Digital Whisper Technologic Papers

פתרון אתגר "Practical Car Hacking" CTF

מאת לירן אטליס

הקדמה

לפני כשבוע נתקלתי בפוסט בלינקדאין של יוצר ה-CTF, ושמתי לב שחלק משמעותי מהאתגר עוסק בתחום שמאוד מעניין אותי - תחום ה-CRF (Cyber RF).

בעשור האחרון, ובעיקר מאז עלייתם של איומי הרחפנים, תחום זה חווה צמיחה מואצת.

מעבר לכך, נעשה בו שימוש נרחב גם בתחומי ה-IoT, Bluetooth, רכבים חכמים, רשתות Wi-Fi ותקשורת סלולרית - למעשה, בכל מכשיר שמעביר מידע באמצעות גלי רדיו.

קישור לאתגר

אתגר Volkswagen CAN Checksum

באתגר זה, נתונות 15 הודעות, כאשר כל הודעה מורכבת מבית ראשון (8 ביטים ב-MSB) המשמש כ-Checksum, ולאחריו רצף ביטים המהווים את תוכן ההודעה עצמה. יחד עם הנתונים נמסרה גם שיטת חישוב ה-Checksum:

- מחושב ע"י אלגוריתם בשם Autosar CRC8 8H2F (עם מידע נוסף ב-PDF)
- הקלט לאלגוריתם ה-Checksum מורכב מתוכן ההודעה (ללא ה-CRC כמובן), בתוספת בית נוסף "סודי" שאותו יש לגלות על מנת לחשב את ה-Checksum של ההודעה.

תחילה, שקלתי לנסות ולחשב את הבית הסודי באופן ידני. עם זאת, די מהר הבנתי כי ניתן לפצח אותו בצורה פשוטה ויעילה באמצעות Brute Force.

התחלתי מלעיין ב-PDF של אלגוריתם החתימה במטרה להבין את מאפייני אלגוריתם ה-CRC - בדגש על הפולינום והערך ההתחלתי בהם נעשה שימוש. מתוך המידע שסופק ניתן היה להסיק כי הפרמטרים של האלגוריתם הם:

- ערך התחלתי: 0xff
- פולינום: 0x2F

CRC result width:	8 bits
Polynomial:	2Fh
Initial value:	FFh
Input data reflected:	No
Result data reflected:	No
XOR value:	FFh
Check:	DFh
Magic check:	42h

Table 7.3: CRC8 Polynomial

כעת, עברתי לכתיבת קוד ב-Python, שהתבסס על מקטע הקוד שסופק כחלק מהאתגר. במסגרת הקוד, יצרתי 2 לולאות:

1. רצה על 15 ההודעות הנתונות
2. מחשבת CRC לכל הודעה באמצעות ריצה על 256 ערכים אפשריים לבית ה"סודי" (בית הוא 8 ביטים: $2^8 = 256$) , עם תנאי עצירה כאשר חישוב ה-CRC מניב תוצאה זהה ל-CRC המקורי של ההודעה הנתונה.

כצפוי, התקבל אותו ערך של הבית הסודי עבור כל אחת מ-15 ההודעות - בהתאם להערה שניתנה באתגר, שלפיה במקרה הספציפי הזה הבית הסודי הוא קבוע. זאת בניגוד למצבים אחרים שבהם ערך הבית הסודי תלוי ב-counter משתנה.

לאחר מציאת הבית הסודי, המשכתי לחישוב ה-CRC עבור הודעת האתגר הסופית, תוך שילוב הבית הסודי - ובכך למעשה הושלם פתרון החלק הראשון של האתגר.

```
guessed secret bytes are (in int): [195, 195, 195, 195, 195, 195, 195, 195, 195, 195, 195, 195, 195, 195, 195]
so in hex the secret byte is:  0xc3
checksum is: 0x35
```

קישור לקוד הפתרון

אתגר 1 Hitag2 Keyfob ID - Part

באתגר זה, נתונה הקלטת RF של 6 לחיצות עוקבות על כפתור בשלט של רכב, כאשר נתון שהשלט של הרכב הוא מסוג (RKE Remote Keyless Entry), ומשתמש בפרוטוקול הצפנה בשם Hitag2. במשימה עלינו לפענח את השידור ולחלץ ממנו את הביטים של שדה ה-UID של השלט, בהתאם ל-documentation שנתון באתגר.

לפי המסמך הנתון, חיפשתי את החלקים הקשורים לפרוטוקול Hitag2, ומצאתי את התרשים הבא:

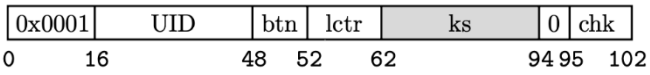
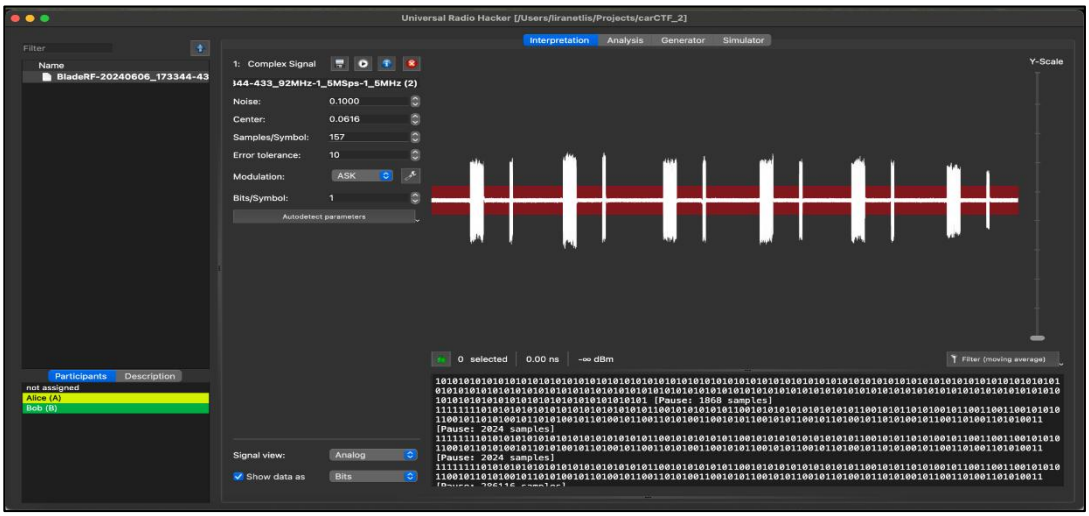


Figure 11: Packet structure of a rolling code for Hitag2. Gray background indicates the keystream part produced by the cipher.

