שניר חן | 325107829

סטגנוגרפיה | מבוא לאבטחת המרחב המקוון

רקע: Steganography היא שיטה להסתרת מידע בתוך קובץ, הודעה תמונה או וידאו.

עבודה זו עוסקת בהסתרת טקס בתוך תמונה, ובפענוח של טקסט מוסתר מתוך תמונה.

חלק א

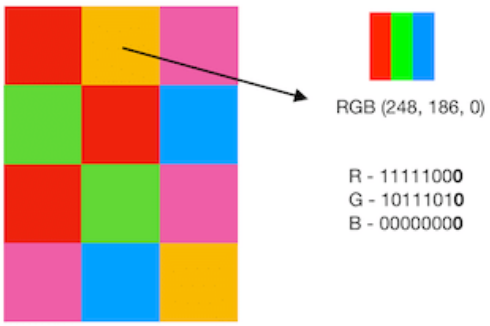
בחלק זה התבקשנו לממש הסתרה של טקסט בתוך תמונה מסוג PNG.

איך הטקסט מוסתר? לפני כן נקפוץ רגע להסבר קצר:

תמונה מורכבת מהמון פיקסלים.

כל פיקסל מורכב משלושה בתים בני 8 ביטים (ערכים 0-255), הנקראים RGB

(Red, Green Blue). כל שלושה בתים שכאלו, מייצגים את העוצמה של כל צבע בפיקסל יחיד. כלומר הפיקסל נצבע בצבע שהוא שילוב של אדום ירוק וכחול בעוצמה שונה.



אז איפה אנחנו מסתירים את הטקסט?

כמו שאמרנו - תמונה מורכבת מפיקסלים, וכל פיקסל מורכב משלושה צבעים.

צבע, מיוצג על ידי בית (Byte) שמורכב מ-8 ביטים (Bits).

ייצוג העוצמה של כל צבע בתוך הבית, הוא כמובן בינארי.

אנחנו מסתירים את הטקסט בתוך הביטים הנמוכים (LSB) שבכל בית.

איך עושים זאת?

לכל אות בשפה האנגלית (וגם לתווים אחרים, כמו רווחים וסימני פיסוק) יש ייצוג ascii שונה. נניח לאות H יש את הייצוג 72, או בבינארי 01001000.

ההסתרה בתמונה נעשית ככה שאנו עוברים על כל הביטים בייצוג הבינארי של ערך הascii של האות , ושמים אותם בביט הנמוך (LSB) של כל בית בתמונה.

לדוגמה, אם ערך ה-ascii של האות H הוא 72, או בבינארי 01001000. אז המעבר יהיה:

0->1->0->0->1->0->0->0

כאשר בכל פעם נשים את הביט שהגענו אליו בתור ה-LSB של הבית הבא שבתמונה.

כך נעשה עד שיסתיימו כל התווים בטקסט שאנו רוצים להסתיר.

המימוש שלי בא לידי ביטוי בפונקציה שתחילה בודקת את תקינות הקלט, ולאחר מכן ממשיכה עם לולאה מקוננת שמחליפה את הביטים המתאימים. הפונקציה נראית כך:

```
def hide(image_path: str, text_to_hide: str) -> str:
    image_as_np_array = utils.png_file_to_rgb_np_array_converter(image_path)
    if image_as_np_array.size < len(text_to_hide) * 8:
        raise ValueError("The given text is too long for the given image.\n"
                         "Try a shorter text, or a bigger image.")
    ascii_values_of_chars = [ord(char) for char in text_to_hide]
    bits_to_hide = ''.join([f'{val:0{NUM_OF_BITS_IN_ASCII_SYMBOL}b}' for val in ascii_values_of_chars])
    for line_num, line in enumerate(image_as_np_array):
        for column_num, column in enumerate(line):
            # looping over all RGB values in image
            rgb_value = image_as_np_array[line_num][column_num]
            for color_index, color_value in enumerate(rgb_value):
                if bits_to_hide == '':
                    new_name = new_path_name(image_path)
                    utils.np_array_to_png_file_converter(np.array(image_as_np_array), new_name)
                    return new_name
                bit_to_hide = bits_to_hide[0]
                bits_to_hide = bits_to_hide[1:]
                image_as_np_array[line_num][column_num][color_index] = utils.set_bit(
                    image_as_np_array[line_num][column_num][color_index], int(bit_to_hide), 0)
    raise Exception("Unexpected error occurred.")
```

