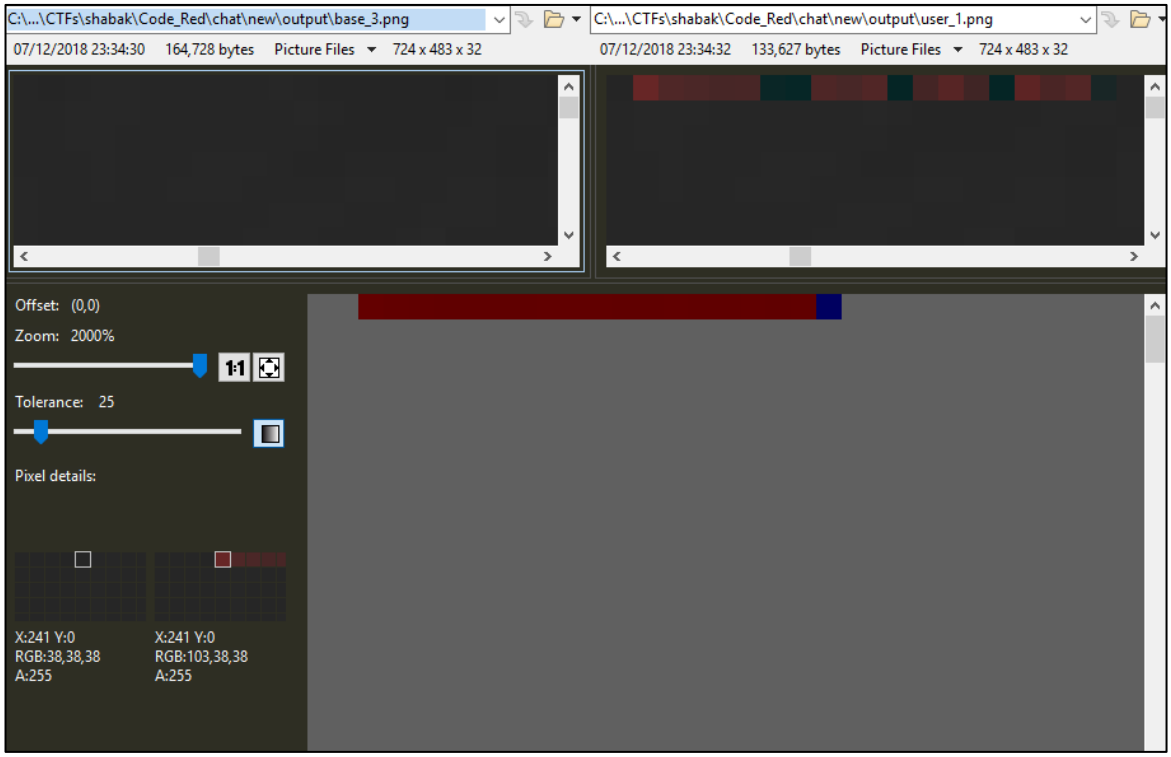
```
POST /messages HTTP/1.1
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Host: 192.168.202.128

400
{"SessionId":"8d671e57-4f6a-4a7c-a22b-724578cecbfa","Content":"<bas64
encoded chunk...>
400
<bas64 encoded chunk of length 0x400...>
400
<bas64 encoded chunk of length 0x400...>
400
<bas64 encoded chunk of length 0x400...>
```

כלומר, כדי לקבל אובייקט JSON נקי, אפשר לומר שהיה צריך לסנן החוצה כל שורה שנייה (אשר כוללת את אורך ה-chunk). נתחיל מאובייקט ה-JSON הראשון, שכולל רשימת תמונות, ונוריד את כל התמונות:

אם נתרגם את קידוד ה-base64 של האובייקטים הבאים ונייצא לקבצים, נקבל גם כן תמונות דומות (הקוד שעושה זאת יצורף בסוף):



התמונות שהורדנו והתמונות שחילצנו נראות דומות, אבל אם נשווה אותן פיקסל-פיקסל נגלה שישנו אזור מסוים שבו יש הבדל בערכי הצבע האדום. כך זה נראה ויזואלית:



אפשר לראות משמאל-למעלה את תמונת הבסיס (שהורדה מהאינטרנט) לצד התמונה שחולצה מאובייקט ה-JSON מימין, ולמטה סימון של הפיקסלים השונים. שימו לב שהתמונה המקורית רציפה, בעוד שנראה שהשינוי נעשה על התמונה המחולצת.

בפינה השמאלית התחתונה אפשר לראות את הערכים עבור פיקסל מסוים, כאשר ערך ה-R של התמונה המקורית הינו 38 והערך עבור התמונה המחולצת הינו 103. נבצע XOR ביניהם ונקבל 0x41, או בתרגום ל-ASCII, האות A. נמשיך כך ונקבל:

```
Ahlan, how are you?
```

נראה טוב! אבל אם נבצע את הפעולה הזאת על התמונות האחרות (כמובן, תוך הקפדה שאנחנו תמיד משווים בין תמונה מחולצת אשר דומה ויזואלית לתמונת בסיס), נקבל במקרים האחרים ג'יבריש!

פה נכנס לפעולה ערך ה-Counter שראינו שצורף לאובייקט ה-JSON. מסתבר שיש לבצע XOR גם איתו. במקרה הראשון, הערך היה 0 כך שהפעולה לא הייתה הכרחית, אך במקרים האחרים הפעולה הזו נדרשת על מנת לקבל פלט הגיוני.

הסקריפט הבא:

- יוריד את התמונות הרלוונטיות מהאינטרנט
- יחלץ את התמונות הרלוונטיות מאובייקטי ה-JSON
- יזהה עבור כל תמונה מחולצת מהי תמונת הבסיס המתאימה באמצעות אלגוריתם בסיסי להשוואת תמונות

