פרויקט בסייבר במקצוע "תכנון ותכנות מערכות" – מערכת זדונית לשליטה מרחוק בין 2 משתמשי קצה

מגיש: הראל שץ 326004041

מנחה: אלון פלומן

בית ספר: תיכון חדרה

תאריך: XX/05/22

הסבר על המערכת

המערכת שבניתי נקראת Malicious VNC Client-Server.

המערכת מאפשרת גישה לממשק משתמש שממנו ניתן להתחיל תהליך שבו שני מחשבים מתחברים, ואז הקליינט יכול לשלוט על מחשב השרת לפי פרוטוקול VNC.

השרת מסטרים לקליינט את מסכו (שיתוף מסך), והקליינט שולח לשרת inputs שהשרת מבצע וכך הקליינט שולח פקודות לשרת ושולט עליו.

בנוסף לכך, ישנם פונקציות זדוניות שהקליינט יכול להריץ על השרת לאחר ההתחברות למשל: להריץ keylogger שישלח ישירות אל הקליינט, לשים את הפעלת השרת ב-registry כך שהשרת יעבוד בלי אותנטיקציה של המשתמש וללא ידיעתו, חסימת הקלט של המשתמש הנשלט (חסימת המקלדת והעכבר), לקיחת סטרים ממצלמת המחשב הנשלט (אם קיימת מצלמה).

מטרות ויעדים

המטרה היא ליצור מערכת שמסוגלת להתמודד עם פעולה אינטנסיבית כמו שיתוף מסך – התוכנה צריכה לעבוד גם על מכשירים חלשים מבחינת חומרה וגם מבחינת קצב האינטרנט.

אני שואף להקטין את ה-input lag ואת ה-delay בשיתוף המסך ובהעברת ה-inputs כמה שאפשר, המערכת צריכה להשתמש באלגוריתמים יעילים ולנצל את החומרה באמצעות שימוש בתהליכונים (threads).

בנוסף צריך לדאוג שהתוכנה לא תתגלה ע"י אנטי-וירוסים שונים בזמן שהקליינט מממש את הפעולות הזדוניות.

מהות המערכת

המערכת יכולה בעיקר לשמש כמערכת שליטה מרחוק רגילה (בלי האמצעים הזדוניים) – ניתן להשתמש בה כדי להתחבר למחשבים מרוחקים ששום אדם אינו יכול להפעיל אותם או לעזור לאדם שלא מבין במחשבים ע"י השתלטות על מחשבו.

אפשר גם להשתמש במערכת בדרך זדונית כמו השתלטות על מחשבים של פושעים וכך להשיג עליהם מידע דרך המחשב שלהם.

התקנה והרצה

כדי להריץ את הפרויקט צריך להוריד אותו תחילה מ-Github:
https://github.com/ArelShatz/Malicious-VNC-Client-Server

לאחר מכן יש להריץ את הקובץ packagesSetup.py שמתקין את כל החבילות הנדרשות.

במידה והקובץ לא רץ או שלא כל החבילות מותקנות נכון, יש להתקין אותן בצורה ידנית באמצעות pip (הקובץ packagesSetup.py מספק מידע על אילו חבילות הותקנו נכון ואילו נכשלו), החבילות שצריך להתקין הן pyscreenshot, mss, opencv-python, colorlog, PyQt5, simplejpeg, zmq, pynput, numpy, tqdm, requests, Pillow.

לאחר שהכל הותקן, יש להריץ את הקובץ main.py זה יפתח את ממשק המשתמש בחלון הראשי.

מהמסך הראשי אפשר לעשות כמה דברים:

אפשר לשנות את ההגדרות מתחת ל-Settings.

אפשר להתחיל Session ע"י לחיצה על Bind מתחת ל-Connection.

אפשר להצטרף ל-Session ע"י לחיצה על Connect מתחת ל-Connection.

אפשר להתנתק מ-Session ע"י לחיצה על Disconnect מתחת ל-Connection (לא יעבוד אם אתם יצרתם את ה-Session).

אפשר לסגור Session ע"י לחיצה על Close מתחת ל-Connection (יעבוד רק אם אתם יצרתם את ה-Session).

פרוטוקול VNC

העברת המידע מתבססת על אלמנטים של הפרוטוקול VNC שהוא פרוטוקול שמאפשר שליטה על מחשבים מרוחקים בצורת peer to peer ושיתוף המסך של המחשב הנשלט.

שיתוף המסך של הפרוטוקול עובד ע"י שליחת הפריימים של מסך המחשב הנשלט ישירות למחשב השולט, בניגוד לפרוטוקולים אחרים כמו RDP שבהם נשלחים הוראות למחשב השולט על איך צריך לבנות מחדש כל פריים שנשלח מהמחשב הנשלט.

אז נשאלת השאלה איך הפרוטוקול הזה יכול לתפקד בצורה חלקה, הרי להעביר פריימים שלמים בצורה חלקה זה דבר שדורש הרבה משאבים.

התשובה היא שהפרוטוקול משתמש בדרכים שונות להעברת הפריימים שיעילות הרבה יותר מהעברת כל הפריימים כמו שהם, כל דרך להעברת פריימים נקראת Encoding (או קידוד).

הקידוד הפשוט ביותר שכל שרת VNC חייב שיהיה לו נקרא Raw Encoding – זוהי דרך להעברת פריימים שבה הפריים הכי עדכני מושווה עם הפריים הקודם (הפריים שמוצג לקליינט) ונמצאים רק האזורים בפריים העדכני שהשתנו מהפריים הקודם וכך ניתן לשלוח רק את האזורים שהשתנו במקום את כל הפריים.