ולבסוף, לאחר שסיימנו להסתיר את כל הביטים, והתמונה כעת עם הטקסט מוסתר בתוכה, היא נשמרת בנתיב זהה, ועם אותו שם בתוספת hidden_ ועם סיומת PNG:

```python
def new_path_name(image_path: str) -> str:
    # Given image has png extension
    if image_path[-4:] == '.png':
        return rf"{image_path[0:-4]}_hidden.png"

    # Given image has no png extension
    return rf"{image_path}_hidden.png"
```

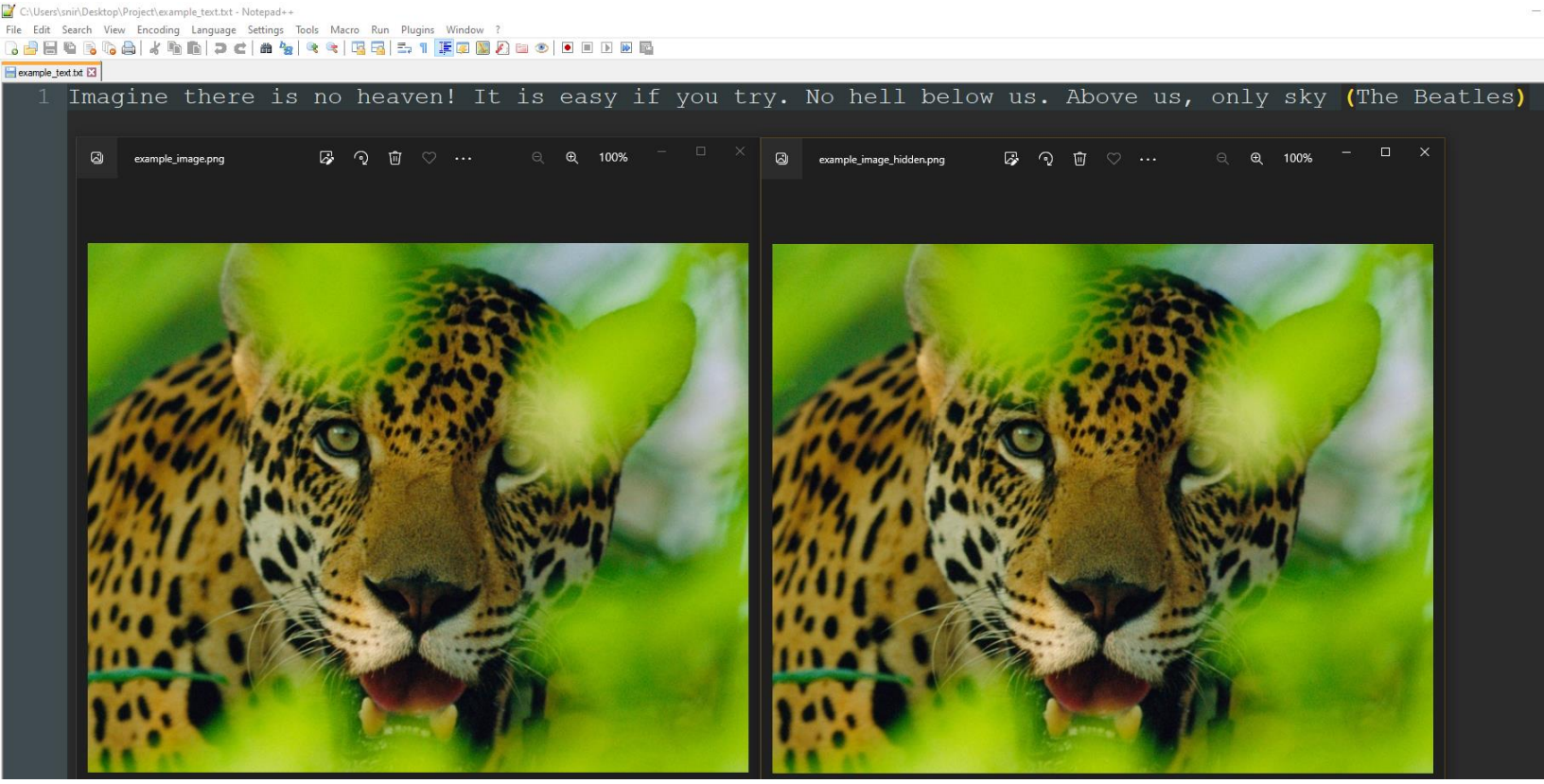
בשביל להריץ את התוכנה נצטרך לספק לה פרמטרים לקובץ תמונה וקובץ טקסט:

```
C:\Users\snir\Desktop\Project>py steg_hide.py --help
usage: steg_hide.py [-h] --image IMAGE --text TEXT

optional arguments:
  -h, --help     show this help message and exit
  --image IMAGE  Path of a PNG image.
  --text TEXT    Path of a TXT file that contains the text you want to hide.
```

