**פרויקט סטנוגרפיה – מבוא לאבטחת המרחב המקוון**

הערה על המימוש הכללי:

בעת המימוש, הקפדתי להשתמש כמה שיותר בפונקציות של הספרייה NumPy. הסיבה לכך, היא שהספרייה היא למעשה עטיפה של קוד C – זה כמובן רץ יותר מהר מקוד Python (הרי שפת C היא שפה Low-Level שמתקפלת לקוד מכונה, לעומת Python, שנחשבת לאחת השפות הכי High Level שמונעות בעזרת Interpreter).

ספריית Numpy מספקת פעולות מועילות מאוד על מערכים, ובפרט, בשני חלקי המטלה השתמשתי בפונקציות flatten() ו-reshape() – הן מאפשרות לשנות את האופן שבהם מסודרים האיברים – כלומר את מימדי המערך, תוך שמירה על הסדר של האיברים מהמימד הראשון לאחרון.

מעבר לכך, עוד זוג פונקציות שנעשה בהן שימוש הוא numpy.packbits, numpy.unpackbits. הפונקציות הללו מאפשרות, בהתאמה, לקחת ערך מספרי, ולפרק אותו לסיביות שלו, או להפך, לקחת מערך סיביות, ולאסוף אותן לכדי ערכים.

גם אופרטורים מתמטיים בסיסים ב-NumPy (כמו +,-,\*,/,&,|) פועלים איבר-איבר בצורה מהירה מאוד – וסייעו בעיבוד של התמונה כמערך.

חלק א׳ – הצפנה

אעיר כי חלק א׳ נעשה במודעות למהות חלק ב׳ – ולמעשה חלק ב׳ יכול לשמש כמפענח של חלק א׳. לפיכך, שמרתי את הסיביות לתוך ערכי הפיקסלים בצורה רציפה, דהיינו, תו ראשון בפיקסלים 0-7, תו שני בפיקסלים 8-15, וכו׳.

להרצת התוכנית:

python steg\_hide.py <PATH\_TO\_IMAGE> “<MESSAGE>”

התוכנית שומרת את התוצאה לאותה תיקייה, עם אותו שם קובץ, בהחלפת הסיומת ל-\_hidden.png.

דוגמה לתמונה שבה מוסתר המסר “Hello, World! My Name is aviv and I am going to do something very simple today. I am going to write a long message and I hope my code will be able to find it quickly and safely.”, לפני ואחרי ההסתרה, משמאל לימין בהתאמה:



קשה עד בלתי אפשרי להבחין בהבדל כלשהו בין התמונות – על האחת, חלק ב׳ מוציא שגיאה, ובאחרת – הוא מצליח לאתר את המסר ולהדפיסו באופן מדויק.

המימוש נעזר בפונקציות של NumPy בצורה עמוקה כדי להאיץ כמה שיותר את ההסתרה:

* לוקחים את מחרוזת הקלט ובונים ממנה NumPy Array של הבתים שמרכיבים אותה.
* מבצעים Unpack Bits כדי לקבל את ערכי הסיביות של המחרוזת באופן רציף.
* בעזרת Bitwise operators מאפסים את הסיביות התחתונות היכן שרלוונטי – והופכים אותן ל-1 במקומות הנדרשים.

חלק ב׳ – פענוח

יש לשים לב כי חלק ב׳ נעזר בקובץ dictionary.txt והוא נדרש להימצא באותה תיקייה שבה הסקריפט מורץ – זה המילון שבו הוא נעזר לשם מטריקת המרחק – שמסייעת לוודא שההודעה שנמצאה תקינה.

כדי להריץ את התוכנית:

python steg\_decode.py <PATH\_TO\_IMAGE>

הרצת הפענוח על הדוגמה הניב את הפלט:

“What would you do if you had one hour to solve this task? Brute force? Random search? Well I would choose...”

הרצה של תוכנית ההצפנה עם מחרוזת ארוכה הניבה את תמונת המנעול המצורפת, והפעלה של תוכנית הפענוח על תמונת הפלט – החזירה בדיוק את ההודעה המוצפנת:

“Hello, World! My Name is aviv and I am going to do something very simple today. I am going to write a long message and I hope my code will be able to find it quickly and safely.”

כנדרש.