האזורים שהשתנו הם בעצם אזורים מלבניים ובכל פריים פשוט נשלח לקליינט רשימה של מלבנים כך שלכל מלבן מכיל metadata, כלומר המידע על התמונה (הקורדינאטות של המלבן, הגובה שלו, האורך שלו) ואת התמונה שלקוחה מהפריים בקורדינאטות האלו כך שהקליינט ידע איפה באיזה אזור בתמונה צריך להוסיף את החלקים שהשתנו מהפריים הקודם.

כדי למצוא את האזורים שמשתנים מהפריים הקודם נשתמש ב-Open CV כך:

קודם כל נהפוך את הפריים העדכני והפריים הקודם ל-GrayScale (שחור-לבן) ונבצע את הפעולה XOR על שתי התמונות (XOR היא פעולה שמחזירה 0 אם שני המספרים הם אותו דבר ומספר שהוא לא 0 אם הם לא שווים), כך שנקבל תמונה חדשה שבה כל החלקים שלא השתנו הם שחורים (מכיוון ש-0 מתורגם לצבע שחור) וכל החלקים שנשארו דומים הם לא שחורים.

אחר כך נשנה את הצבע של כל האזורים הלא שחורים ללבנים ולבסוף נמצא את הקורדינאטות, האורך והגובה של כל האזורים הלבנים בתמונה ואז ניקח את כל המלבנים בפריים העדכני בעזרת הקורדינאטות שמצאנו.

VidGear

בפרויקט הזה השתמשתי ב-Framework שנקרא VidGear שאחראי על כל מה שקשור בהעברת וידאו (שיתוף מסך, סטרימים, הפעלת מצלמה, כתיבה של סטרים לקובץ ועוד...).

עד עכשיו השתמשתי בשני כלים של VidGear (שנקראים Gears): ScreenGear ו-NetGear.

VidGear נמצא בקבצי הפרוייקט מכיוון שהתאמתי אותו אישית לפי הצרכים שלי ולכן אי אפשר להוריד את החבילה מהאינטרנט.

ScreenGear הוא כלי מעטפת ל-pyscreenshot רק שהוא מוסיף multithreading.

השתמשתי בגרסה מותאמת אישית של ScreenGear שמוסיפה מחסום.

pyscreenshot הוא בעצמו מעטפת להרבה פתרונות לצילום מסך (PyQt5, mss, PIL...), אני אישית בחרתי להשתמש ב-mss בגלל שהוא נתמך בכל פלטפורמה והוא מהיר מאוד.

NetGear הוא כלי שנועד לשליחה של וידאו ברשת שמשתמש בחבילה pyzmq (חבילה להעברת מידע ברשת) אבל הוא מוסיף גם multithreading.

באמצעות NetGear אפשר להתחבר עם מחשב אחר ולהתחיל להסטרים אליו מידע.

ב-NetGear יש גם אפשרות ל-lossy compression, כלומר דחיסה שבה נאבד מידע מהתמונה, כלומר הורדת האיכות של התמונה, באמצעות הספרייה simplejpeg כך שגודל התמונה יורד וכך העברת התמונה תהיה מהירה יותר על חשבון הזמן שלוקח לכווץ את התמונה.

הגרסה של NetGear שבה השתמשתי היא גרסה מותאמת אישית: הוספתי גם את הקטנת ממדיי התמונה לפני שליחתה כך שתהיה קטנה יותר בזיכרון ויהיה קל יותר להעביר אותה ברשת ובנוסף השתמשתי ב-loseless compression, כלומר דחיסה שבה לא נאבד מידע מהתמונה אבל היא תהיה אפילו קטנה יותר בזיכרון.

כל אלו תורמים לחיבור להיות חלק מאוד וכמעט בלי delay.

בהמשך גם אשלב את ה-Raw Encoding ב-NetGear כדי שהעברת הפריימים תהיה אפילו מהירה יותר.

loseless/lossy compression

שתי שיטות הכיווץ האלו שונות לגמרי אבל אפשר לשלב את שתיהם כדי לכווץ תמונה.

lossy compression הוא כיווץ שבו מאבדים מידע מהתמונה, על ידי שימוש באלגוריתם שמוריד את איכות התמונה בכך שהוא מחבר כמה פיקסלים עם צבעים שונים לצבע אחד שהוא הממוצע של הצבעים של כל הפיקסלים שהתחברו לדוגמא:

אם האלגוריתם החליט לחבר את שני הפיקסלים האלה הוא יחליף אותם בצבע שדומה מספיק לשניהם.

loseless compression הוא כיווץ שבו לא מאבדים שום מידע מהתמונה, על ידי שימוש האלגוריתם שבעצם משמש לכיווץ כל מידע ולא רק תמונות.

האלגוריתם עובר על הבתים שמרכיבים את התמונה ואם הוא מוצא רצף שחוזר על עצמו הוא מכווץ אותם כך: אם האלגוריתם מצא רצף של בתים 0xFF 0xFF 0xFF 0xFF 0xFF הוא מכווץ את הרצף לבתים 0x05 0xFF.

בעצם האלגוריתם החליף את הרצף של חמש הבתים הדומים בבית אחד שאומר כמה בתים חוזרים על עצמם יש ברצף (0x05) ועוד בית שאומר מה חוזר על עצמו (0xFF) וכך לא איבדנו שום מידע.

אפשר להשתמש ב-lossy compression קודם ואז ב-loseless compression כדי לכווץ את התמונה מאוד.