ניתן לדוגמה את example_image.png ואת example_text.txt ונריץ את התוכנה:

```
C:\Users\snir\Desktop\Project>py steg_hide.py --image example_image.png --text example_text.txt

The hiding process started. The time is: 2023-01-17 21:34:11.061665

The hiding process finished. The time is: 2023-01-17 21:34:11.289915

The text was hidden successfully in the image!
It was saved in example_image_hidden.png
```

הריצה הצליחה. להלן שתי התמונות, וקובץ הטקסט:



משמאל התמונה הרגילה, ומימין התמונה עם הטקסט המוסתר –

"Imagine there is no heaven! It is easy if you try. No hell below us. Above us, only sky (The Beatles)"

השינוי אינו נראה לעין. נראה שהוא באמת התבצע בחלק ב כשנפענח את התמונה שמימין.

חלק ב

בחלק זה התבקשנו לממש פענוח של טקסט מתוך תמונה עם טקסט חבוי – רק שהפעם יש קאטצ':

הטקסט לאו דווקא חבוי בביטים הכי פחות משמעותיים שבכל בית, אלא יכול להיות חבוי בכל אחד משלושת הביטים הכי פחות משמעותיים (מעתה ואילך אתייחס אליהם בתור ערוצי LSB_0 LSB_1 LSB_2) – בצורה כזו שכל מילה במשפט יכולה להיות ב"ערוץ LSB" אחר. עם זאת, כל מילה מוסתרת מתחילתה ועד סופה באותו ערוץ LSB.

(אך מילים שונות במשפט יכולות להיות באחד משלושת ערוצי ה-LSB השונים)

ואם זה לא מספיק מסובך, אז לאו דווקא מתחילים להסתיר את המשפט בתחילת התמונה, אלא יכולים להתחיל את ההסתרה מכל בית בתמונה.

אז איך מימשתי את זה?

מיד אסביר, אך לפני כן, נראה שהמימוש אכן עובד, תוך פענוח של הטקסט שהסתרנו קודם לכן, בחלק א:

```
C:\Users\snir\Desktop\Project>py steg_decode.py --help
usage: steg_decode.py [-h] --image IMAGE

optional arguments:
  -h, --help     show this help message and exit
  --image IMAGE  Path of a PNG image with hidden text.
```

נספק את הנתיב המתאים ונריץ את התוכנה המפענחת:

```
Select C:\Windows\System32\cmd.exe

C:\Users\snir\Desktop\Project>py steg_decode.py --image example_image_hidden.png

The decoding process started. The time is: 2023-01-17 22:17:32.156947

The text was found successfully! The time is 2023-01-17 22:17:45.204095

The text is: Imagine there is no heaven! It is easy if you try. No hell below us. Above us, only sky (The Beatles)
And it was saved in 325107829.txt
```

ואכן קיבלנו את הטקסט המתאים!

עכשיו כשהשתכנענו שההסתרה והפענוח עובדים, נעבור לחלק הכיף – המימוש.