- ישווה בין ערכי ה-R של כל פיסקל ופיסקל בתמונת הבסיס (base) ובתמונה המחולצת (user), ויחשב את Counter ^ base[x,y][R] ^ user[x,y][R] במידה ו-user[x,y][R] != base[x,y][R]

```
from PIL import Image, ImageFont, ImageChops
import functools, requests, operator, logging
import base64, pickle, heapq, json, glob, math
import re, os

re_line_with_single_number = re.compile(r'^[0-9A-F]+$')
OUT_DIR = "output"
USER_IMAGE_PREFIX = "user_"
BASE_IMAGE_PREFIX = "base_"
CACHE_FILE = "cache.db"

#http://www.guguncube.com/1656/python-image-similarity-comparison-using-several-
techniques
def image_similarity_histogram_via_pil(filepath1, filepath2):

    image1 = Image.open(filepath1)
    image2 = Image.open(filepath2)

    h1 = image1.histogram()
    h2 = image2.histogram()

    rms = math.sqrt(functools.reduce(operator.add,list(map(lambda a,b: (a-b)**2,
h1, h2)))/len(h1) )
    return rms

def clean_json_str(s):
    lines = s.split("\n")
    return "".join(lines[::2]) # Return every other line

def extract_json_objects_from_stream(stream_id):
    res = []
    stream_file = "follow-stream-{}.txt".format(stream_id)
    logging.info("Extracting JSON objects from stream file:
'{}'".format(stream_file))
    with open(stream_file) as f:
        for raw_json_str in re.findall(r"(\{[^}]+\})", f.read()):
            j = json.loads(clean_json_str(raw_json_str))
            res.append(j)
    logging.info("Extracted {} JSON objects".format(len(res)))
    return res

def find_base_img(img):
    logging.info("Finding base image for '{}'".format(img))
    similarity = []
    for base_img in
glob.glob(os.path.join(OUT_DIR,"{}*.png".format(BASE_IMAGE_PREFIX)) ):
        score = image_similarity_histogram_via_pil(base_img, img)
        logging.debug("\tScore for '{}' is '{}'".format(base_img, score))
        heapq.heappush(similarity, (score, base_img))
    best_match = heapq.heappop(similarity)[1]
    logging.info("Best match for '{}': '{}'".format(img, base_img))
    return best_match

def decode_message(img1, img2, xor):
    image1 = Image.open(img1)
```

```
    image2 = Image.open(img2)

    assert(image1.size == image2.size)

    xSize = image1.size[0]
    ySize = image1.size[1]

    diff = []
    for i in range(xSize):
        for j in range(ySize):
            if image1.getpixel((i, j)) != image2.getpixel((i, j)):
                v1 = image1.getpixel((i, 0))[0]
                v2 = image2.getpixel((i, 0))[0]
                diff.append(chr(v1 ^ v2 ^ xor))
    return ("".join(diff))

logging.basicConfig(level=logging.INFO, format='Log:\t%(message)s')

json_objects = []
for stream_id in ["3", "11"]:
    json_objects += extract_json_objects_from_stream(stream_id)

if not os.path.exists(OUT_DIR):
    logging.info("Output directory missing, loading from scratch")
    os.mkdir(OUT_DIR)

    image_to_counter = dict()
    for i, obj in enumerate(json_objects):
        if "Urls" in obj:
            for url_index, url in enumerate(obj["Urls"]):
                logging.info("Fetching URL: {}".format(url))
                response = requests.get(url, stream=True)
                out_path =
os.path.join(OUT_DIR,"{}{}.png".format(BASE_IMAGE_PREFIX,url_index))
                logging.info("Saving to: {}".format(out_path))
                with open(out_path, "wb") as o:
                    o.write(response.content)
        else:
            out_image_name = "{}{}.png".format(USER_IMAGE_PREFIX, i)
            out_path = os.path.join(OUT_DIR, out_image_name)
            logging.info(
                "Saving image from object #{} (Counter: {}) to: {}".format(i,
obj["Counter"], out_path))
            with open(out_path, "wb") as o:
                o.write(base64.b64decode(obj["Content"]))
                image_to_counter[out_image_name] = int(obj["Counter"])

    pickle.dump( image_to_counter, open( CACHE_FILE, "wb" ) )
else:
    logging.info("Output directory missing, loading from cache")
    image_to_counter = pickle.load( open( CACHE_FILE, "rb" ) )

for img in glob.glob( os.path.join(OUT_DIR,"{}*.png".format(USER_IMAGE_PREFIX)) ):
    logging.info("Extracting message from '{}'".format(img))
    base_img = find_base_img(img)
    print ("\n", decode_message(img,
base_img,image_to_counter[os.path.basename(img)]),"\n")
```

הפלט:

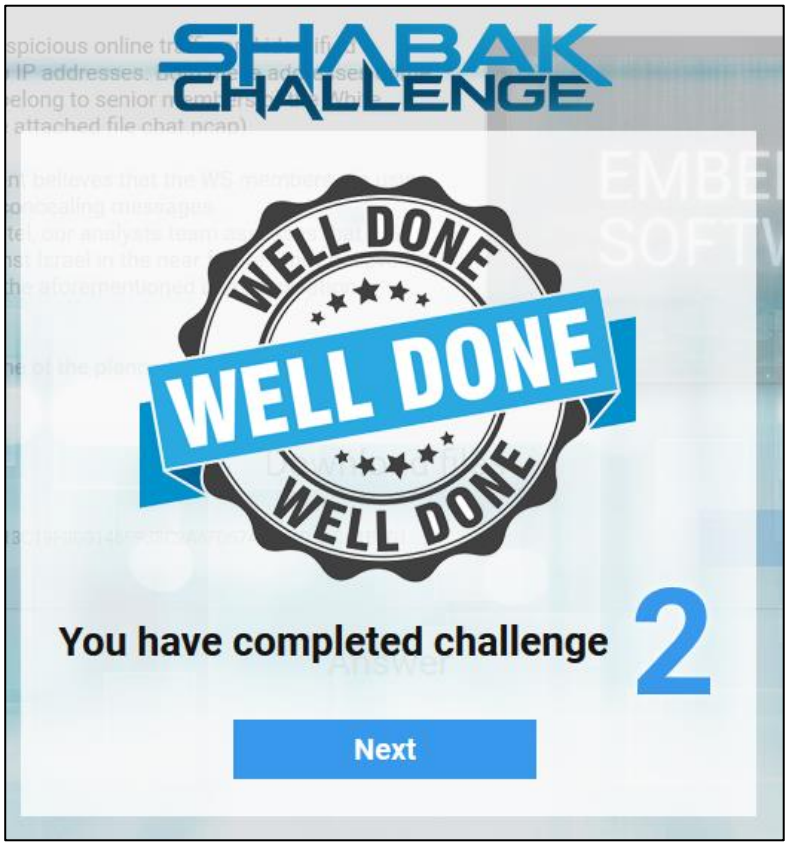
```
Log:    Extracting JSON objects from stream file: 'follow-stream-3.txt'
Log:    Extracted 6 JSON objects
Log:    Extracting JSON objects from stream file: 'follow-stream-11.txt'
Log:    Extracted 4 JSON objects
Log:    Output directory missing, loading from scratch
Log:    Fetching URL: https://static.wixstatic.com/media/57cf4c_afea1f0bb82348d9bdc24653ea3208f9~mv2.png
Log:    Saving to: output\base 0.png
Log:    Fetching URL: https://static.wixstatic.com/media/57cf4c 9fa5cba479a24e73a24fb52163d9209b~mv2.png
Log:    Saving to: output\base_1.png
Log:    Fetching URL: https://static.wixstatic.com/media/57cf4c_4080f95bf84349e5887042c3a06f7114~mv2.png
Log:    Saving to: output\base_2.png
Log:    Fetching URL: https://static.wixstatic.com/media/57cf4c_0bf3bbad5f74409bad0a3a10b1dbd537~mv2.png
Log:    Saving to: output\base 3.png
Log:    Fetching URL: https://static.wixstatic.com/media/57cf4c 0aa6e7ffcc024f7ba2b6611f72f2432d~mv2.png
Log:    Saving to: output\base_4.png
Log:    Fetching URL: https://static.wixstatic.com/media/57cf4c_d9f88c5ddc93488d91ac03c56cc901ae~mv2.png
Log:    Saving to: output\base_5.png
Log:    Fetching URL: https://static.wixstatic.com/media/57cf4c_56a9ed0fd9c84c98935307aebb4783f7~mv2.png
Log:    Saving to: output\base 6.png
Log:    Saving image from object #1 (Counter: 0) to: output\user_1.png
Log:    Saving image from object #2 (Counter: 3) to: output\user_2.png
Log:    Saving image from object #3 (Counter: 5) to: output\user 3.png
Log:    Saving image from object #4 (Counter: 6) to: output\user 4.png
Log:    Saving image from object #5 (Counter: 8) to: output\user 5.png
Log:    Saving image from object #6 (Counter: 1) to: output\user 6.png
Log:    Saving image from object #7 (Counter: 2) to: output\user_7.png
Log:    Saving image from object #8 (Counter: 4) to: output\user_8.png
Log:    Saving image from object #9 (Counter: 7) to: output\user 9.png
Log:    Extracting message from 'output\user 1.png'
Log:    Finding base image for 'output\user 1.png'
Log:    Best match for 'output\user 1.png': 'output\base 6.png'

 Ahlan, how are you?

Log:    Extracting message from 'output\user 2.png'
Log:    Finding base image for 'output\user 2.png'
Log:    Best match for 'output\user_2.png': 'output\base_6.png'

 Allahumdulillah! How are all the brothers?

Log:    Extracting message from 'output\user 3.png'
Log:    Finding base image for 'output\user 3.png'
Log:    Best match for 'output\user_3.png': 'output\base_6.png'

 We are all very proud of you.

Log:    Extracting message from 'output\user 4.png'
Log:    Finding base image for 'output\user_4.png'
Log:    Best match for 'output\user_4.png': 'output\base_6.png'

 We want the party to start at 20:10 exactly.

Log:    Extracting message from 'output\user 5.png'
Log:    Finding base image for 'output\user_5.png'
Log:    Best match for 'output\user_5.png': 'output\base_6.png'

 Inshallah, Allahu Akbar.

Log:    Extracting message from 'output\user 6.png'
Log:    Finding base image for 'output\user_6.png'
Log:    Best match for 'output\user_6.png': 'output\base_6.png'

 Ahlan habibi, I'm fine.

Log:    Extracting message from 'output\user_7.png'
Log:    Finding base image for 'output\user_7.png'
Log:    Best match for 'output\user_7.png': 'output\base_6.png'

 How are you?

Log:    Extracting message from 'output\user_8.png'
Log:    Finding base image for 'output\user_8.png'
```

```
Log:    Best match for 'output\user_8.png': 'output\base_6.png'

 They are all excited for new year's eve.

Log:    Extracting message from 'output\user_9.png'
Log:    Finding base image for 'output\user_9.png'
Log:    Best match for 'output\user_9.png': 'output\base_6.png'

 Inshallah, hopefully a lot of people will come.
```