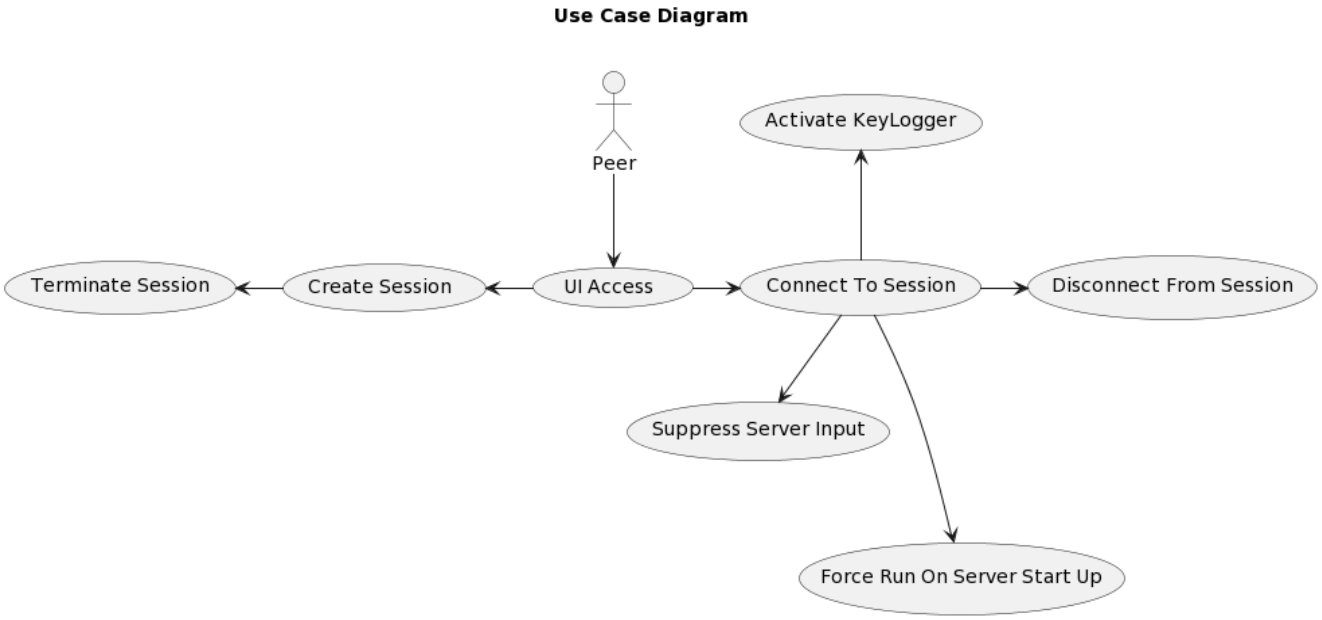
Framerate

הגבלת ה-fps חשובה מכיוון שגורם להעברת הפריימים להיות בקצב קבוע, בנוסף ניתן להשתמש בשיטה זו כדי להוריד את השימוש במעבד ואת השימוש ב-network בשביל מחשבים שלא יכולים לעמוד בשיתוף המסך הכבד.

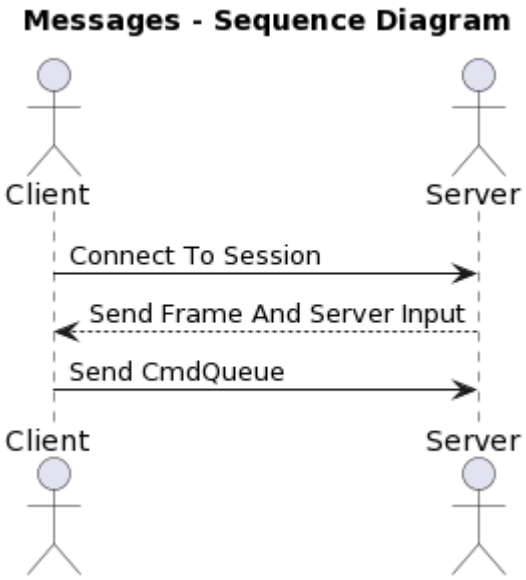
על מימוש מחסום ה-fps נדבר אחר כך.