המימוש היה די ארוך, אעבור כאן על הקטעים המרכזיים והחשובים.

הכל מתחיל בפונקציה הראשית decode:

```
def decode(image_as_np_array: np.ndarray) -> str:
    max_words = 0
    best_guess = ''
    for i in range(8):
        symbols_of_channels = get_hidden_ascii_symbols_of_channels_from_image_as_np_array(image_as_np_array, i)
        guess = guess_hidden_text(symbols_of_channels)
        if len(guess.split(SPACE)) > max_words:
            max_words = len(guess.split(SPACE))
            best_guess = guess
    return best_guess
```

הפונקציה הזו מייצרת בכל פעם רשימה של סימבולים (תווי ascii) משלושת ערוצי ה-LSB.

לאחר מכן, היא שולחת את הסימבולים משלושת הערוצים שמצאנו לפונקציה guess_hidden_text והיא בתורה מספקת את הניחוש הטוב ביותר שלה לטקסט חבוי.

בסופו של דבר, המפשט הארוך ביותר שנמצא, הוא זה שנחזיר בתור הניחוש הטוב ביותר שלנו, ואותו נדפיס ונשמור לקובץ.

יש כאן לולאה של 8 איטרציות כדי לבדוק את כל המיקומים מהם התחילו להצפין את המשפט. למעשה בכל פעם אנו מספקים לפונקציה get_hidden_ascii_symbols_of_channels_from_image_as_np_array

אופסט מסוים, בין 0 ל-7 וממנו אנו מתחילים לבצע את תהליך חילוץ הסימבולים:

```
def get_hidden_ascii_symbols_of_channels_from_image_as_np_array(image_as_np_array: np.ndarray, start_index) -> [[int]]:
    image_as_np_array = image_as_np_array.flatten()
    image_as_np_array = image_as_np_array[start_index:]

    # To make the array divisible by NUM_OF_BITS_IN_ASCII_SYMBOL
    num_of_colors_to_ignore_at_the_end = image_as_np_array.size % config.NUM_OF_BITS_IN_ASCII_SYMBOL
    if num_of_colors_to_ignore_at_the_end:
        image_as_np_array = image_as_np_array[:-num_of_colors_to_ignore_at_the_end]

    hidden_ascii_symbols_of_channels = [copy.deepcopy(image_as_np_array) for _ in range(MAX_HIDE_CHANNELS)]

    for i in range(MAX_HIDE_CHANNELS):
        # Get the LSB-i of each color (byte) in the image
        mask = utils.set_bit_1(0, i)
        hidden_ascii_symbols_of_channels[i] = (hidden_ascii_symbols_of_channels[i] & mask)
        # Combine each 8 bits to a uint-8
        hidden_ascii_symbols_of_channels[i] = np.packbits(hidden_ascii_symbols_of_channels[i])
        # Convert each uint-8 to its ascii representation
        hidden_ascii_symbols_of_channels[i] = list(map(chr, hidden_ascii_symbols_of_channels[i]))

    return hidden_ascii_symbols_of_channels
```

תהליך חילוץ הסימבולים משתמש בספרייה בשם numpy שעוזרת לנו בפעולות שמצריכות מטריצות, כמו כאן.

תכלס מה שקורה כאן זה שאני מייצר mask-ים שונים לכל ערוץ LSB, ומחלץ בעזרת פעולות bitwise את הביטים המתאימים. לאחר מכן כל 8 ביטים אני הופך לבית בייצוג מספרי, ולבסוף כל ייצוג מספרי אני הופך לתו ה-ascii המתאים.

הידד!

עכשיו כשבידינו כל הסימבולים שבפוטנציאל חלקם הם אלו שהוסתרו, אנו ממשיכים לשלב הבא – הלוא הוא ניחוש הטקסט החבוי:

```python
def guess_hidden_text(symbols_of_channels: [[str]]) -> str:
    valid_text_channels = [[]] * len(symbols_of_channels)

    for i in range(len(valid_text_channels)):
        valid_text_channels[i] = get_valid_symbol_combinations_of_channel(symbols_of_channels[i])
        valid_text_channels[i] = remove_non_valid_one_letter_words(valid_text_channels[i])
        valid_text_channels[i] = remove_non_english_words(valid_text_channels[i])
        valid_text_channels[i] = add_punctuations_if_exists(valid_text_channels[i], symbols_of_channels[i])

    all_texts = [text for valid_text_channel in valid_text_channels for text in valid_text_channel]
    all_texts.sort(key=lambda t: t.start_index)

    hidden_text = longest_words_strike(all_texts)
    return hidden_text
```

אנחנו מתחילים עם שלושה ערוצי LSB שמלאים בסימבולים.