מהתרשים אפשר להסיק שיהיה ניתן לזהות "בקלות" את שדה ה-UID, כי הוא יופיע לאחר 2 הבתים "0x0001". בנוסף, לאורך המאמר רשום שלרוב נעשה שימוש בקידוד Manchester - שזהו סוג של קידוד ביטים שמשלב את השעון הפנימי, כך שהאות הדיגיטלי יהיה מסונכרן לשעון (מה שמקל על הסנכרון של האות הנקלט), בתוספת של תרומה לזיהוי שגיאות בסיסי.

מכאן, אגש לפענוח האות בעזרת תוכנת URH הנתונה באתגר (Universal Radio Hacker), שהיא בעצם כלי להצגה, פענוח של אותות ופרוטוקולי תקשורת, Fuzzing ושידור מחדש של האות.

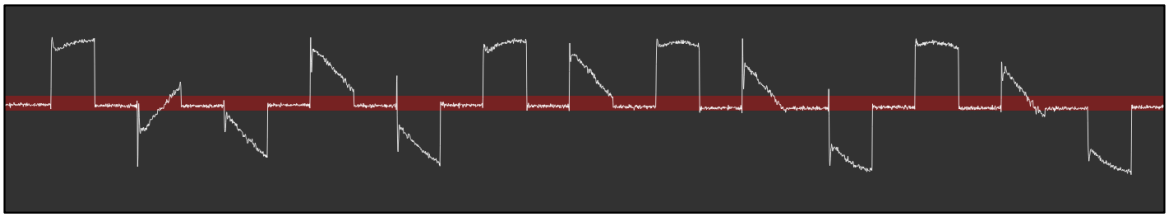
כשנפתח את הקובץ בתוכנה, נראה את התצוגה הבאה:



ניתן לראות כי ישנם אכן 6 אותות RF בזמן שמשודרים אחד אחרי השני, כאשר כל שידור מורכב מחלק "רחב" יותר וחלק "צר" שמגיע מיד אחריו.

כעת, יש למצוא את הפרמטרים האופטימליים לפענוח האות לביטים (אודה בשלב זה שרוב התמונות צולמו אחרי שכבר מצאתי את כלל הפרמטרים ופתרתי את האתגר).

אם נסתכל בהגדלה על אחד האותות, נראה שהגל נראה כך:

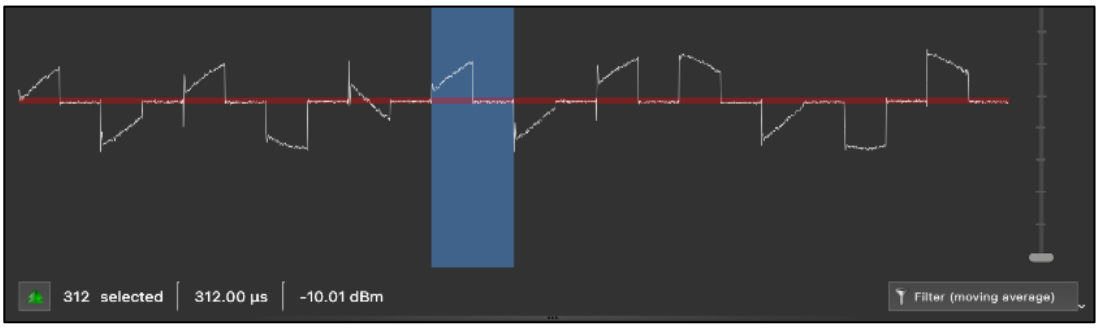


ממנו, ניתן להסיק כי הגל הוא מאופנן ב-ASK, היות ונראה שהתדר נראה זהה לאורך כלל השידור (ולכן זה לא אפנון FSK). אגיד כי לדעתי היה יותר קל להעביר את הקובץ דרך GNU Radio לפני כן, ושם לבצע ערך מוחלט על כל האות לטובת פענוח קל יותר, אבל נצמדתי ל-URH כפי שצויין באתגר.

הפרמטרים בתוכנה הם:

1. רעש (Noise) - פרמטר הקובע את ה"רוחב" של הפס האדום, שכל עוצמה של האות מתחתיו או מעליו תיחשב כ-'1', וכל מה שבתוכו כ-'0'. בתוכנה ניתן לראות אותו כפס אדום, שמתרחב ככל שמגדילים את ערך ה-Noise.
2. מרכז (Center) - פרמטר הקובע את מיקום הפס האדום של הרעש ביחס לאות עצמו.
3. דגימות לסימבול (Samples/Symbol) - פרמטר הקובע כמה דגימות מהאות עצמו יחושבו לכדי סימבול יחיד. ארחיב על פרמטר זה בהמשך.
4. סבילות לשגיאות (Error Tolerance) - פרמטר הקובע את מרווח השגיאה שלנו ביחס לפרמטר הדגימות לסימבול שקבענו. אם לדוגמא קבענו כי נרצה 10 דגימות לסימבול, אז אם נקבע את פרמטר Error Tolerance להיות 2, אז התוכנה תוכל לסטות 2 דגימות יותר או פחות לסימבול.
5. ביטים לסימבול (Bits/Symbol) - פרמטר המתייחס לאפנון, וקובע כמה ביטים מיוצגים ע"י סימבול יחיד.

אחרי שקבענו כי המודולציה היא מסוג ASK, נשתמש ב-Autodetect parameters כדי לתת לתוכנה לקבוע את הפרמטרים אוטומטית ונשנה כל פרמטר בהתאם. לאחר הפעלת Autodetect parameter הסתכלתי על האות וקבעתי בעצמי את ערך ה-Samples/Symbol בצורה הבאה:



ניתן לראות שסימנתי באות 2 סימבולים רצופים שהם בעצם שווי ערך ל-'10', ובהסתכלות בצד שמאל, רואים שמסומנות 312 דגימות. אם נחלק את הערך הזה ב-2 נקבל 166 דגימות, את ערך זה קבעתי להיות הערך של Samples/Symbol.