דיאגרמות UML

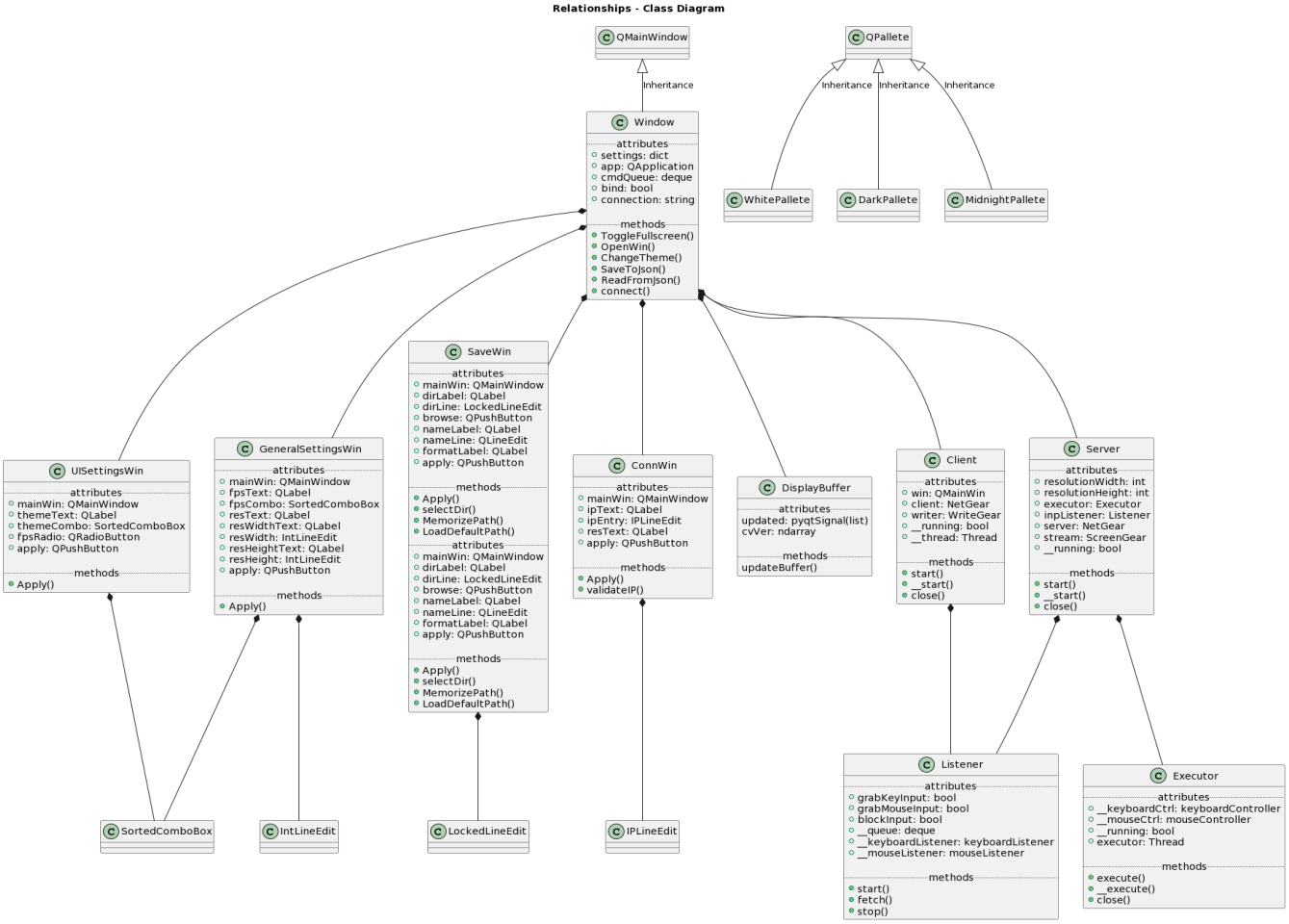
Use case Diagram



Sequence Diagram



Class Diagram



packagesSetup.py

הקובץ הזה לא קשור להרצת הפרויקט עצמו, אלא להתקנת כל החבילות החיצוניות שנדרשות להפעלת התוכנה.

utils.py

קובץ זה מכיל פונקציות עזר שבהם השתמשתי, משתנים וקבועים.

קובץ זה כולל פעולות עזר למימוש הגבלת הפריימים לשנייה:

כדי להגביל את כמות הפריימים לשנייה צריך איכשהו להבין כמה זמן עובר בין כל פעם ש-frame מתחיל עד לסופו.

כדי להשיג את הזמן המדויק בכל רגע, נשתמש בפקודה perf_counter() מהספרייה המובנת time.

הסיבה שלא נשתמש הפקודה time.time היא שהפקודה הזאת לא מדוייקת,
perf_counter() היא הדרך המדויקת ביותר לקבל את הזמן הנוכחי.

נשתמש בפקודה פעמיים, פעם אחת בתחילת הפריים ופעם אחת בסופו (לאחר הפעולה הנדרשת), נחסר את הסוף וההתחלה ונקבל את הזמן שלקח לבצע את הפרים הנוכחי.

השלב הבא יהיה לראות כמה זמן עבר (הפרש הזמנים) ולחכות את הזמן שנשאר לדוגמה:
אם נרצה שה-fps יהיה 24 נצטרך שכל פריים ימשך $\frac{1}{24} = 0.041666$ שניות ולכן נחסיר את הפרש הזמנים מהזמן שהפריים אמור להימשך כך: $0.041666 - timeDiff$.

מה שקיבלנו הוא הזמן שצריך לחכות עד לפריים הבא ולכן נכניס את הערך הזה לפונקציית העזר halt, העוצרת את התוכנית למספר מסוים של שניות.

הסיבה שלא השתמשתי בפקודה time.sleep היא שהפקודה הזאת לא מדויקת לרמה של מילישניות וגם לא עקבית – פעם אחת הפקודה יכולה לחכות קצת פחות המזמן המבוקש ופעם קצת יותר ולכן בניתי פעולה משלי שתהיה מדויקת ועקבית.

הפעולה SpinLock היא גם פעולה שמחכה מספר מסוים של שניות, היא הפעולה הכי מדויקת.

הפעולה בודקת את הזמן הנוכחי ללא הפסקה בלולאת while, הזמן הנוכחי מוחסר מהזמן שבו הפונקציה החלה לפעול ונעשית בדיקה האם הזמן שעבר מאז תחילת הפונקציה קטן המזמן שהמשתמש ביקש לחכות, אם כן הפעולה מסתיימת.

הבעיה בשיטה הזו היא שלולאת ה-while מבזבזת הרבה cpu ולכן יש את הפונקציה halt.

הפונקציה halt היא קצת פחות מדויקת מ-SpinLock אבל היא מדויקת מספיק ולא משתמשת בהרבה משאבים בעזרת שימוש משולב בין הפונקציה SpinLoop לפונקציה time.sleep.

ScreenGear.py

זהו קובץ ששייך לספריה VidGear אבל הוספתי בו הגבלת פריימים בעזרת השיטה שהסברתי בקובץ utils.py ובעזרת שימוש בפונקציית העזר halt.

NetGear.py

גם הקובץ הזה שייך לספריה VidGear אבל הוספתי בו את הכיווץ ה-loseless בעזרת שתי משתנים נוספים שניתן להכניס: rle_compression – בוליאני, קובע האם להשתמש בדחיסה.

rle_compression_strength – מספר מ- 1 עד 9, קובע באיזה חוזק צריכה להיות הדחיסה (1 – הכי חלש, 9 – הכי חזק).