השיחה עצמה:

```
Ahlan, how are you?
Allahumdulillah! How are all the brothers?
We are all very proud of you.
We want the party to start at 20:10 exactly.
Inshallah, Allahu Akbar.
Ahlan habibi, I'm fine.
How are you?
They are all excited for new year's eve.
Inshallah, hopefully a lot of people will come.
```

כלומר, המתקפה מתוכננת ל-31/12/18 בשעה 20:10:



אתגר #1: Find the Code

הוראות האתגר:

We have received intel that White September is trying to recruit Israeli residents to operate on its behalf within Israel. Our SIGINT unit discovered a seemingly innocent website where remote technological jobs are advertised. They believe this site is WS's recruiting front.

Based on the advertised jobs, the unit infers that WS is attempting to recruit tech agents from within key positions in Israeli security and government organizations.

After submitting a fake resume, the SIGINT agents received a password-protected ZIP file and an instruction document. The instructions informed them that the ZIP password is a series of digits and that the ZIP file contains Python code and two images. To advance to the next stage of recruitment, the unit's agents need to find a secret code in the image files.

Your Mission:

Unlock the ZIP file, extract its contents, and crack the code implanted in the images.

ראשית, נפצח את הסיסמא של קובץ ה-zip באמצעות John the Ripper:

```
root@kali:/media/sf_CTFs/shabak/Find_the_Code# ~/utils/john/run/zip2john clues.zip >
zip.hashes

ver 2.0 efh 5455 efh 7875 clues.zip/clue.png PKZIP Encr: 2b chk, TS_chk, cmplen=2424517,
decmplen=2428065, crc=4E28B3FE ver 2.0 efh 5455 efh 7875 clues.zip/clueTwo.jpg PKZIP Encr:
2b chk, TS_chk, cmplen=522, decmplen=12427, crc=B30EDBEE ver 2.0 efh 5455 efh 7875
clues.zip/something.txt PKZIP Encr: 2b chk, TS_chk, cmplen=299, decmplen=532, crc=729ED871
NOTE: It is assumed that all files in each archive have the same password. If that is not
the case, the hash may be uncrackable. To avoid this, use option -o to pick a file at a
time.

root@kali:/media/sf_CTFs/shabak/Find_the_Code# ~/utils/john/run/john --incremental=digits
zip.hashes
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Warning: OpenMP is disabled; a non-OpenMP build may be faster
Press 'q' or Ctrl-C to abort, almost any other key for status
262626 (clues.zip)
1g 0:00:00:00 DONE (2018-12-06 22:53) 1.851g/s 48474p/s 48474c/s 48474C/s 262507..262005
Use the "--show" option to display all of the cracked passwords reliably
Session completed

root@kali:/media/sf_CTFs/shabak/Find_the_Code# unzip -P 262626 clues.zip
Archive:  clues.zip
  inflating: clue.png
  inflating: clueTwo.jpg
  inflating: something.txt
```

מכיוון שידענו שהסיסמא מורכבת מספרות בלבד, הפיצוח לקח שניות בודדות. הסיסמא היא 262626.

נבחן את something.txt:

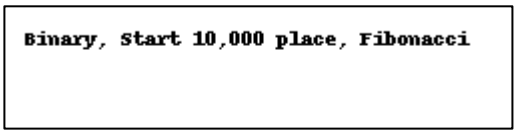
```
#env 3.7 from PIL import Image, ImageFont import textwrap from
pathlib import Path def find_text_in_image(imgPath): image =
Image.open(imgPath) red_band = image.split()[0] xSize =
image.size[0] ySize = image.size[1] newImage = Image.new("RGB",
image.size) imagePixels = newImage.load() for f in range(xSize):
for j in range(zSize) if bin(red_band.getpixel((i, j)))[-1] ==
'0': imagePixels[i, j] = (255, 255, 255) else: imagePixels[i, j]
= (0,0,0) newImgPath=str(Path(imgPath).parent.absolute())
newImage.save(newImgPath+'/text.png')
```

נראה כמו סקריפט פייתון. נסדר את השורות ואת הריווח, נתקן כמה שגיאות קלות ונסיר תלויות מיותרות:

```
#env 3.7
from PIL import Image, ImageFont
#import textwrap from pathlib
#import Path

def find_text_in_image(imgPath):
    image = Image.open(imgPath)
    red_band = image.split()[0]
    xSize = image.size[0]
    ySize = image.size[1]
    newImage = Image.new("RGB", image.size)
    imagePixels = newImage.load()
    for i in range(xSize):
        for j in range(ySize):
            if bin(red_band.getpixel((i, j)))[-1] == '0':
                imagePixels[i, j] = (255, 255, 255)
            else:
                imagePixels[i, j] = (0,0,0)
    #newImgPath=str(Path(imgPath).parent.absolute())
    newImage.save('./text.png')
```

הסקריפט מחפש תמונה נסתרת באמצעות התמקדות בפיקסלים האדומים של התמונה בלבד. כעת אפשר להריץ את הסקריפט על התמונה clue.png שמצאנו בתוך קובץ ה-zip.