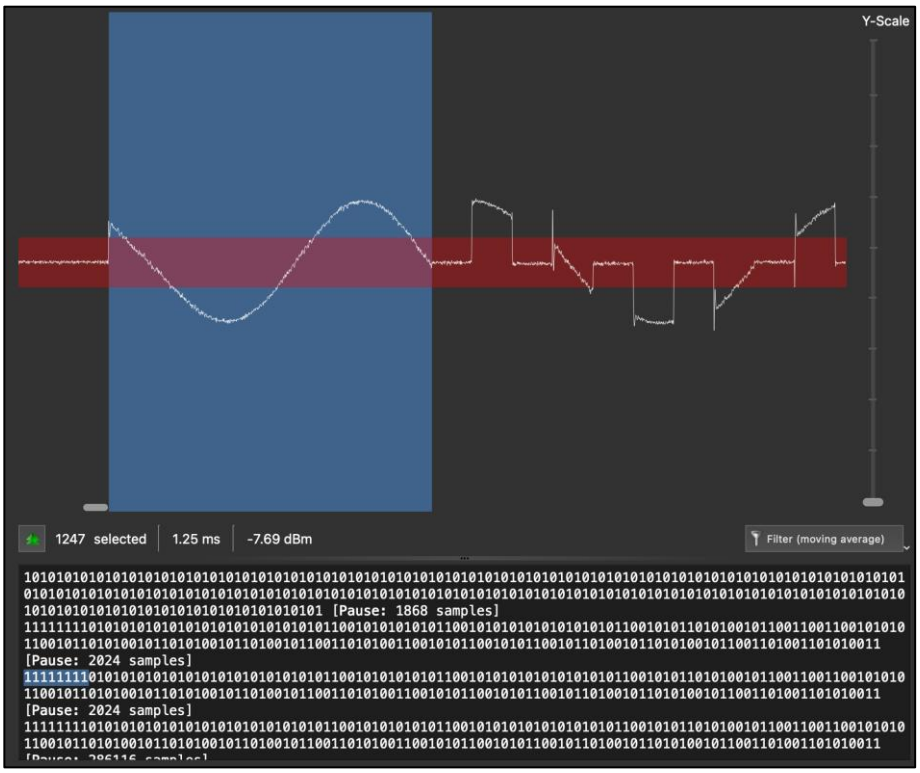
סימנתי בכוונה 2 דגימות רצופות כדי לקבל "מיצוע" לערך, ולא לשפוט על פי דגימה בודדת, וכך לקבל דיוק טוב יותר. בהמשך הניתוח, שיניתי את ערך זה להיות 167 כי הוא עבד יותר טוב (לבסוף, זה עניין של כמה מדוייק מסמנים בתוכנה מחזור שלם של ביט).

מכאן החלטתי שאני אתייחס למתחי האות כ-'1' או '0', ואתייחס לערכי המתח השליליים גם כ-1. כי ברוב המאמרים שלטי רכב הם באפנון OOK (אפנון אמפליטודה של '1' או '0'), אז קבעתי את ערך Bits/Symbol ל-1, מה שמפשט יותר את הפענוח.

כעת, נביט בחלק התחתון של הביטים המפוענחים (נסמן Show data as: Bits), ונוכל לשים לב לכמה תופעות:

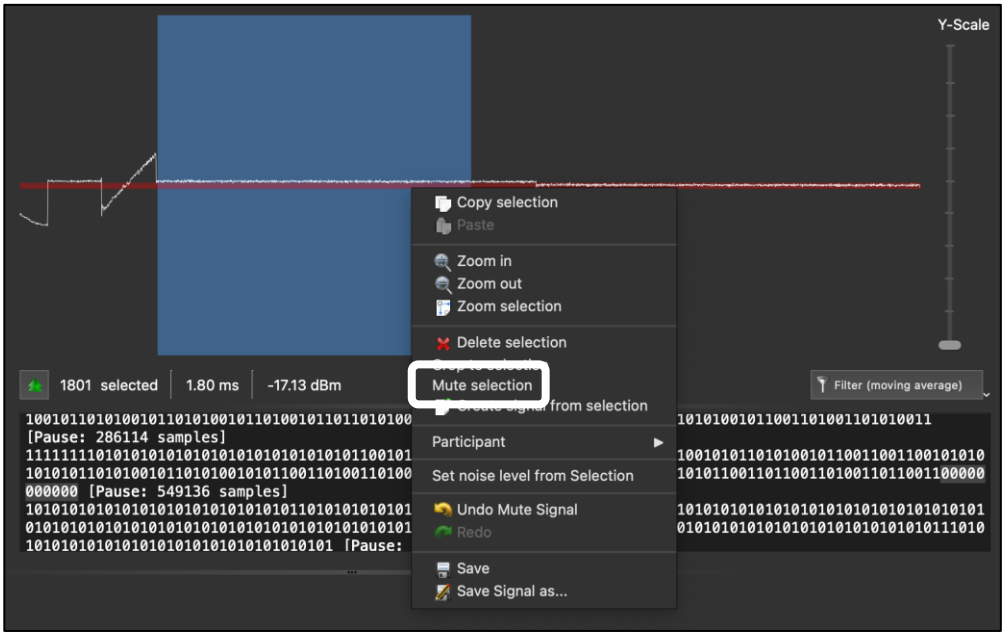
כל "לחיצת כפתור" בשלט מתורגמת ל-סדרה של 5 פקטות, כאשר:

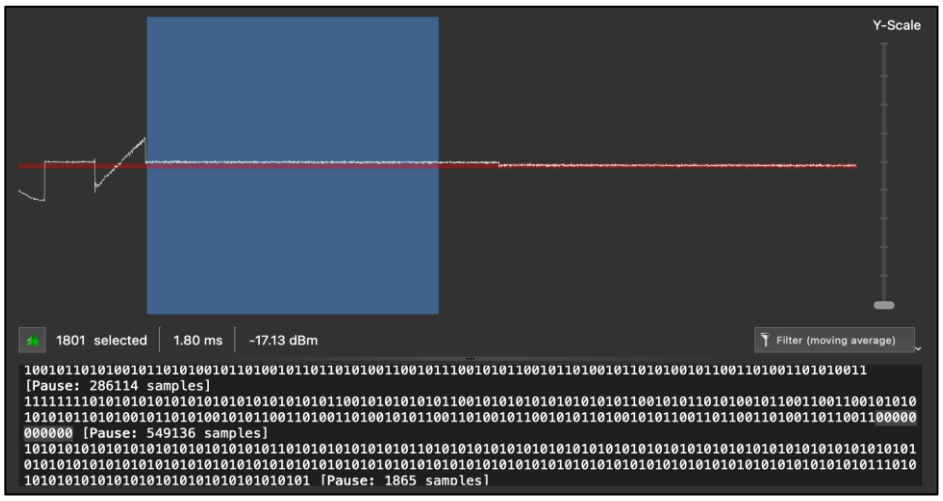
- הפקטה הראשונה הינה למעשה ה-preamble, כי מדובר ברצף של 0xaaaa והיא נועדה לסמן שעכשיו יגיעו פקטות נוספות. רצף זה של 0 ו-1 נועד כדי שהמקלט יסנן את האות טוב יותר ובמקביל יסתנכרן לקבלת פקטות.
- לאחר פקטת ה-preamble, ניתן לראות שכל פקטה מתחילה עם גל סינוס, שמתורגם ל-'11111111', ולאחריו מתחיל באמת זרם המידע של הפקטה.