בשביל הדחיסה עצמה השתמשתי בספריה zlib.

בנוסף הוספתי גם את כיווץ התמונה לפני שליחתה בהתאם לדרישת ה-client בעזרת הפקודה cv2.resize.

בנוסף בקובץ זה מימשתי את ה-raw encoding כחלק מפרוטוקול VNC שעליו הסברתי בפרק "פרוטוקול VNC".

האלגוריתם עצמו בנוי בעזרת cv2 וממומש באופן שהוסבר בפרק "פרוטוקול VNC".

InputListener.py

הקובץ הזה מכיל class שנקרא Listener והוא אחראי על כל מה שקשור בלקיחת ה-input של המשתמש וארגונו בעזרת החבילה pynput.

Listener מגדיר את ה-keyboardListener שאוסף את כל מה שקשור לקלט של המקלדת והוא מגדיר גם את mouseListener שאוסף את כל המידע שקשור לקלט מהעכבר.

גם ה-keyboardListener וגם ה-mouseListener הם לפי יורשים מ-threading.Thread בסופו של דבר ולכן הם לא חוסמים, Listener הוא בעצם איגוד של שני ה-threads האלה.

בשני ה-threads ניתן להגדיר callbacks כך שהקוד יקפוץ לפונקציה מסוימת שהגדרנו במקרה של event מסויים.

ה-callbacks הקיימים ב-keyboardListener הם:

on_press – במקרה של לחיצה על מקש, נכנס לפונקציה עם המקש הנלחץ כמשתנה.

on_release – במקרה של עזיבה של מקש, נכנס לפונקציה עם המקש הנלחץ כמשתנה.

ה-callbacks הקיימים ב-mouseListener הם:

on_click – במקרה של לחיצה על מקש בעכבר, נכנס לפונקציה עם המקש הנלחץ כמשתנה.

on_move – במקרה של הזזת העכבר, נכנס לפונקציה עם הקורדינאטות של המיקום החדש של העכבר (לאחר ההזזה) כמשתנים.

on_scroll – במקרה של גלילה של העכבר, נכנס לפונקציה עם הקורדינאטות של המיקום הנוכחי של העכבר, כמות הגלילה (מספר חיובי לגלילה מטה, מספר שלילי לגלילה מעלה, ערך הגלילה תלוי בחוזק הגלילה) כמשתנים.

כל ה-callbacks לוקחים את כל המשתנים שהם מקבלים, מאגדים אותם ל-tuple, ושמים אותם בטור שמוגדר בתוך ה-Listener.

הסיבה לטור היא שלפעמים המשתמש ילחץ על הרבה כפתורים ויעשה הרבה פעולות בו זמנית ולכן צריך טור, בנוסף הטור נחוץ להוצאת המידע משני ה-threads השונים, שכן שימוש בטור היא הדרך הכי מקובלת לעשות זאת.

את הטור לקחתי מהספרייה המובנית collections, אובייקט הטור הוא collections.deque.

ה-Listener מכיל כמה פונקציות:

__init__ - שמגדירה את הטור, ה-keyboardListener ואת ה-mouseListener.

start – שמפעילה את שני ה-threads.

stop – שעוצרת את שני ה-threads ומרוקנת את הטור.

fetch – מחזירה את הטור ומרוקנת אותו לפריים הבא.

השימוש ב-Listener הוא כך שבכל פריים נבצע fetch ובכך נשיג את כל הקלטים שהתבצעו בפריים הנוכחי, הוא משומש בצד הלקוח ששולח את ה-input שלו כדי לשלוט במחשב השרת.

executor.py

הקובץ הזה מכיל class שנקרא Executor שאחראי על ביצוע inputs בעזרת הספרייה pynput.

Executor מגדיר את ה-keyboardController ואת ה-mouseController שנדרשים כדי לשלוט במקלדת ובעכבר, שניהם מהחבילה pynput.

הפונקציות ב-Executor הן:

__init__ - מגדירה את ה-mouseController ואת ה-keyboardController.

execute – מפעילה thread על הפונקציה __execute,

__execute – הפעולה מקבלת queue שמוחזר מ-Listener, מתחילה להוציא ממנו את כל האלמנטים עד שהוא ריק, כל אלמנט הוא input בצורת tuple, לכן נעשית בדיקה באיזה סוג input מדובר לפי האלמנט הראשון ב- tuple, ואז ה-input מבוצע לפי סוגו.

"M" – תזוזת העכבר, ה- Executor מבצע תזוזה של העכבר לקורדינאטות שמצורפות.

"P" – לחיצה את מקש במקלדת, ה- Executor מצבע את הלחיצה של המקש המצורף.

"R" – עזיבה של מקש במקלדת, ה- Executor מבצע את העזיבה של המקש המצורף.

"C" – לחיצה על מקש בעכבר, ה- Executor מזיז את העכבר לקורדינאטות המצורפות ולוחץ על המקש המצורף.

"S" – גלילה של העכבר, ה- Executor מזיז את העכבר לקודינאטות המצורפות וגולל את העכבר לפי העוצמה שמצורפת.

close – הפונקציה מחכה שהקריאה הנוכחית ל-execute תסתיים, סוגרת את ה-thread, ועושה release לכל מקש במקלדת ובעכבר וזאת כדי לתקן באג שגרם למקש בעכבר או במקלדת עדיין להיות לחוץ אחרי שהקוד הסתיים (הבאג קיים מכיוון שהלקוח יכול לסגור את התוכנה בצד שלו בזמן שמקש מסוים לחוץ, כך שאף פעם לא נשלח event של release לשרת ולכן המקש לא נעזב בסיום הקוד).