התוצאה היא התמונה הבאה:



שימו לב שקיימת עוד תמונה, clueTwo.jpg, אך הסקריפט לא עוזר במקרה של התמונה הזו. כנראה שהמשפט שהוצאנו מ-clue.png אמור לעזור לנו לפענח רמז שמסתתר ב-clueTwo.jpg. clueTwo.jpg היא בסך הכל תמונה לבנה גדולה עם מלבן אפור קטן בפינה הימנית התחתונה (המסגרת השחורה אינה חלק מהתמונה ונועדה להדגשת הגבולות בלבד):

הפרט המעניין היחיד בתמונה הוא אזור מסוים בלב הייצוג הבינארי של התמונה, שבו קיים הטקסט הבא:

```
000004B0  00 A2 8A 28 00 A2 8A 28 00 A2 8A 28 00 A2 8A 28  .¢.(.¢.(.¢.(.¢.(
000004C0  00 A2 8A 28 00 A2 8A 28 00 A2 8A 28 00 A2 8A 28  .¢.(.¢.(.¢.(.¢.(
000004D0  00 A2 8A 28 00 A2 8A 28 00 A2 8A 28 00 A2 8A 28  .¢.(.¢.(.¢.(.¢.(
000004E0  00 62 79 6F 75 62 67 64 62 6F 78 7A 6D 6B 74 78  .byoubgdboxzmktx
000004F0  00 A2 8A 28 00 A2 69 71 72 A2 8A 28 00 A2 8A 28  .¢.(.¢iqr¢.(.¢.(
00000500  00 A2 8A 74 00 A2 8A 28 00 A2 8A 28 00 A2 8A 28  .¢.t.¢.(.¢.(.¢.(
00000510  00 A2 8A 28 00 A2 8A 28 00 A2 8A 28 00 A2 8A 28  .¢.(.¢.(.¢.(.¢.(
00000520  00 A2 8A 28 00 A2 8A 28 00 A2 8A 28 00 A2 8A 28  .¢.(.¢.(.¢.(.¢.(
```

ניסינו אינספור כלים וכיוונים על מנת לפצח את הרמז, עד שהגענו אל הפירוש הבא:

- יש להתייחס אל התמונה כבינארי
- יש לקפוץ אל הביט ה-10,000 של הקובץ
- משם, יש לחלץ את הבתים בהתבסס על קפיצות בהתאם לסדרת פיבונאצ'י

הסקריפט הבא מבצע זאת:

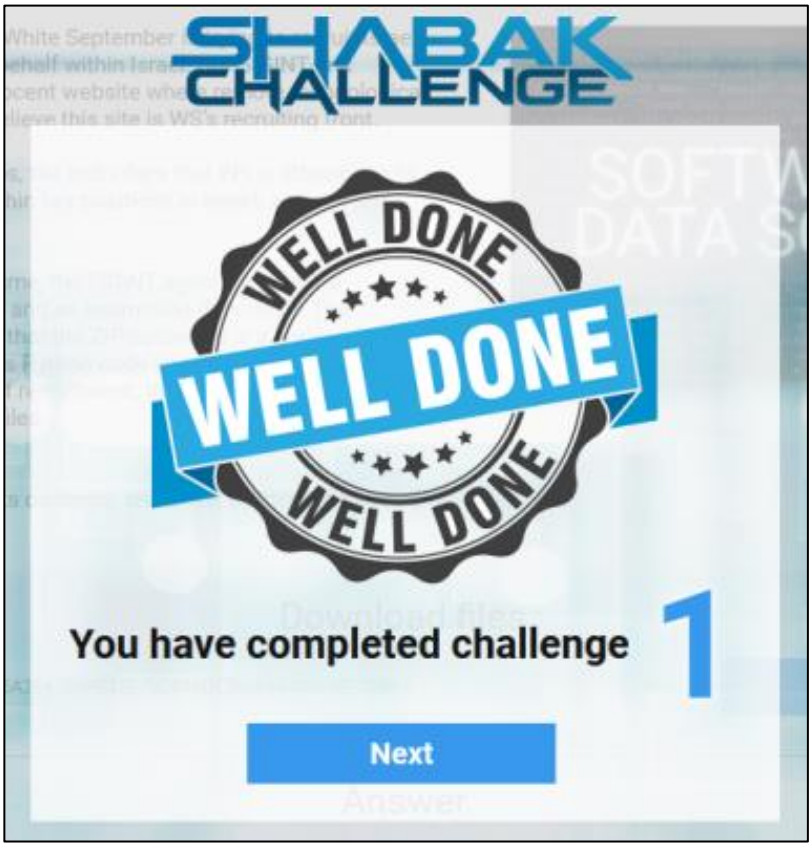
```
import mmap, os, string

def bits_to_bytes(bits):
    return bits // 8

def memory_map(filename, access=mmap.ACCESS_WRITE):
    size = os.path.getsize(filename)
    fd = os.open(filename, os.O_RDWR)
    return mmap.mmap(fd, size, access=access)

def get_next_fib():
    a = 0
    b = 1
```

```
    while True:
        yield a
        s = a + b
        a = b
        b = s

with memory_map("clueTwo.jpg", access=mmap.ACCESS_READ) as f:
    index = bits_to_bytes(10000)

    for offset in get_next_fib():
        index += offset
        c = chr(f[index])
        if c not in string.printable:
            break
        print(c, end='')
```

התוצאה:

```
yougotit
```

באופן מפתיע ביותר, yougotit לא התקבל כתשובה, אבל you got it כן התקבל:



אתגר #2: The Persian

הוראות האתגר:

The code you discovered enabled our SIGINT unit to proceed in the recruitment process. The unit received a mysterious file. Finding the code in the file will enable us to further advance in the WS recruiting process.

Your Mission:

Analyze the file's content and discover the code

קובץ ה-zip המצורף הכיל קובץ אשר היה קרוי WhoAmI.jpg, אך תוכנו למעשה היה טקסטואלי. התוכן הכיל טקסט רב אשר במבט ראשון התאים לתבנית הבאה:

```
{"0x1": [{"text": "z9u05d3su05e9u05d7u05e4gz p2P <more...>", "value":
69349}, {...}, {...}]}
```

מה שבלט לעין היה החזרות הרבות של מחרוזות מסוג u05**XX** - ייצוג של טווח יוניקוד שמתאים לעברית.