עובדים בשיטת המסננת – בעבור כל ערוץ, אנחנו מסננים את כל מה שלא קשור, עד כמה שאפשר. השלבים הם:

1. סינון התווים שהם לא ascii – בסיום שלב זה אנחנו נשארים עם מערך של רצפי תווים קריאים (אותיות abc, רווחים וסימני פיסוק). בעבור כל רצף כזה אני שומר גם start_index המסמל את האינדקס של התו בו התחיל הרצף מבין התווים בערוץ ה-LSB המתאים.
2. סינון המילים שהן רק אות אחת, והן לא רווח, i או a (אלו הם התווים היחידים שעומדים בפני עצמם)
3. סינון מילים שהן לא מילים באנגלית – ארחיב בהמשך.
4. הוספת סימני פיסוק למילים, אם היו סימני פיסוק מוסתרים לצד המילה.

לבסוף, כשיש רצפי מילים הגיוניות עם פיסוק ורווחים בשלושה ערוצים שונים, אנו מאחדים את כל הערוצים לרשימה אחת, וממיינים אותה לפי ה-start_index של כל מילה.

כשיש לנו את הרשימה המאוחדת הזו, אנו רצים עליה ומחפשים את הרצף הכי ארוך (משפט). התהליך הזה נעשה בפונקציה longest_words_strike עליה ארחיב בהמשך.

אבל קודם, אסביר איך סיננתי את המילים שלא נמצאות בשפה האנגלית:

```python
def remove_non_english_words(words: [Text]) -> [Text]:
    filtered_words = []

    for word in words:
        if word.text == SPACE:
            filtered_words.append(word)

        english_word = probably_english_word(word)

        if english_word:
            # Capital letters can be only at the start of a word
            word_is_valid = True
            for letter in english_word.text[1:]:
                if letter in VALID_UPPERCASE_LETTERS:
                    word_is_valid = False
                    break
            if word_is_valid:
                filtered_words.append(english_word)

    return filtered_words
```

```python
def probably_english_word(word: Text) -> Text or None:
    """
    Notice - our MOST_COMMON_ENGLISH_WORDS list is lowercase.
    :param word: Combination of readable symbols with text which is not '' and not  ' '.
    :return: What is most likely to be the word in english of the given text, or None if nothing was found.
    """
    text = word.text
    text_lower = text.lower()
    start_index = word.start_index

    if text_lower in config.MOST_COMMON_ENGLISH_WORDS_LOWERCASE:
        return word

    for word in config.MOST_COMMON_ENGLISH_WORDS_LOWERCASE:
        if word == text_lower:
            return word
        if word in text_lower:
            if len(text_lower) - len(word) <= 1:
                if text_lower[1:] == word:
                    return Text(start_index + 1, text[1:])
                return Text(start_index, text[:-1])
    return None
```

לפונקציות האלו אנו מגיעים מצוידים מראש, עם רשימה ב-lowercase של המילים הנפוצות ביותר באנגלית. (מעתה ואילך אתייחס לרשימה הזו כאל המילון שלנו)

תחילה אנחנו בודקים אם יש התאמה מושלמת של המילה שלנו למילה במילון. אם כן – מצוין, אנחנו נשמור את המילה.

אם אין התאמה, יכול להיות שעדיין מדובר במילה שנרצה להשאיר. לדוגמה hellox או xhello זו לא מילה באנגלית, אך יכול להיות שבמקרה הופיעה האות x או כל אות אחרת לפני או אחרי מילה אמיתית באנגלית.

אני בודק זאת, ואם אכן מדובר במילה כזו – מעולה, אני משאיר אותה, ומעדכן את הטקסט שלה ואת ה-start_index במידת הצורך.