השימוש ב-Executor הוא שבכל פריים שבו נקבל את הטור של ה-inputs נקרא לפונקציה execute כך שבכל פריים מתבצעים רק ה-inputs שנשלחו ממחשב אחר בפריים מסוים.

UI/UX

בפרויקט הזה השתמשתי בספרייה PyQt5 כדי ליצור את הממשק למשתמש.

הממשק יאפשר למשתמש ליצור session כך שמשתמש אחר יוכל להתחבר או יאפשר להתחבר ל-session שנפתח ע"י מחשב אחר.

כאשר שני משתמשים יתחברו יהיה ניתן להפעיל את שיתוף המסך (הסטרים יופיע באזור המסך השחור).

הממשק משתמש בקובץ json פשוט שנקרא "config.json" כדי לשמור את הגדרות המשתמש.

Palettes.py

קובץ פשוט שמגדיר שלושה סוגים שונים של Palettes: White, Dark, Midnight.

ה-Palettes הם בעצם הגוון של הממשק והם משפיעים על כל החלונות ועל כל האלמנטים בממשק, ניתן לשנות את הגוון דרך חלון הגדרות הממשק, להגדרה קוראים "theme".

כל ה-Palettes יורשים מהאובייקט QPalette שמובנה ב-PyQt5.

הגוון White הוא הגוון ברירת המחדל, יוצרים אותו ע"י אובייקט ריק שמאתחל את QPalette.

את הגוונים Dark ו-Midnight יוצרים ע"י שינוי הצבעים של כל אלמנט בנפרד (למשל שינוי צבע הכתב או שינוי ה-drop down menu).

ComboBoxes.py

קובץ זה נועד להכיל הגדרות של ComboBoxes מותאמים אישית.

כרגע יש רק אובייקט אחד שנקרא SortedComboBox שיורש מ-QComboBox שהוא אובייקט מובנה ב-PyQt5.

הסיבה לאובייקט היא שה-ComboBox שמובנה ב-PyQt5 מסרבל את האלמנטים שבתוכו בכל פעם שהמשתמש בוחר באפשרות אחרת ולכן בניתי אובייקט משלי שמשאיר את כל האלמנטים מסודרים לא משנה באיזה אפשרות בוחרים.

הרעיון הוא שנשמרת הרשימה של הסדר ההתחלתי שבו הוכנסו האלמנטים וכל פעם שהמשתמש בוחר באפשרות מסוימת, היא נהיית האפשרות הראשונה ושאר האפשרויות מוצגות בסדר לפי הרשימה.

LineEdits.py

קובץ זה נועד להכיל הגדרות של LineEdits מותאמים אישית.

כרגע ישנם שני אובייקטים: LockedLineEdit ו-IPLineEdit שיורשים מ-QLineEdit שהוא אובייקט מובנה ב-PyQt5.

LockedLineEdit הוא אובייקט של LineEdit שהמשתמש לא יכול לגעת בו, אי אפשר לכתוב לתוכו כלום, חוץ מאשר לכתוב לתוכו בדרך תכנותית.

אובייקט זה משמש כדי להציג טקסט קבוע למשתמש שלא ניתן לעריכה בצורת LineEdit, בקוד הוא משומש בחלון השמירה שבו ה-path לתיקייה שנבחרה ע"י המשתמש מוצג ב-LockedLineEdit.

IPLineEdit הוא אובייקט של LineEdit שלוקח רק קלטים שמכילים את הספרות 0-9 ונקודה, כלומר רק קלטים שיכולים להיות ב-ip (המשתמש לא יכול להכניס שום תו אחר).

כל פעם שהמשתמש רושם טקסט נוסף ל-LineEdit נעשית בדיקה האם כל המחרוזת שהמשתמש הכניס מורכבת רק מהספרות 0-9 ונקודה, אם כן לא קורה כלום אבל אם לא המחרוזת לא תודפס בתוך ה-LineEdit ותחזור למצבה הקודם (לפני הכנסת הקלט הפגום).

השימוש של אובייקט זה הוא רק בשדה ה-ip בחלון ההתחברות כדי למנוע הכנסת Ip שגויה.

DisplayBuffer.py

קובץ זה מגדיר את האובייקט DisplayBuffer שמשמש להצגת המסך של השרת.

זהו אובייקט מותאם אישית שיורש מ-QLabel שהוא אובייקט שמובנה ב-PyQt5, הסיבה לירושה מ-QLabel היא שהאובייקט הזה יכול להציג גם תמונות בעזרת התכונה pixmap.

לאובייקט הזה יש פונקציה אחת – updateBuffer, שמקבלת מערך של numpy שמייצג תמונה, מתרגמת את התמונה לסוג QImage שנתמך ע"י PyQt, ושמה את התמונה ב-Label.

באתחול DisplayBuffer, הפונקציה updateBuffer נקראת עם תמונה של מסך שחור.

ConnectionWindow.py

זהו קובץ שבו יש את class שנקרא ConnWin שמתאר תת חלון שמטרתו היא לאפשר למשתמש להתחבר ל-Session.

במסך יש רק QLabel שאומר "enter ip:" QLabel בלי שום טקסט שנקרא resText (result text) ואובייקט מסוג IPLineEdit שעליו הוסבר מקודם.

על המשתמש להכניס את ה-ip של המחשב שיצר את ה-Session שאליו הוא רוצה להתחבר וללחוץ על ה QButton עם הטקסט "connect".