- לאחר מכן, מגיעות 3 פקטות עם תוכן זהה (ניתן לראות זאת לאחר טיוב שאר הפרמטרים) - רואים את זה טוב יותר כשמסמנים (Show data as: Hex):

אם לא רואים את זה טוב, סימן שהפרמטרים לא טוייבו עד הסוף, אז ממליץ לנסות את הפרמטרים שמופיעים בתמונה הראשונה. לפעמים פרמטרים אלו לא מספיקים כי התוכנה לא מעבדת טוב את סיום השידור ומוסיפה אפסים. כדי לתקן את זה, ניתן לעשות "Mute" לחלקים אלו. מצורפות 2 דוגמאות ל-'0' במקומות שבהם הפקטה אמורה להסתיים:





- בשלב זה, לאחר שטייבנו את כלל הפרמטרים, אנחנו באמת רואים שהשידורים נראים בצורה הזו:

```
aaaa
```

אתגר Hitag2 Keyfob ID - Part 2

אתגר זה הוא למעשה המשך של האתגר הקודם והוא מסתמך על הפענוח שביצענו בו, כאשר הפעם המטרה היא לפענח את המפתח של Hitag2. מהאתגר עצמו, נתון לנו הקוד שבו נשתמש לטובת הפענוח, ונאמר שאפשר להשתמש ב-crack5 (שמשתמש ב-CPU) או ב-crack5opencl. אני אישית בחרתי להשתמש ב-crack5.

לטובת הפתרון, כתבתי בפייתון מערך שלוקח את כל פקטות השידור (חוץ מאלו שהם רצף 0xaaaa), לאחר שהן עברו פענוח מנצ'סטר, והחסרתי מהן את השדה ההתחלתי של 0x0001. משם, הקוד מבצע parse לכל פקטה, מחלץ את כלל השדות ומוסיף 2 שדות חדשים לפי ההנחיות הרשומות באתגר:

1. שדה ks inverted - רשום שקוד הפענוח מצפה לקבל את שדה ה-ks אבל inverted - כלומר, עלינו לקחת את אותו שדה ולבצע עליו XOR עם 0xffffffff. שדה זה הוא למעשה שדה ה-aR בהתאמה לכתוב באתגר.
2. שדה nR - מורכב לפי הכתוב באתגר:
   a. 18 ביטים ב-MSB שהם '0'
   b. 10 ביטים משדה ה-counter מהפקטה עצמה
   c. 4 ביטים של שדה ה-button מהפקטה עצמה

כעת נבחר 2 זוגות של ערכי nR, aR, (אני אישית לקחתי מהשידור הראשון ואחד לפני אחרון, כי כתוב במאמר שבירת ההצפנה שעדיף פקטות לא "עוקבות", כדי שתהיה כמה שפחות קורלציה בין הפקטות) את ה-UID, ונזין לקוד שבירת ההצפנה:

```
./ht2crack5 0x0200472A 0x00000EC1 0xCCA437E3 0x00000E71 0xCD177663
```

לאחר הרצה והמתנה קצרה נקבל:

```
Thread 5 slice 1/256
Thread 2 slice 1/256
Thread 6 slice 1/256
Thread 3 slice 1/256
Thread 7 slice 1/256
Thread 1 slice 1/256
Thread 2 slice 2/256
Thread 7 slice 2/256
Thread 3 slice 2/256
Thread 4 slice 2/256
Thread 1 slice 2/256
Thread 6 slice 2/256
Thread 5 slice 2/256
Thread 0 slice 2/256
Key: 72B7C3CCE726
```

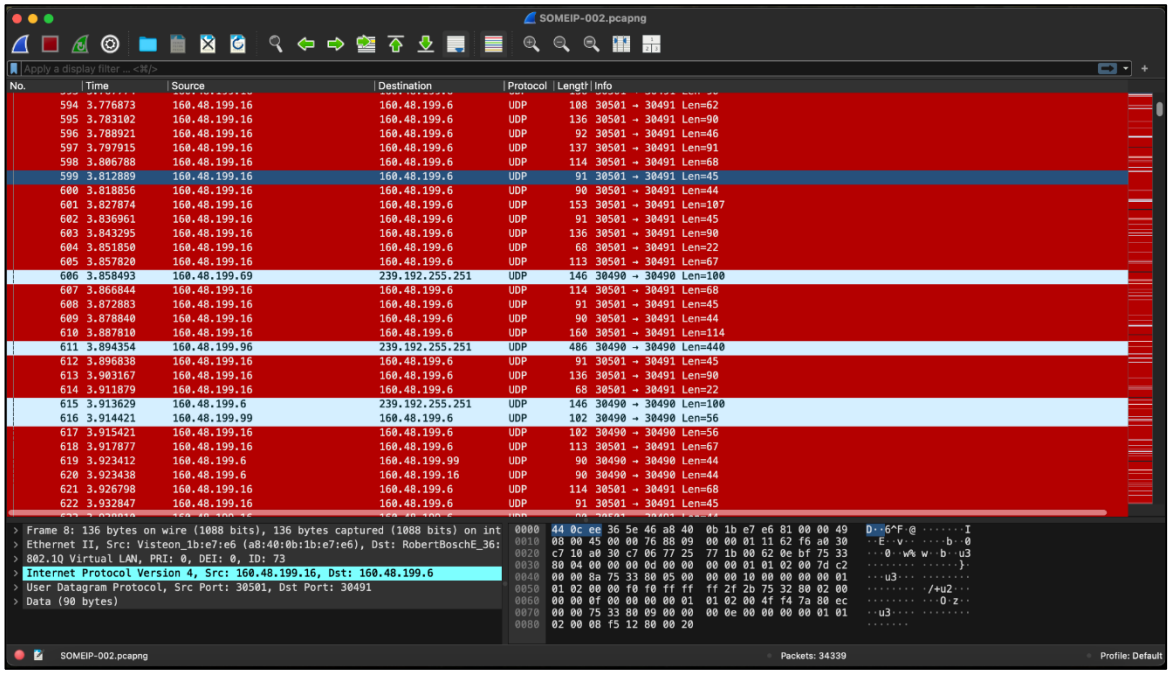
וזהו הפתרון לאתגר :) קישור לקוד הפתרון

אתגר AVTP Video

באתגר זה נתון קובץ מסוג pcapng עם הקלטה של תעבורת אינטרנט שהוקלטה מרכב BMW G30. ההקלטה בוצעה באמצעות האזנה על חיבור האינטרנט שבין ה-BDC (יחידת gateway המקשרת בין כל מערכות הרכב כמו נעילה, חלונות, אזעקה ועוד) לבין מצלמת הרוורס.