בדיקה אחרונה לפני שאני מחזיר את רשימת המילים באנגלית שמצאתי, היא בדיקה של אותיות קטנות/גדולות. מילים באנגלית יכולות להתחיל באות גדולה, אך לא יכול להיות שבאמצע מילה תהיה אות גדולה – ולכן אני מסיר את האופציות האלו.

כל המילים באנגלית בידינו!

עשינו זאת בעבור כל אחד משלושת ערוצי ה-LSB, ועכשיו אחרי שאיחדנו את שלושת הערוצים לרשימה אחת ארוכה, הגיע הזמן לבדוק מה המשפט הארוך ביותר, ולהחזיר אותו.

```
def longest_sentence(all_words: [Text]) -> str:
    """...."""
    max_strike_len = 0
    longest_strike = []

    for i in range(len(all_words)):
        current_longest_strike = longest_sentence_from_word(i, all_words)
        current_strike_len = len(current_longest_strike)
        if current_strike_len > max_strike_len:
            max_strike_len = current_strike_len
            longest_strike = copy.deepcopy(current_longest_strike)

    return ''.join([word.text for word in longest_strike])
```

אנו עוברים על כל המילים.

כל מילה אנחנו שולחים לפונקציה שמתחילה לבדוק רצף הגיוני ארוך ביותר החל מהמילה הנוכחית.

לבסוף, אנחנו מכריזים על המשפט הארוך ביותר כעל הרצף הארוך ביותר שהגענו אליו מכל בדיקות הרצפים האפשריים, ומחזירים אותו.

כדי לבדוק מהו הרצף ההגיוני הארוך ביותר החל ממילה מסוימת, אנו משתמשים בפונקציה רקורסיבית לא מאוד מסובכת:

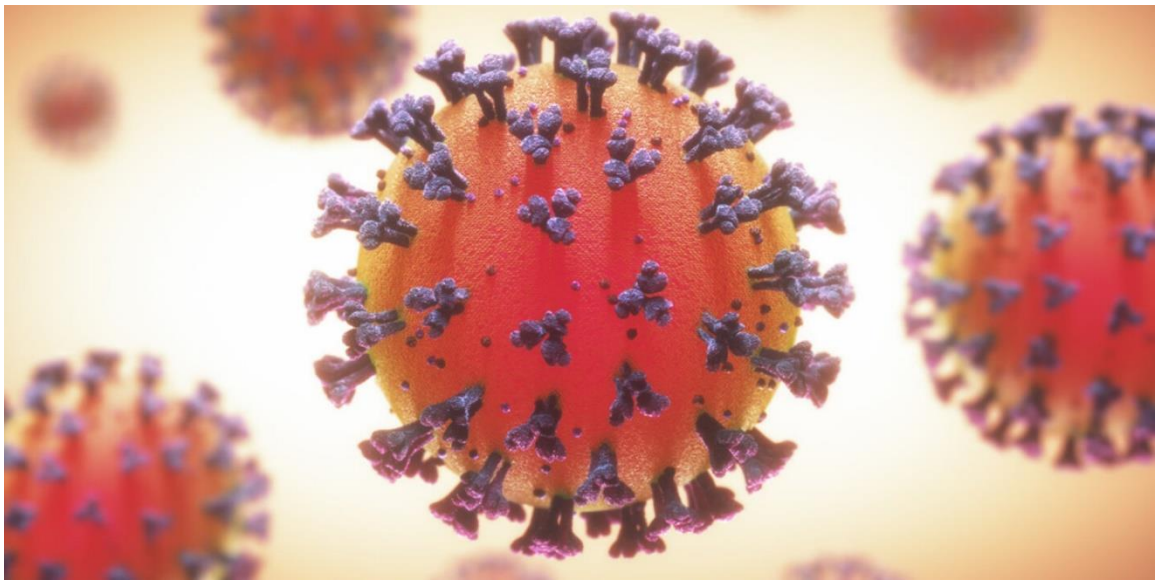
```python
def longest_sentence_from_word(i: int, all_words: [Text], should_be_space=False) -> [Text]:
    """This recursive function, returns the longest strike of words and spaces starting from a specific word...."""
    current_word = all_words[i]

    if all_words[i].text == SPACE and not should_be_space:
        return []
    if all_words[i].text != SPACE and should_be_space:
        return []

    # Because we combined 3 hidden channels (of 3 LSBs) we may have up to 3 options to continue the strike
    possible_options_to_continue = []
    next_start_index_should_be = all_words[i].start_index + len(all_words[i].text)

    for j in range(i + 1, len(all_words)):
        if all_words[j].start_index < next_start_index_should_be:
            continue
        if all_words[j].start_index > next_start_index_should_be:
            break
        possible_options_to_continue.append(j)

    possible_strikes = [longest_sentence_from_word(j, all_words, not should_be_space) for j in
                        possible_options_to_continue]

    return [current_word] if possible_strikes == [] else [current_word] + max(possible_strikes, key=len)
```