עם הלחיצה הקוד יכנס לפונקציה שנקראת Apply שבה נבדק ה-ip שהמשתמש הכניס, ע"י הפונקציה validateIP שמבצעת split על מחרוזת ה-ip ובודקת האם כל מספר במחרוזת המחולקת הוא בין 0 ל-255, אם לא, ה-Label שנקרא resText יקבל את הטקסט "Invalid IP" ויקבל צבע אדום כך שההודעה תופיע למשתמש.

אם ה-ip נכון, resText יקבל את הטקסט "...Connecting", יקבל צבע לבן והקוד יקרא לפונקציה connect בקובץ main.py שעליה אסביר אחר כך.

SaveWindow.py

זהו קובץ שמכיל את האובייקט SaveWin שמייצג את תת החלון שקשור בשמירת הסטרים של השרת לקובץ וידאו.

החלון מכיל QLabel שאומר "output file directory:", אובייקט מסוג LockedLineEdit שהוסבר מקודם ומשמש להצגת ה-path של התיקייה שבה בחר המשתמש לשמור את קובץ הוידאו.

הקוד מתחיל בקריאה לפונקציה LoadDefaultPath, שמנסה לטעון את מה שכתוב בקובץ lastPath לתוך ה-path, אם הקוד נכשל, הוא טוען את ה-defaultPath שהוא "C:\untitled.mp4".

בנוסף ישנו QButton שכאשר לוחצים עליו, הקוד נכנס לפונקציה selectDir שבו נפתח QFileDialog שהוא חלון שבו המשתמש בוחר את התיקייה שבו הוא רוצה לאכסן את הסטרים.

לאחר בחירת התיקייה הפונקציה MemorizePath שבה ה-path לתיקייה שנבחרה נשמר לקובץ txt שנקרא lastPath, כך שכאשר המשתמש יפתח את החלון שוב, הוא לא יצטרך לטעון שוב את אותה התיקייה אלא היא כבר תופיע.

בנוסף ישנו עוד QLabel שאומר "output file name:" ו-QLineEdit, שבו המשתמש אמור לכתוב את שם הקובץ שיכיל את הסטרים.

כאשר המשתמש לוחץ על ה-QButton שנקרא apply, הקוד נכנס לפונקציה שנקראת Apply, שבה ישנה לולאה אינסופית שבה בכל iteration נבדק האם קובץ באותו ה-path ועם אותו השם שהמשתמש הכניס כבר קיים, אם כן מוסיפים לשם הקובץ (n), כאשר n הוא מספר הקובץ הפנוי הראשון שקיים לדוגמה: אם נרצה לשמור את הקובץ בשם file.mp4 והתיקייה מכילה את הקבצים הבאים:

file.mp4

file,mp4(1)

file.mp4(2)

file.mp4(3)

הקובץ שנרצה לשמור ישמר בשם file.mp4(4).

בנוסף הקובץ מכיל פונקציית עזר שנקראת DirOf שמקבלת path לקובץ ומחזירה את ה-Directory של הקובץ או התיקייה.

UISettingsWindow.py

זהו קובץ שמכיל את האובייקט UISettingsWin שמייצג את החלון שמכיל הגדרות שמאפשרות לשנות את ממשק המשתמש.

הקובץ מכיל QLabel שאומר "Theme:", ואובייקט מסוג SortedComboBox שעליו הסברתי מקודם ומאפשר למשתמש לבחור theme, כלומר את נראות הממשק מבחינת הצבע (palette).

בנוסף ישנה הגרה שמורכבת מ-QLabel שאומר "show stream fps:" ו-QRadioBox שמאפשר למשתמש לבחור האם להראות את ה-fps של הסטרים במהלכו או שלא.

עם לחיצה על ה-QButton שנקרא apply הקוד נכנס לפונקציה שנקראת Apply שבה נקראת הפונקציה ChangeTheme בקובץ main.py שמשנה את ה-theme, ושומרת את ההגדרות שנבחרו ב-dictionary של ההגדרות.

main_window_ui.ui

זהו קובץ שנוצר בעזרת התוכנה qt designer שהיא תוכנה ליצירת UI/UX בצורה ויזואלית ופשוטה, קובץ זה מתאר את החלון הראשי והוא נועד להיפתח רק ע"י qt designer.

main_window_ui.py

גם זה קובץ שמתאר את החלון הראשי, רק שהוא קובץ פייתון שאפשר להריץ.

כדי להשיג את הקובץ הזה הייתי צריך לתרגם את קובץ ה-ui לקובץ py ע"י שימוש בפקודה pyuic5 main_window_ui.ui -o main_window_ui.py שמגיעה עם התקנת qt designer.

main.py

זהו הקובץ המרכזי שיש להריץ כדי להפעיל את המערכת.

קובץ זה מכיל את ה-class שנקרא Window שמייצג את החלון הראשי של התוכנה.

פונקציית ה-__init__ של האובייקט קוראת לפונקציה setupUi בקובץ המוכן main_window_ui.py דרך ירושה, פונקציה זו מאתחלת את החלון לפי הקובץ main_window_ui.ui שנוצר ב-qt designer.

בנוסף ישנה קריאה לפונקציה ReadFromJson שפותחת את הקובץ config.json וטוענת את ההגדרות ששמורות שם, אם אין שם שום הגדרות/הקובץ לא קיים/הקובץ נהרס, ייווצר קובץ חדש ויטענו הגדרות ברירת המחדל.

פה גם מוגדר ה-settings dictionary שבו מוחזקות כל ההגדרות הנוכחיות, אם חלק בקוד משנה הגדרה כלשהי, הוא חייב לשמור את השינוי ב-dictionary הזה.