נתון עוד כי בקובץ נראה תחילה תעבורה של פקטות מסוג SOME/IP. כאשר הרכב עובר להילוך אחורי, מוזרם ווידאו לצד תעבורת ה-SOME/IP. מטרת האתגר היא לפענח את זרם הווידאו הזה. מצלמת הרוורס מכוונת אל דף נייר שעליו מופיע הדגל (flag).

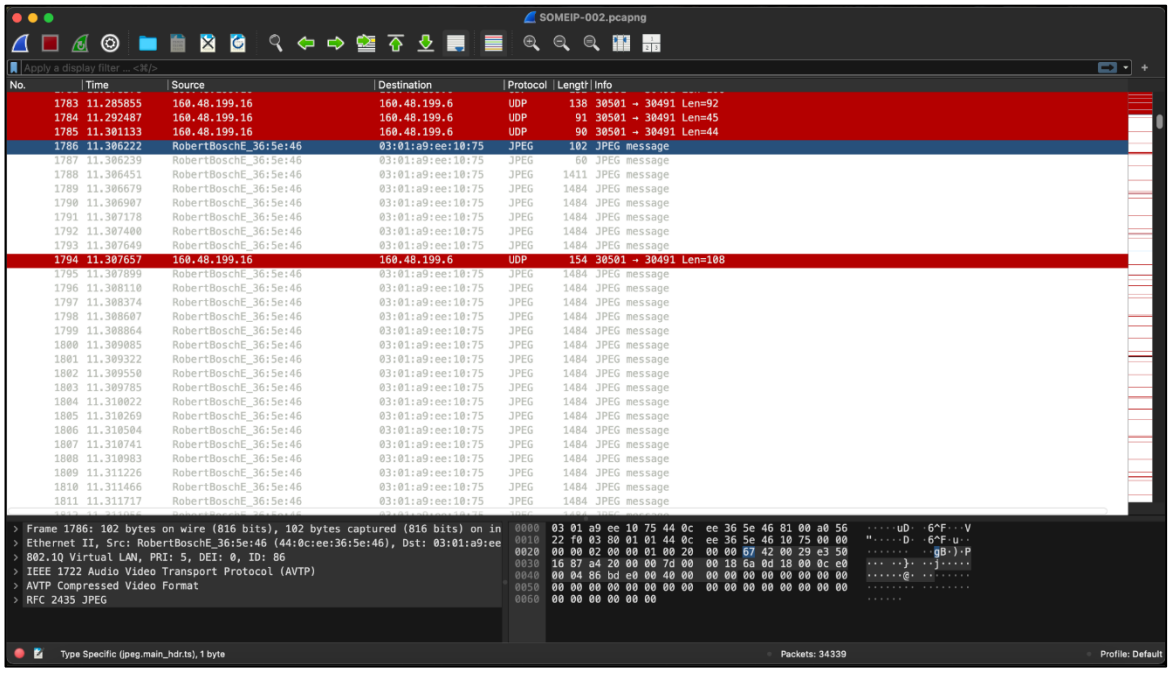
התחלתי מלפתוח את הקובץ ב-wireshark:



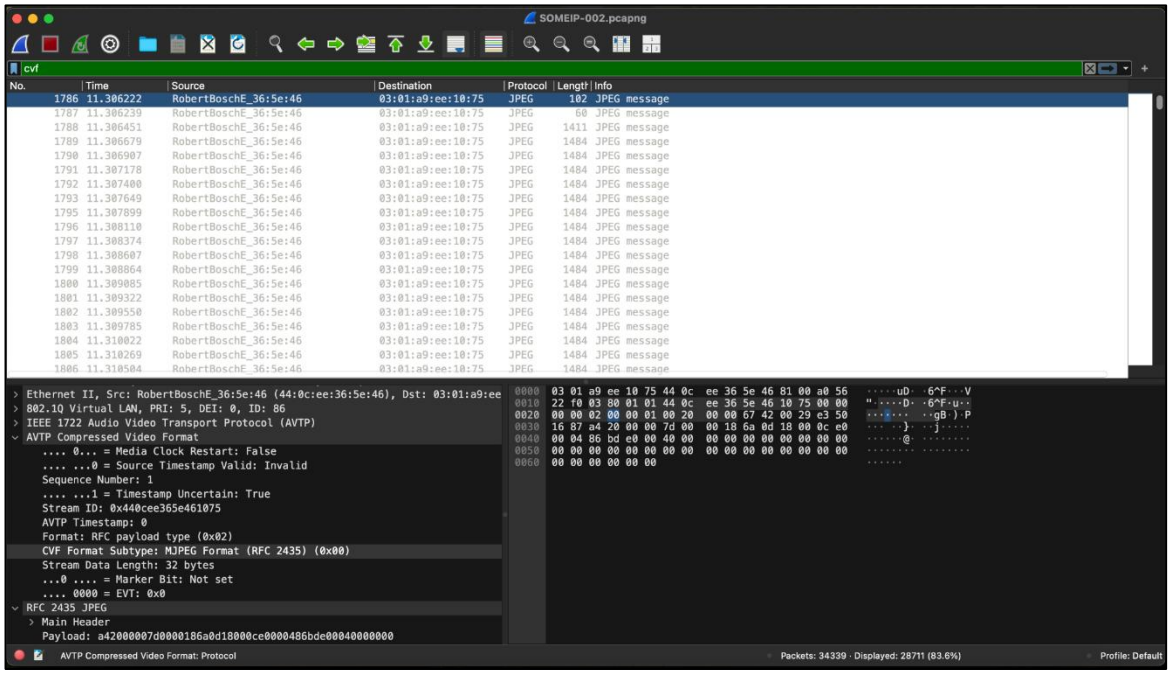
מבדיקה קצרה באינטרנט, גיליתי שפרוטוקול SOME/IP הוא פרוטוקול תקשורת לרכבים שתומך בהודעה על "אירועים" וקריאה לתהליכים אחרים.