הפעולה הטבעית הבאה הייתה לתרגם את התווים לעברית (שימו לב שרק לתווים מהתבנית u05XX קיים תרגום טריוויאלי לעברית, מה שמשאיר לא מעט ספרות ותווים באנגלית). התוצאה הייתה ג'יבריש. כמו כן, מחכה לנו נתון נוסף של value שלא השתמשנו בו.

אפשרות אחת, שאמנם נראתה קלושה במבט ראשון אך בדיעבד התגלתה בתור הכיוון הנכון, הייתה לחשב את הסכום הגימטרי של כל האותיות בעברית. הסכום שחושב עבור האובייקט הראשון התאים לערך הנקוב!

מכאן, הדבר הטבעי הבא היה לוודא את התוצאה עבור כל שאר האובייקטים. אולם, תוך כדי ריצה התגלו הבעיות הבאות:

1. לא לכל האובייקטים קיים ערך בשדה value, לעיתים במקום ערך היה כתוב פשוט "?".
2. לא כל האובייקטים הכילו שדה "text".

בפעם הראשונה שהשדה "text" היה חסר, הטקסט נכלל בשדה שנקרא "return" במקום. בפעם הבאה שהשדה היה חסר, הטקסט נכלל בשדה "in". בפעם השלישית, בשדה "base64", ולכולם כבר אמור להיות ברור שמדובר פה במסר נסתר עם הוראות נוספות.

לאחר שליפת כל השמות של השדות שהכילו טקסט ואשר לא נקראו "text", המסר שהתקבל היה:

```
return in base64 sum of values below median
```

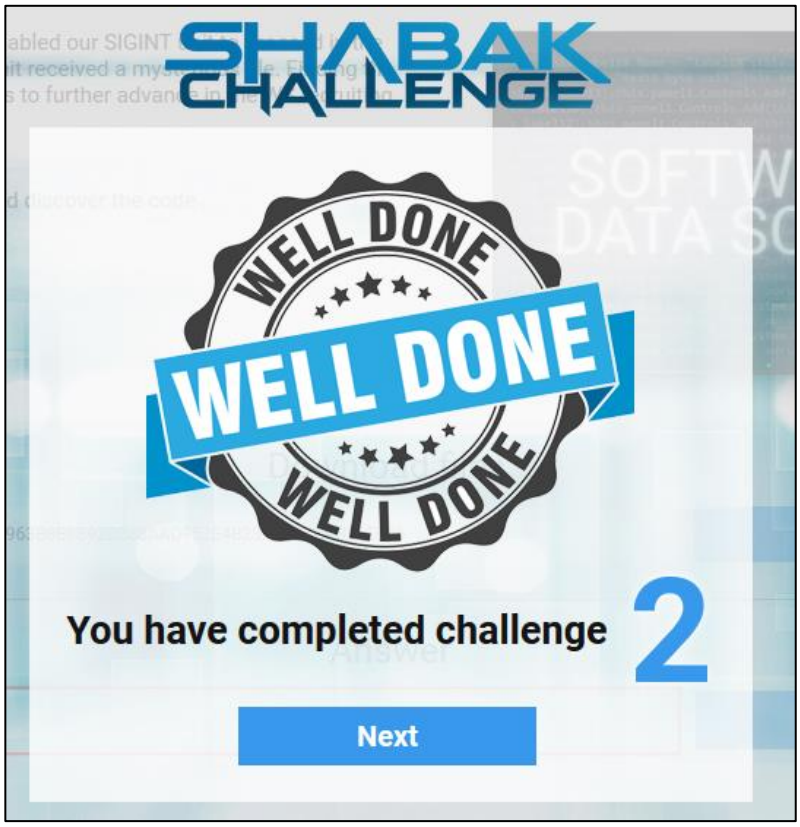
אם כך, כל שנותר הוא לעקוב אחרי ההוראות:

```
import statistics, logging, base64, json, re

heb_utf_regex = "u(05[a-f0-9]{2})"

gematria_map = {
    0x05D0: 1, # ﻩspan>
    0x05D1: 2, # ﻫspan>
    0x05D2: 3, # 工span>
    0x05D3: 4, # 甑span>
    0x05D4: 5, # 伯span>
    0x05D5: 6, # 弯span>
    0x05D6: 7, # 漯span>
    0x05D7: 8, # 缯span>
    0x05D8: 9, # 輯span>
    0x05D9: 10, # 鼯span>
    0x05DA: 20, # 罗span>
    0x05DB: 20, # 鬯span>
    0x05DC: 30, # 烈span>
    0x05DD: 40, # span>
    0x05DE: 40, # □span>
    0x05DF: 50, # Ｏspan>
    0x05E0: 50, # □pan>
    0x05E1: 60, # □pan>
    0x05E2: 70, # □pan>
    0x05E3: 80, # □pan>
    0x05E4: 80, # pan>
    0x05E5: 90, # pan>
    0x05E6: 90, # pan>
    0x05E7: 100, # pan>
    0x05E8: 200, #
    0x05E9: 300, #
    0x05EA: 400, #

}

def get_gematria_sum(text):
    sum = 0
    for res in re.findall(heb_utf_regex, text):
        v = int(res, 16)
        sum += gematria_map[int(res, 16)]
    return sum

logging.basicConfig(level=logging.INFO, format='Log:\t%(message)s')

with open ("WhoAmI.jpg") as f:
    message = ""
    values = []

    all_text = f.read()
    all_text = all_text.replace("?", '"""')
    j = json.loads(all_text)

    for section_id, text_val_arr in j.items():
        logging.info("Section: " + section_id)
        for text_val in text_val_arr:
            text_label = "text"
            for x in text_val:
                if x == "value":
                    pass
                elif x != text_label:
                    message += x + " "
                    text_label = x
```

```
            text = text_val[text_label]
            value = text_val["value"]
            calculated_value = get_gematria_sum(text)
            if (value != ""):
                logging.info("Gematria value from source: {}, calculated value:
{}".format(value, calculated_value))
                assert(int(value) == calculated_value)
            else:
                logging.info("Gematria value not found in source, calculated
value: {}".format(calculated_value))

            values.append(calculated_value)

    print("Secret message: {}".format(message))

    med = statistics.median(values)
    print("Median: {}".format(med))

    s = sum([v for v in values if v < med])
    print("Sum of values under median: {}".format(s))

    print("Base64: ", base64.b64encode(str(s).encode("utf-8")))
```

התוצאה:

```
Secret message: return in base64 sum of values below median
Median: 63664.5
Sum of values under median: 2501577
Base64:  b'MjUwMTU3Nw=='
```



אתגר #3: The Usual Suspect

הוראות האתגר:

An ISA cyber-attack against White September's servers procured a set of data files. The files contained WS members' browsing activity. An analysis of the servers' logs revealed the conspiratorial activity of ten apparent suspects. We know there are more suspects but have been unable to identify them as of yet. The number of additional suspects is unknown.

Your Mission:

Identify the remaining suspects based on the activity of the provided list of suspects.

Find the most frequently used IP address for each of the additional suspects.

***Your solution should be submitted as a comma-separated list of IP addresses in ascending order, without any spaces or additional characters.**

For example: 11.1.11.1,2.22.22.2 (assuming two new suspects were found).