כל הפונקציות ששמן הוא לפי התבנית on_action_{ACTION}_triggered הן פונקציות events, כלומר הן נקראות כש-event מסוים מתרחש – יכול להיות שה-event קורה בגלל לחיצה של המשתמש על אחד מאפשרויות ה-menu או בגלל שהמשתמש לחץ על קיצור במקלדת שמקושר ל-event כלשהו למשל Ctrl+S גורם ל-event שקשור לפתיחת חלון השמירה לפעול.

ישנם events של פתיחת חלונות בעזרת הפונקציה OpenWin שמקבלת חלון ושם ופותחת חלון חדש לפי האובייקט שהועבר לפונקציה ושם החלון יהיה השם שהועבר לפונקציה.

ישנו event שנקרא Fullscreen שבו נקראת הפונקציה ToggleFullscreen שמשנה את הביט השלישי בערך שמחירה הפונקציה המובנת windowState (ערך זה אחראי על המצב של המסך המלא).

ישנו גם event שנכנס לפונקציה closeEvent, ה-event הזה נדלק כאשר המשתמש מנסה לסגור את התוכנה בדרך כלשהי (חוץ מאשר אם הוא מכריח את סגירת התוכנה).

בדיקות מיוחדות

בדיקת ip שהמשתמש מכניס כ-input – האובייקט ConnWin שמייצג את חלון ההתחברות מכיל את האובייקט IPLineEdit, שבעצמו הוא סוג של בדיקה שמכריחה את המשתמש להכניס רק את הספרות מ-0 עד 9 ונקודה כך שהמשתמש לא יוכל להכניס אותיות אחרות שלא קשורות ל-ip וכך הבדיקה הסופית על ה-ip תהיה פשוטה יותר.

הבדיקה השנייה מתבצעת אחרי שהמשתמש לוחץ על הכפתור "apply" בפונקציה validateIP.

הבדיקה מחלקת את ה-ip לפי הנקודות ואז בודקת האם אורך הרשימה שבה יש כל חלקי ה-ip היא באורך 4, אם לא, ה-ip לא תקין ולכן תקפוץ הודעה בחלון שתגיד "Invalid IP" באדום, אם כן הבדיקה ממשיכה.

לאחר מכן מתבצעת בדיקה האם כל אלמנט ברשימה הוא בין 0 ל-255, אם לא, ה-ip לא תקין ותקפוץ הודעת השגיאה, אם כן הבדיקה מסתיימת וקופצת הודעה בחלון שתגיד "Connecting..." בלבן.

בדיקת האם הקובץ שנשמר כבר קיים – האובייקט SaveWin שמייצג את חלון השמירה של הסטרים לקובץ מכיל בדיקה כאשר המשתמש לוחץ על הכפתור "apply".

הבדיקה בודקת האם השם של הקובץ כבר קיים ב-path שהמשתמש בחר בלולאה אין סופית.

בכל iteration של הלולאה נוסף לסוף שם הקובץ "(n)" כאשר n הוא מספר האיטרציות עד לאותו הרגע.

הבדיקה מסתיימת רק כאשר נמצא שם פנוי של קובץ שלא קיים.

רפלקציה

כשהתחלתי לעבוד על הפרויקט, לא היה לי מושג בעיבוד תמונה – הניסיון הראשון שלי לשלוח רצף של תמונות דרך הרשת היה כושל מכיוון שניסיתי לשלוח כל תמונה כמו שהיא.

אבל ככל שלמדתי יותר טכניקות כמו: threads, loseless/lossy compression, VNC encodings והגבלת ה-fps, כך המערכת הפכה ליעילה יותר.

היה לי כיף מאוד ללמוד דברים חדשים, שיטות שונות שלא ידעתי לפני וגם לתכנן מערכת בגודל כזה.

אם היה לי עוד זמן הייתי משפר את יעילות האלגוריתמים המרכזיים כך:

- פייתון היא שפה איטית מאוד ולכן יכולתי לכתוב חלקים ספציפיים שלוקחים הרבה זמן עיבוד בשפה כמו Cython שהיא דומה לפייתון רק הרבה יותר מהירה.

- מוסיף את היכולת לצפות במצלמה של המחשב הנשלט שנתמכת דרך הספרייה Vidgear שבה השתמשתי דרך היכולת שנקראת CamGear.

- מוסיף את היכולת לכמה מחשבים להתחבר ולשלוט במחשב אחד.

ביבליוגרפיה

ספריות חיצוניות:

VidGear - https://github.com/abhiTronix/vidgear

pyscreenshot – https://github.com/ponty/pyscreenshot

mss – https://github.com/BoboTiG/python-mss

opencv-python – https://github.com/opencv/opencv-python

colorlog – https://github.com/borntyping/python-colorlog

PyQt5 – https://www.riverbankcomputing.com/software/pyqt/

sjpeg – https://github.com/webmproject/sjpeg

pyzmq – https://github.com/zeromq/pyzmq

pynput – https://github.com/moses-palmer/pynput

pynput docs - https://pynput.readthedocs.io/en/latest/mouse.html

numpy – https://github.com/numpy/numpy

tqdm – https://github.com/tqdm/tqdm

requests – https://github.com/psf/requests

Pillow – https://github.com/python-pillow/Pillow

מקורות אחרים:

הסבר מפורט על פרוטוקול VNC ועל מימושו -
https://datatracker.ietf.org/doc/html/rfc6143#section-7.7

פרויקט לדוגמא שמציג את מימוש הפרוטוקול -
https://github.com/cair/pyVNC/tree/master/pyVNC

הסבר על השיטות שונות לעצור את התוכנית לזמן מסוים - https://blat-/blatnik.github.io/computerBear/making-accurate-sleep-function

אתר ליצירת דיאגרמות UML בצורה פשוטה - https://www.planttext.com/