כלומר, הגיוני שתעבורה זו תהיה מסוג UDP כמו שאנחנו באמת רואים בתמונה. קישור למידע (עמוד 5)

דפדפתי ובהמשך ראיתי שבפקטה מספר 1786 (11.306222 שניות לאחר תחילת ההקלטה), מופיעות פקטות שרשום בהן "JPEG" בשדה הפרוטוקול:



מכאן רציתי כבר לסנן שאראה רק את הפקטות הקשורות להעברת הווידאו, לכן תחילה שקלתי לעשות זאת לפי סינון של כתובות IP של מקור + יעד, אבל אם מסתכלים על הפקטה, רואים שרשום שהיא מסוג Compressed Video Format. חיפשתי באינטרנט ומצאתי שאפשר לסנן בעזרת מילת המפתח "cvf" וזה אכן סינן:



ניסיתי להסתכל על כל פקטה בנפרד ולראות מאפיינים דומים, וראיתי כמה תופעות:

1. כל השדות ב-wireshark עד השדה "Type Specific" **כולל,** הגיוניים מבחינת ערכים (אלה שגדלים ב-1, אכן מתנהגים כך, אלו שקבועים נשארים קבועים ועוד).

```
AVTP Compressed Video Format
    .... 0... = Media Clock Restart: False
    .... ...0 = Source Timestamp Valid: Invalid
    Sequence Number: 1
    .... ...1 = Timestamp Uncertain: True
    Stream ID: 0x440cee365e461075
    AVTP Timestamp: 0
    Format: RFC payload type (0x02)
    CVF Format Subtype: MJPEG Format (RFC 2435) (0x00)
    Stream Data Length: 32 bytes
    ...0 .... = Marker Bit: Not set
    .... 0000 = EVT: 0x0
RFC 2435 JPEG
  Main Header
      Type Specific: Unspecified (103)
      Fragment Offset: 4325417
      Type: Dynamically assigned (227)
```

פרט לשדות EVT, Marker Bit שהם לא מתנהגים לפי האפיון של פרוטוקול RFC 2435, שבו רשום שהשדה מרקר ביט אמור להיות '1' כל סוף פקטה, ואין להתייחס לשדה EVT. במקרה שלנו, ה- Marker Bit תמיד '0' בכל הפקטות, בעוד שדה EVT הוא לרוב 0x0 פרט לפקטה אחרונה של כל שידור, ובה הוא מקבל את הערך 0x4, מה שנתן לי קצה חוט להתעסק עם פרמטרים אלו, אפרט בהמשך.

2. השדה Fragment Offset וכל מה שבא אחריו אינם עקביים - הערך שלו משתנה לחלוטין בכל פקטה ללא הקשר של ערכים עולים/יורדים - מה שמרמז על parsing לא נכון של הפקטה, במיוחד לאור מה שרשמתי בסעיף הקודם על שדות EVT, Marker Bit.

ניסיתי לטפל בשדות EVT, Marker Bit ו-Fragment Offset בתקווה שאם אפתור את הבעיה איתם זה יפתור את כל השאר. ניסיתי בעזרת קוד למחוק את 2 הביטים הראשונים החל מסיום שדה "Stream Data Length" והוספתי 2 ביטים בסוף הפקטה במקום. פתרון זה אכן טיפל בבעיה וה-Marker Bit קיבל '1' בכל סיום פקטה, אבל השדה Fragment Offset עדיין לא היה הגיוני.

בנוסף לכך, השדה "Type Specific" הוא לרוב מסוג "Unspecified". שילוב כל ההבחנות האלו גרם לי לחשוב שכנראה לא כדאי לי להסתמך על ה-Parsing של wireshark.

אם נסתכל על הפקטות:

פקטה ראשונה:

```
0000   03 01 a9 ee 10 75 44 0c ee 36 5e 46 81 00 a0 56
0010   22 f0 03 80 01 01 44 0c ee 36 5e 46 10 75 00 00
0020   00 00 02 00 00 01 00 20 00 00 67 42 00 29 e3 50
0030   16 87 a4 20 00 00 7d 00 00 18 6a 0d 18 00 0c e0
0040   00 04 86 bd e0 00 40 00 00 00 00 00 00 00 00 00
0050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060   00 00 00 00 00 00
```

פקטה שנייה:

```
0000   03 01 a9 ee 10 75 44 0c ee 36 5e 46 81 00 a0 56
0010   22 f0 03 80 02 01 44 0c ee 36 5e 46 10 75 00 00
0020   00 00 02 00 00 01 00 04 00 00 68 ce 3c 80 00 00
0030   00 00 00 00 00 00 00 00 00 00 00 00
```

פקטה שלישית (חתוכה בגלל כמות גדולה של מידע):

```
0000   03 01 a9 ee 10 75 44 0c ee 36 5e 46 81 00 a0 56
0010   22 f0 03 80 03 01 44 0c ee 36 5e 46 10 75 00 00
0020   00 00 02 00 00 01 05 59 00 00 7c 85 88 84 00 00
0030   6b 7b e9 8f f2 c1 90 00 10 12 f4 45 23 c8 a5 d8
0040   df 27 4e 07 ed 29 f1 c0 0c 76 23 77 9f 15 de 7c
0050   0c f4 97 f1 fc 0f 03 c1 2d 57 1f 7f 81 e9 13 a4
0060   2d bf 7c 65 7f fe 9f 82 3d 7c ff ff fc 2d d6 fe
```

פקטה מספר 1889 (חתוכה בגלל כמות גדולה של מידע):

```
0000   03 01 a9 ee 10 75 44 0c ee 36 5e 46 81 00 a0 56
0010   22 f0 03 80 60 01 44 0c ee 36 5e 46 10 75 01 0f
0020   d0 ab 02 00 00 01 05 92 00 00 5c 81 88 88 00 10
0030   26 b1 07 7a ef 8a 86 58 0a 45 00 00 40 fa 8b 8e
0040   07 4c 0a 82 f1 58 47 6a 39 70 55 bb 6f f3 ab 78
0050   01 51 cf c6 80 01 e8 c0 05 78 fc 17 f3 77 77 7f
0060   a8 da 1d f0 e1 c4 70 c1 15 f7 22 3a f5 e3 e7 84
```

נשים לב כי בית מספר 0x2a (המסומן בצהוב), הוא תמיד יהיה עם אחד מהערכים הבאים: 0x67, 0x7c, 0x68, 0x5c. על-כן, כל הבתים עד לאותו בית דומים מאוד אחד לשני.