לאתגר צורף קובץ CSV של 10,000,000 רשומות (כ-600 מגה). הרשומות נראו כך:

```
$ head log.csv
uid, uip, date, url
8771,186.189.5.113,01/08/2014 17:24,http://jlt.edu/
7175,241.64.83.103,01/11/2014 12:54,http://sds-german.cc/
7903,224.4.135.115,01/03/2014 10:52,http://fdd.co.uk/
2690,211.105.56.228,01/01/2014 19:13,http://epc.org/
1518,198.77.195.195,01/11/2014 19:33,http://www.popshop.dk/
6589,225.108.49.121,01/07/2014 00:32,http://mend.co.uk/
6756,116.225.186.227,01/01/2014 20:44,http://simi.org/
7207,73.151.121.164,01/30/2014 07:19,http://mnj.cc/
1199,114.9.216.164,01/25/2014 14:21,http://lra.org/
```

כלומר, אוסף רשומות שכל אחת כוללת מזהה משתמש, כתובת IP, תאריך, וכתובת אתר. כמו כן, צורפה רשימה של חשודים:

```
2449, 6796, 9237, 4024, 3538, 3608, 7239, 435, 5206, 2211
```

בסך הכל, הרשימה כללה 10,000 רשומות אשר התייחסו לחשודים:

```
$ cat log.csv | egrep
"^(2449|6796|9237|4024|3538|3608|7239|435|5206|2211)," | wc -l
10000
$ cat log.csv | egrep
"^(2449|6796|9237|4024|3538|3608|7239|435|5206|2211)," | head
3538,67.141.120.237,01/16/2014 07:31,http://isa.edu/
5206,10.192.20.173,01/23/2014 17:51,http://fdd.co.uk/
4024,230.167.210.226,01/12/2014 23:26,http://apcls.info/
3608,127.95.83.100,01/24/2014
18:02,http://www.math.rutgers.edu/~sontag/fermi-eng.html
4024,143.204.212.207,01/16/2014 18:48,http://www.realestatebook.com/
7239,143.204.212.207,01/12/2014 20:26,http://emetic.edu/
```

```
435,42.74.74.110,01/02/2014 21:40,http://professional-edu.blogspot.com/
2211,109.242.247.39,01/06/2014
12:08,http://www.noodletools.com/noodlebib/
3538,51.70.255.188,01/10/2014 14:50,http://alansmith17.tumblr.com/
4024,230.167.210.226,01/13/2014 07:52,http://kdp.edu/
```

המחשבה הראשונית הייתה לחפש באילו אתרים ביקרו כל החשודים, ולראות אילו עוד משתמשים ביקרו באתרים הללו:

```
suspects = [2449, 6796, 9237, 4024, 3538, 3608, 7239, 435, 5206, 2211]
suspect_history = defaultdict(set)
non_suspect_history = defaultdict(set)
with open('log.csv') as f:
    csv_reader = csv.reader(f, delimiter=',')
    next(csv_reader, None)  # skip the headers
    for row in csv_reader:
        if int(row[UID]) in suspects:
            suspect_history[row[UID]].add(row[URL])
        else:
            non_suspect_history[row[UID]].add(row[URL])

    common_sites_for_suspects =
set.intersection(*list(suspect_history.values()))
    print ("Common sites for suspects: ", common_sites_for_suspects)
    counter = 0
    for user, history in non_suspect_history.items():
        if common_sites_for_suspects.issubset(history):
            counter += 1
    #        print (user)
    print ("Number of non-suspects who visited all common sites: ",
counter)
```

התוצאה: ישנם שני אתרים שכל החשודים ביקרו בהם ('http://cgsb.net', 'http://anla.gr/'), אבל 3128 משתמשים אחרים ביקרו בשניהם. לא מספיק טוב בשביל לצמצם את הרשימה. הרעיון הבא היה לבדוק אילו משתמשים השתמשו בכתובות IP שבהן השתמש חשוד כלשהו:

```
suspect_ips = set()
with open('log.csv') as f:
    csv_reader = csv.reader(f, delimiter=',')
    next(csv_reader, None)  # skip the headers
    for row in csv_reader:
        if int(row[UID]) in suspects:
            suspect_ips.add(row[IP])

    print("IPs used by suspects: ", suspect_ips)
    f.seek(0)
    next(csv_reader, None)  # skip the headers
    counter = 0
    for row in csv_reader:
        if int(row[UID]) not in suspects and row[IP] in suspect_ips:
            #print(row)
            counter += 1
    print ("Number of times which users used suspects' IPs: ", counter)
```

התוצאה: משתמשים רגילים השתמשו בכתובות של חשודים 9659 פעמים. יותר מדי. כדי לחשב מסלול מחדש, הבטנו בנתונים הגולמיים:

```
user_to_ips = defaultdict(set) # user to all the ips he used
with open('log.csv') as f:
    csv_reader = csv.reader(f, delimiter=',')
    next(csv_reader, None)  # skip the headers
    for row in csv_reader:
        user_to_ips[int(row[UID])].add(row[IP])

for user, ip_set in user_to_ips.items():
    print ("!" if user in suspects else " ", user, len(ip_set), ip_set)
```

הסקריפט מדפיס מיפוי של כל משתמש אל כל כתובות ה-IP שלו. אם המשתמש חשוד, בתחילת השורה יודפס סימן קריאה.

ראשית, נבחן את נתוני החשודים:

```
$ cat user_to_ips.txt | grep !
! 3538 10 {'127.95.83.100', '124.9.188.155', '162.219.33.114', … , '109.242.247.39'}
! 5206 10 {'127.95.83.100', '162.219.33.114', '10.192.20.173', … , '41.239.144.6'}
! 4024 10 {'230.167.210.226', '162.219.33.114', '10.192.20.173', … , '143.204.212.207'}
! 3608 10 {'127.95.83.100', '139.210.78.22', '124.9.188.155', … , '109.242.247.39'}
! 7239 10 {'127.95.83.100', '162.219.33.114', '10.192.20.173', … , '143.204.212.207'}
! 435 10  {'127.95.83.100', '162.219.33.114', '10.192.20.173', … , '41.239.144.6'}
! 2211 10 {'127.95.83.100', '162.219.33.114', '51.70.255.188', … , '41.239.144.6'}
! 2449 10 {'127.95.83.100', '124.9.188.155', '162.219.33.114', … , '41.239.144.6'}
! 6796 10 {'230.167.210.226', '10.192.20.173', '68.17.81.83', … , '143.204.212.207'}
! 9237 10 {'230.167.210.226', '10.192.20.173', '68.17.81.83', … '143.204.212.207'}
```

מה שקופץ פה לעין הוא החזרות הרבות של כתובות IP בין חשודים. למשל, הכתובת הראשונה של החשוד הראשון (127.95.83.100) היא גם הכתובת הראשונה של החשוד השני, הרביעי, החמישי, השישי, השביעי והשמיני. התופעה חוזרת עם כתובות אחרות. מכאן הגיע הרעיון "לצבוע" את הכתובות הללו ולראות אילו משתמשים השתמשו בכתובות אלו:

```
all_suspect_ips = set()
for suspect in suspects:
    all_suspect_ips.update(user_to_ips[suspect])

with open("user_to_ips.txt") as f, open("user_to_ips_colored.txt", "w")
as o:
    user_to_ips_content = f.read()
    for ip in all_suspect_ips:
        user_to_ips_content = user_to_ips_content.replace(ip,
"SUSPECT_IP")
    o.write(user_to_ips_content)
```