הרצה על מספר תמונות המצורפות כאן (מלבד המנעול , שמוזכר לעיל כמובן – שעבר הצפנה) בנוסף הניב הדפסה של שגיאה – כולן תמונות אקראיות מהאינטרנט, שלא היה צריך להימצא בהם מסר כלל – ואכן – התוכנה לא מצאה כזה עם הפרמטרים המוגדרים לה.

A picture containing person

Description automatically generatedA picture containing bubble

Description automatically generated

הקוד מתועד היטב, ולמרות זאת, מספר הערות לגבי המימוש, ופירוט אודות אופן חיפוש המחרוזת:

ברור שצריך להתחיל את חילוץ התווים מתוך התמונה בכל אחד מ-8 הפיקסלים הראשונים – שהרי כל תו הינו בית, שמכיל 8 סיביות, ואנו פורשים סיבית על כל פיקסל – מהפיקסל ה-9 והלאה נחזור על עצמנו כמובן. אופן הפעולה עבור כל פיקסל התחלה הוא כך:

1. הפונקציה decode\_string מחלצת ומחזירה מחרוזות רציפות של תווי ASCII מתוך התמונה, בכל אחד מ-3 הסיביות הנמוכות – כפי שהוגדר במטלה. היא מתחילה ב-offset שצוין למעלה, ובמקרה הנוכחי – תמיד תחזיר 3 מחרוזות. היא עושה את הפעולות ההפוכות, במידה מסוימת, לחלק א׳ – משתמשת בפעולות Bitwise כדי לחלץ את הסיביות במיקום הרלוונטי, אוספת קבוצות של 8 סיביות לכדי ערכי הבתים, מתרגמת חזרה לתווים – ומתקבלת מחרוזת. ביצוע הפעולה מראש ובעזרת NumPy ולא Online תוך חיפוש המחרוזת – קריטי לביצועים – ומפשט את התוכנית בצורה משמעותית.
2. הפונקציה find\_message פועלת בצורה פשוטה – היא מחפשת הודעה עבור המחרוזות שהופקו ע״י decode\_string, בכל אחד מהאינדקסים האפשריים. היא נעזרת בפונקציה רקורסיבית פשוטה, decode\_string\_recursive, ששואפת למצוא את ההודעה התקינה הארוכה ביותר, שעונה לכל התנאים בשאלה. הפונקציה הרקורסיבית מסננת את הקלטים כדי לפעול מהר יותר:
3. בודקת שלא יצאנו מתחומי המחרוזת (תנאי עצירה).
4. בודקת שהתו הנוכחי תקין.
5. בודקת שבין מילים שונות, תמיד מפרידים רווחים (כאשר עוברים לקרוא את ההמשך במחרוזת שונה – בהכרח קוראים מילה אחרת, וזה התנאי הנבדק).

לפני שהיא מחזירה תשובה סופית (תנאי עצירה נוסף) – היא בודקת בעזרת הפונקציה is\_message\_valid שההודעה מכילה מספיק מילים בכלל ומספיק מילים שנמצאות במילון בפרט (50% ומעלה). שאר התנאים, כאמור, נבדקים בתוך הפונקציה הרקורסיבית עצמה.

הארגומנטים לפונקציה הרקורסיבית מפורטים בתוך הקוד בתיעוד בצורה מפורטת. באופן כללי, הפונקציה בונה את ההודעה תו-תו במורד עץ הרקורסיה בעזרת כל אחת מהמחרוזות שניתנות כקלט משלב 1, ואם נכשלת בהוספת תו – בודקת את תקינות ההודעה הנוכחית, ובכך שואפת למצוא את ההודעה התקינה הארוכה ביותר.

עוד אעיר, כי פנייה למחרוזת באינדקס שמועבר בקריאה הרקורסיבית מהיר בהרבה משינוי המחרוזות – בגלל העתקות שכמובן מתבצעות בכל שינוי מחרוזת. שינוי המחרוזת בכל קריאה מאט את התוכנית עד פי 30 – ולכן העברתי ברקורסיה אינדקס נוכחי.

רוב הפרמטרים ניתנים לשינוי בקבועים בראש התוכנית – החל התווים שיכולים להכיל מילה מתוך המילון, תו מפריד בין המילים (רווח), וכלה בתווים נוספים שיכולים להופיע בין מילים.