לאחר חיפוש קצר באינטרנט מצאתי שהערכים 0x68, 0x67 הם ערכים שקשורים ל-NAL (Network Abstraction Layer) של שידור ווידאו בפורמט H.264:

```
1.3.  Network Abstraction Layer Unit Types

   Tutorial information on the NAL design can be found in [12], [13],
   and [14].

   All NAL units consist of a single NAL unit type octet, which also
   co-serves as the payload header of this RTP payload format.  The
   payload of a NAL unit follows immediately.

   The syntax and semantics of the NAL unit type octet are specified in
   [1], but the essential properties of the NAL unit type octet are
   summarized below.  The NAL unit type octet has the following format:

      +---------------+
      |0|1|2|3|4|5|6|7|
      +-+-+-+-+-+-+-+-+
      |F|NRI|  Type   |
      +---------------+

   The semantics of the components of the NAL unit type octet, as
   specified in the H.264 specification, are described briefly below.

   F: 1 bit
      forbidden_zero_bit.  The H.264 specification declares a value of
      1 as a syntax violation.

   NRI: 2 bits
      nal_ref_idc.  A value of 00 indicates that the content of the NAL
      unit is not used to reconstruct reference pictures for inter
      picture prediction.  Such NAL units can be discarded without
      risking the integrity of the reference pictures.  Values greater
      than 00 indicate that the decoding of the NAL unit is required to
      maintain the integrity of the reference pictures.

   Type: 5 bits
      nal_unit_type.  This component specifies the NAL unit payload type
      as defined in table 7-1 of [1], and later within this memo.  For a
      reference of all currently defined NAL unit types and their
      semantics, please refer to section 7.4.1 in [1].
```

שזה מעודד ואומר שאני בכיוון, כי לא מצאתי Header-ים שמתאימים לפורמט JPEG (xffd80 או xffd90).
נשים גם לב שתמיד 2 הפקטות של תחילת שידור חדש יהיו תחילה עם 0x67, לאחר מכן 0x68 ולאחריה פקטות מידע עם 0x7c או 0x5c בהמשך. אין ברירה, ניתחתי קצת את המבנה של בתים אלו:

Type (decimal)	Type (binary)	NRI	F	ערך בינארי	ערך בבסיס hex
7	00111	11	0	01100111	**0x67**
8	01000	11	0	01101000	**0x68**
28	11100	10	0	01011100	**0x5c**
28	11100	11	0	01111100	**0x7c**

ערך ה-NRI משפיע לבסוף האם התמונה היא מסוג IDR או non-IDR (משפיע על הבית שנוסיף בהמשך לקובץ שנכתוב), ומבדיקה באפיון, מצאתי כי ערך '11' מתאים לתמונה מסוג IDR, ו-'10' מתאים ל-non-IDR בהתאם.

נשאר רק להבין מה המשמעות של כל סוג בית - אם נחזור שוב לאותו אפיון נראה את הטבלה הבאה:

```
Fragmentation unit: Used to fragment a single NAL unit over multiple
RTP packets.  Exists with two versions, FU-A and FU-B, identified
with the NAL unit type numbers 28 and 29, respectively.  Specified in
section 5.8.

Table 1.  Summary of NAL unit types and their payload structures

   Type   Packet    Type name                        Section
   ---------------------------------------------------------
   0      undefined                                     -
   1-23   NAL unit  Single NAL unit packet per H.264   5.6
   24     STAP-A    Single-time aggregation packet     5.7.1
   25     STAP-B    Single-time aggregation packet     5.7.1
   26     MTAP16    Multi-time aggregation packet      5.7.2
   27     MTAP24    Multi-time aggregation packet      5.7.2
   28     FU-A      Fragmentation unit                 5.8
   29     FU-B      Fragmentation unit                 5.8
   30-31  undefined                                    -
```

אפשר להבין שסוג 28 אלו פקטות שהן חלקים מהפריים עצמו, בעוד שסוגים 7,8 אמורים להיות פקטות "עצמאיות".

עכשיו, אם נסתכל על הבית הבא, לפי האפיון הוא אמור להיראות כך:

```
   The FU header has the following format:

      +---------------+
      |0|1|2|3|4|5|6|7|
      +-+-+-+-+-+-+-+-+
      |S|E|R|  Type   |
      +---------------+



Wenger, et al.              Standards Track                    [Page 29]

RFC 3984           RTP Payload Format for H.264 Video      February 2005


   S: 1 bit
      When set to one, the Start bit indicates the start of a fragmented
      NAL unit.  When the following FU payload is not the start of a
      fragmented NAL unit payload, the Start bit is set to zero.

   E: 1 bit
      When set to one, the End bit indicates the end of a fragmented NAL
      unit, i.e., the last byte of the payload is also the last byte of
      the fragmented NAL unit.  When the following FU payload is not the
      last fragment of a fragmented NAL unit, the End bit is set to
      zero.

   R: 1 bit
      The Reserved bit MUST be equal to 0 and MUST be ignored by the
      receiver.

   Type: 5 bits
      The NAL unit payload type as defined in table 7-1 of [1].
```

בהתאמה, אם נסתכל על אותו בית בפקטות אחרות:

- בפקטה שמתחילה ב-0x67, מופיע הערך 0x42
- בפקטה שמתחילה ב-0x68, מופיע הערך 0xce
- בפקטה שמתחילה ב-0x5c, מופיעים הערכים האפשריים: 0x81, 0x41, 0x01
- בפקטה שמתחילה ב-0x7c, מופיעים הערכים האפשריים: 0x85, 0x45, 0x05

נסיק מכאן:

- כאשר ה-MSB בבית הנ"ל הוא '1' (כאשר חצי הבית הראשון גדול יותר מ-0x8) זוהי פקטה עם התחלה של fragment
- כאשר חצי הבית הראשון הוא בין הערכים 0x4, 0x7 - זו פקטה אחרונה של אותו fragment
- כאשר חצי הבית הראשון הוא 0x0 - זו פקטה שמכילה fragment נוסף מאותו חלק שלם