התוצאה, כאשר מסננים רק משתמשים שאינם חשודים (כלומר, ללא סימן קריאה):

```
$ cat user_to_ips_colored.txt | grep SUSPECT_IP | grep -v !
  7159 10 {'SUSPECT_IP', 'SUSPECT_IP', '219.62.226.13', 'SUSPECT_IP',
'126.127.244.219', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP',
'SUSPECT_IP'}
  5630 10 {'53.101.7.178', '176.143.187.81', 'SUSPECT_IP', '219.62.226.13',
'82.148.13.233', '126.127.244.219', '114.14.209.5', '58.176.126.71',
'7.199.192.98', '100.115.183.61'}
  9091 10 {'53.101.7.178', '176.143.187.81', 'SUSPECT_IP', '219.62.226.13',
'82.148.13.233', 'SUSPECT_IP', '126.127.244.219', '7.199.192.98', 'SUSPECT_IP',
'100.115.183.61'}
```

```
  1808 10 {'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP',
'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP'}
  5772 10 {'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP', '219.62.226.13',
'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP',
'SUSPECT_IP'}
  5590 10 {'53.101.7.178', '176.143.187.81', 'SUSPECT_IP', 'SUSPECT_IP',
'219.62.226.13', 'SUSPECT_IP', '126.127.244.219', '7.199.192.98', 'SUSPECT_IP',
'100.115.183.61'}
  7051 10 {'176.143.187.81', 'SUSPECT_IP', 'SUSPECT_IP', '219.62.226.13',
'SUSPECT_IP', '126.127.244.219', 'SUSPECT_IP', 'SUSPECT_IP', '100.115.183.61',
'SUSPECT_IP'}
  6529 10 {'176.143.187.81', 'SUSPECT_IP', 'SUSPECT_IP', '219.62.226.13',
'SUSPECT_IP', '126.127.244.219', '7.199.192.98', 'SUSPECT_IP', '100.115.183.61',
'SUSPECT_IP'}
  4918 10 {'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP',
'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP'}
  87 10 {'53.101.7.178', '176.143.187.81', 'SUSPECT_IP', '82.148.13.233',
'219.62.226.13', 'SUSPECT_IP', '126.127.244.219', '114.14.209.5',
'7.199.192.98', '100.115.183.61'}
  9482 10 {'176.143.187.81', 'SUSPECT_IP', 'SUSPECT_IP', '219.62.226.13',
'SUSPECT_IP', '126.127.244.219', 'SUSPECT_IP', 'SUSPECT_IP', 'SUSPECT_IP',
'SUSPECT_IP'}
```

זה נראה כמו הכיוון הנכון - ישנם משתמשים (כמו למשל 4918) שכל כתובות ה-IP שלהם הן כתובות של חשודים!

נייצר סקריפט שיבצע את הזיהוי כולו מההתחלה ועד הסוף:

```
from collections import defaultdict, Counter
import logging, pickle, socket, time, csv, os

def most_common(lst):
    data = Counter(lst)
    return data.most_common(1)[0][0]

UID      = 0
IP       = 1
DATE     = 2
URL      = 3

LOG_FILE     = "log.csv"
CACHE_FILE   = "cache.db"

suspects = []

logging.basicConfig(level=logging.INFO, format='Log:\t%(message)s')

with open("hint.txt") as hint_f:
    for suspect in hint_f.readlines():
        suspect = suspect.rstrip()
        suspects.append(suspect)

logging.info("Suspects: {}".format(suspects))

if not os.path.exists(CACHE_FILE):
    logging.info("Cache file missing, loading from scratch")
    user_to_ips              = defaultdict(set)   # user to all the ips he used
    ips_to_user              = defaultdict(set)   # ip to all the users which
used it
    user_to_nonunique_ips    = defaultdict(list)  # user to list of IPs he used
    user_to_most_common_ip   = dict()             # user to most common IP he used
    all_suspect_ips          = set()              # Set of all suspect IPs
```

```
    with open(LOG_FILE) as f:
        time1 = time.time()
        csv_reader = csv.reader(f, delimiter=',')
        logging.info("Parsing log file...")
        for i, row in enumerate(csv_reader):
            user_to_ips[row[UID]].add(row[IP])
            ips_to_user[row[IP]].add(row[UID])
            user_to_nonunique_ips[row[UID]].append(row[IP])
        time2 = time.time()

    logging.info("Parsed {} rows in {} seconds".format(i, time2-time1))

    logging.info("Number of unique users:
{}".format(len(user_to_ips.keys())))
    logging.info("Number of unique IPs:
{}".format(len(ips_to_user.keys())))

    logging.info("Calculating most common IP per user...")
    for user, ip_list in user_to_nonunique_ips.items():
        mc = most_common(ip_list)
        user_to_most_common_ip[user] = mc

    logging.info("Searching for all suspect IPs...")
    for suspect in suspects:
        all_suspect_ips.update(user_to_ips[suspect])

    logging.info("All suspect IPs: {}".format(all_suspect_ips))

    pickle.dump( (user_to_ips, ips_to_user,
                  user_to_most_common_ip,
                  all_suspect_ips), open( CACHE_FILE, "wb" ) )

else:
    logging.info("Loading from cache")
    user_to_ips, ips_to_user, user_to_most_common_ip, all_suspect_ips \
        = pickle.load( open( CACHE_FILE, "rb" ) )

non_suspects_using_suspect_ips = defaultdict(int)
for user, user_ips in user_to_ips.items():
    if user in suspects:
        continue
    for ip in user_ips:
        if ip in all_suspect_ips:
            non_suspects_using_suspect_ips[user] += 1

logging.info("Non-suspects using suspect IPs: {}"
                .format(non_suspects_using_suspect_ips))

top_n = 3
logging.info("Fetching top {} new suspects:".format(top_n))
d = Counter(non_suspects_using_suspect_ips)
ips_for_new_suspects = []
for user, num_common_ips in d.most_common(top_n):
    mc_ip = user_to_most_common_ip[user]
    print ("New suspect: {}, # suspicious IPs: {}, Most common IP: {}"
            .format(user, num_common_ips, mc_ip))
    ips_for_new_suspects.append(mc_ip)
```