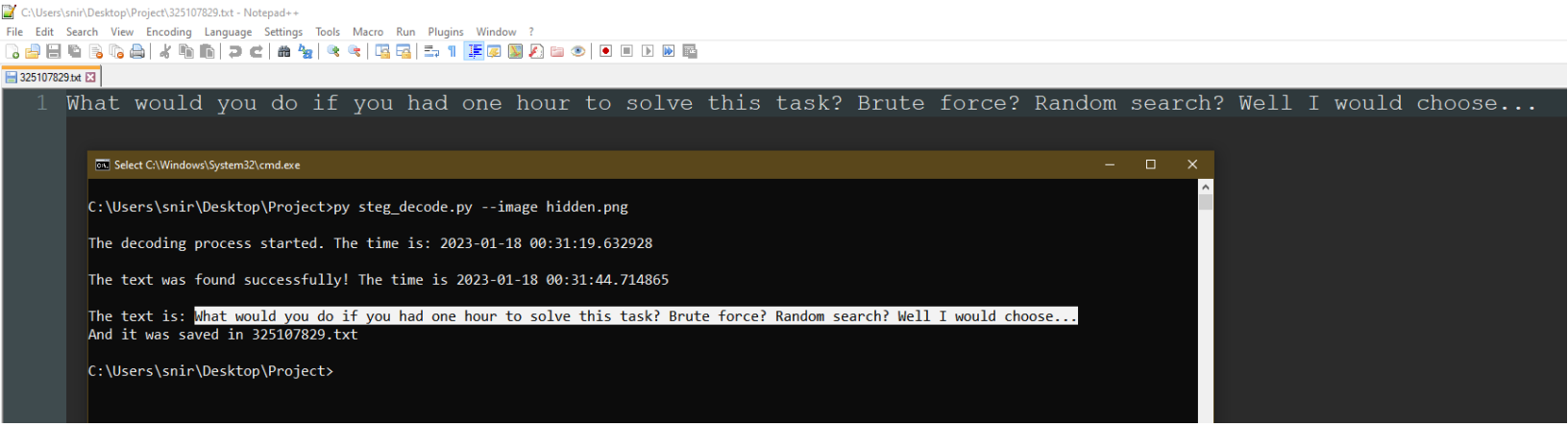
הפונקציה הזו מסתכלת על ה-start_index של המילים, ולפי גודל המילה, מחשבת מה האינדקס בו אמורה להיות המילה הבאה בתור.

כמו כן, מתבצעת בדיקה שלאחר כל מילה באנגלית יש רווח, ולאחר כל רווח יש מילה באנגלית (כדי שנקבל משפט תקין)

סיימנו לעבור על כל הלוגיקה של התוכנה!

הגענו לרגע המיוחל - להריץ את תוכנת הפענוח שלנו על התמונה הנתונה hidden.png:





אז לסיכום – מה הייתי עושה אילו הייתה לי רק שעה לפתור את המשימה הזו?

ככל הנראה הייתי כותב קוד שמפענח את כל ערוצי ה-LSB האפשריים, מדביק את הכל ב++notepad ומחפש עם Ctrl+F מילים רנדומליות עד שהיה נגמר לי הזמן 😊