אוקיי, הבנו את מבנה כל פקטה. הרעיון הוא שצריך "לבנות מחדש" כל פעם את יחידת ה-NAL בקובץ חדש, בדיוק בפורמט שמתאים לפי האפיון. כלומר, צריך בכל fragment חדש להוסיף 0x0000001, לשרשר fragment-ים עוקבים, להוסיף NAL header מתאים (במידת הצורך) ולדלל מידע תעבורתי שלא נחוץ. ניגשתי לעבודה וכתבתי קוד פייתון שעובד באופן הבא:

1. מבצע סינון של כלל הפקטות ומשאיר רק את אלו של הווידאו
2. עובר על כלל פקטות הווידאו, מכל פקטה שומר את המידע שלה לפי אורך הפקטה שרשום בה, וחותך את כלל הבתים בהתחלה עד בית מספר 0x2a.
3. כעת נבחן לפי הבית הראשון:

a. 0x67, 0x68 - זוהי פקטה עצמאית. לכן, ה-payload יתחיל ב-0x0000001 ונשרשר את שאר הפקטה (כולל הבית הראשון)

b. 0x5c, 0x7c - פקטת מידע שיכולה להיות התחלה/אמצע/סוף של fragment. מה-payload עצמו, נוריד את הבית הראשון.

i. אם דגל s שווה ל-'1' זוהי התחלת fragment. במקרה זה עלינו להוסיף 0xc0000001 בתחילת ה-payload.

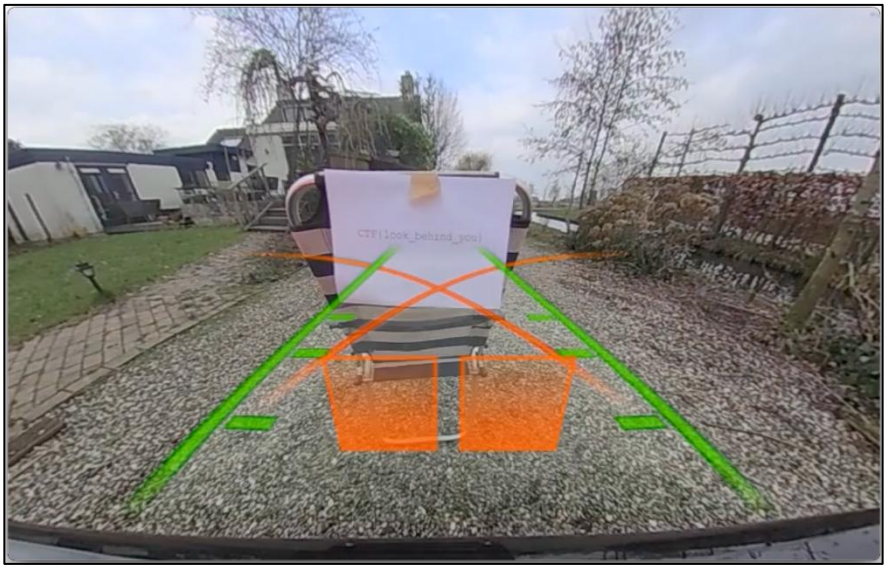
1. במקרה שבו הבית הוא 0x5c - זה מקרה שמתאים ל-NRI='10', כלומר פקטה מסוג non-IDR ועלינו להוסיף לאחר ה-0x0000001 בית עם הערך 0x41, ולאחריו את ה-payload.
2. במקרה שבו הבית הוא 0x7c - זה מקרה שמתאים ל-NRI='11', כלומר פקטה מסוג IDR ועלינו להוסיף לאחר ה-0x0000001 בית עם הערך 0x56, ולאחריו את ה-payload. את המידע לאיזה ערך בית להוסיף מצאתי בעזרת: Annex B

nal_unit_type	Content of NAL unit & RBSP syntax structure
0	Unspecified
1	Coded slice of a non-IDR picture *slice_layer_without_partitioning_rbsp()*
2	Coded slice data partition A *slice_data_partition_a_layer_rbsp()*
3	Coded slice data partition B *slice_data_partition_b_layer_rbsp()*
4	Coded slice data partition C *slice_data_partition_c_layer_rbsp()*
5	Coded slice of an IDR picture *slice_layer_without_partitioning_rbsp()*
6	Supplemental enhancement information (SEI) *sei_rbsp()*
7	Sequence parameter set *seq_parameter_set_rbsp()*
8	Picture parameter set *pic_parameter_set_rbsp()*
9	Access unit delimiter *access_unit_delimiter_rbsp()*
10	End of sequence *end_of_seq_rbsp()*
11	End of stream *end_of_stream_rbsp()*
12	Filler data *filler_data_rbsp()*
13	Sequence parameter set extension *seq_parameter_set_extension_rbsp()*
14	Prefix NAL unit *prefix_nal_unit_rbsp()*
15	Subset sequence parameter set *subset_seq_parameter_set_rbsp()*
16 - 18	Reserved
19	Coded slice of an auxiliary coded picture without partitioning *slice_layer_without_partitioning_rbsp()*
20	Coded slice extension *slice_layer_extension_rbsp()*
21	Coded slice extension for depth view components *slice_layer_extension_rbsp()* (specified in Annex I)
22 - 23	Reserved
24 - 31	Unspecified

ii. אם דגל s לא שווה ל-'1' זהו סוף/אמצע של fragment, ובמקרה זה ניקח את ה-payload ונוריד את 2 הבתים הראשונים (למעשה, פשוט נשרשר אותו ל-payload הקודם)

c. כעת, נכתוב את ה-payload לקובץ בינארי, נריץ עליו ffmpeg ונקבל את הווידאו עם ה-flag.

הערה - בסוף הווידאו היה mirrored אז הוספתי שורה בשביל לעשות לו flip horizontal.



קישור לקוד הפתרון

סיכום

זו הייתה הפעם הראשונה שבה ניסיתי לפתור אתגר CTF היות ואין הרבה כאלו בתחום ה-CRF, וזה היה מאוד מהנה. היו נקודות שמאוד אתגרו אותי אבל לבסוף הצלחתי לפתור הכל ולסיים במקום רביעי (למרות שבמקביל אני במשרה מלאה בצבא).

על המחבר

שמי לירן אטליס, בן 27 ומהנדס חשמל בשירות קבע, עוסק בעולם הניווט הלווייני ואלגוריתמים ב-C++. בין כל תחביביי אני מאוד אוהב את תחום ה-CRF ולהתעסק עם מכשירי SDR. מוזמנים להתחבר אלי ב-LinkedIn!