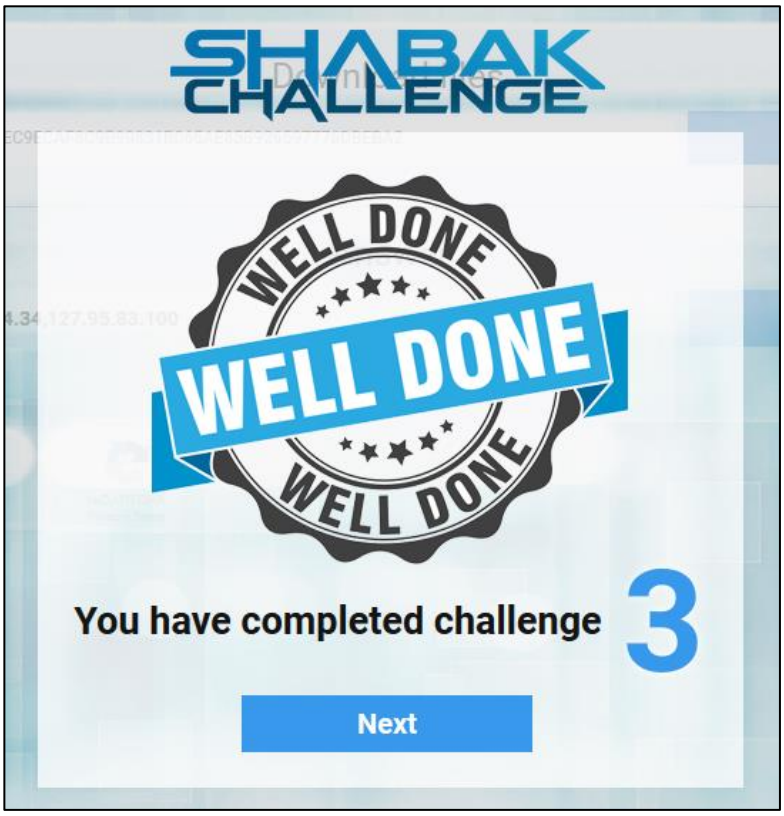
```
print (",".join(sorted(ips_for_new_suspects,
        key=lambda ip: socket.inet_aton(ip))))
```

הסקריפט מייצר רשימה של משתמשים רגילים שהשתמשו בכתובות IP של חשודים, ומתוכם שולף את שלושת המשתמשים שהשתמשו בכמות הגדולה ביותר של כתובות חשודות.

הוא מייצג את התוצאה בצורה ממויינת לפי כתובת ה-IP הנפוצה ביותר של כל משתמש, כפי שהאתגר דרש.

```
התוצאה:Log:    Suspects: ['2449', '6796', '9237', '4024', '3538', '3608',
'7239', '435', '5206', '2211']
Log:    Cache file missing, loading from scratch
Log:    Parsing log file...
Log:    Parsed 10000000 rows in 32.40831255912781 seconds
Log:    Number of unique users: 10001
Log:    Number of unique IPs: 10010
Log:    Calculating most common IP per user...
Log:    Searching for all suspect IPs...
Log:    All suspect IPs: {'104.45.191.227', '103.205.114.34',
'139.210.78.22', '127.95.83.100', '130.76.88.3', '51.70.255.188',
'124.9.188.155', '114.79.247.223', '41.239.144.6', '109.242.247.39',
'78.12.11.167', '211.104.116.62', '58.149.209.97', '138.27.249.121',
'67.141.120.237', '10.192.20.173', '162.219.33.114', '42.74.74.110',
'143.204.212.207', '68.17.81.83', '230.167.210.226'}
Log:    Non-suspects using suspect IPs: defaultdict(<class 'int'>,
{'7159': 8, '5630': 1, '9091': 3, '1808': 10, '5772': 9, '5590': 4,
'7051': 6, '6529': 5, '4918': 10, '87': 2, '9482': 7})
Log:    Fetching top 3 new suspects:
New suspect: 1808, # suspicious IPs: 10, Most common IP: 41.239.144.6
New suspect: 4918, # suspicious IPs: 10, Most common IP: 103.205.114.34
New suspect: 5772, # suspicious IPs: 9, Most common IP: 127.95.83.100
41.239.144.6,103.205.114.34,127.95.83.100
```

ואכן, אלו החשודים שהשב"כ חיפש:



והתוצר הסופי:

מילות סיכום

באופן כללי, מדובר ביוזמה מבורכת שתופסת תאוצה בשנים האחרונות, ואנחנו מקווים להמשיך ולראות עוד אתגרים בשנים הקרובות, כחלק מקמפיין גיוס או מכל סיבה אחרת.

בסך הכל, נהנינו לפתור את שני המסלולים של סדרת האתגרים. שני המסלולים האחרים: Hardware ו-Signal Processing, הצריכו מומחיות אחרת לחלוטין ולמעשה הפנייה הזו למספר קהלים היא נקודה נוספת לחיוב.

מהצד השני, היו מספר עניינים אשר פגמו מעט בחוויה - למשל, העובדה שהיה צריך להוסיף רווחים ב-Find the Code, או ההתעקשות על רמת דיוק של 5 ספרות אחרי הנקודה ב-Cat and Mouse (כנראה שבשלב מסוים נוספה הבהרה בשדה התשובה בתרגיל). ועדיין, חלק מהתרגילים הצריכו חשיבה מחוץ לקופסא, וחלק דימו בצורה יחסית מדויקת (כנראה) עבודת מודיעין אמיתית.

התחלנו עם מסלול ה-Embedded כי בכך אנחנו עוסקים ביומיום, והצלחנו לפתור את שתי השאלות ביום רביעי בלילה. ביום חמישי נתקלנו בכתבה הזו במאקו:

> שירות הביטחון הכללי (שב"כ) השיק אתמול (רביעי) קמפיין גיוס חדש, שנועד למצוא את אנשי הטכנולוגיה המוכשרים ביותר להצטרף לשורותיו. 150,000 גולשים כבר נכנסו לאתר של האתגר: גולשים רבים ממדינות שונות ניסו להתמודד עם האתגר - ונכון לשעה 10:00 הבוקר (חמישי) רק שניים הצליחו לפתור את כל שלבי האתגר.
>
> לסיפורים הכי מעניינים והכי חמים – הצטרפו לפייסבוק שלנו
>
> גולשים מארה"ב, אנגליה, קנדה, צרפת ורוסיה ניסו לפתור את האתגר - וכך גם משתמשים מעזה, אפגניסטן, טורקיה, איחוד האמירויות, מלזיה, עומאן, קירגיזסטן, מצרים, ערב הסעודית, מרוקו, אינדונזיה, פקיסטן ועירק.
>
> השניים שפתרו את כל השלבים של האתגר הצליחו באחד המסלולים, שנקרא תוכנה משובצת. באתר יש עוד שלושה מסלולים של האתגר שאף אחד עדיין לא הצליח לפתור. כמה אלפים בודדים הצליחו לפתור רק את השלבים הראשונים של האתגר בקטגוריות השונות, ומאות שלחו קורות חיים והגישו מועמדות למשרות הטכנולוגיות השונות.
>
> באתר www.israelneedsu.com ("ישראל צריכה אותך") נמצא האתגר, שנכתב על-ידי מומחים ומהנדסים של השב"כ. מדובר בחידה מורכבת הבנויה ממספר שלבים ומשלבת כמה אלמנטים טכנולוגיים מתקדמים, הנוגעים לתחומי הפעילות של גוף הביטחון.

היה נחמד מאוד לראות שהיינו הראשונים לפתור מסלול כלשהו, ולזכות ברבע שעה של תהילה אנונימית.