



Silent
Net

תיק פרויקט Silent Net



מגיש: עומר כפיר (330869017) תיכון הרצוג כפר סבא יב'3

שם עבודה: Silent Net

שם מנחה: אופיר שביט

שם חלופה: הגנת סייבר ומערכות הפעלה

תאריך הגשה: 24/05/2025

תוכן עניינים

| | |
|----|--|
| 5 | מבוא |
| 5 | ייזום |
| 5 | תיאור כללי |
| 5 | הגדרת לקוח |
| 6 | מטרות ויעדים |
| 7 | בעיות תועלות וחסכונות |
| 7 | סקר שוק – חקירת פתרונות קיימים |
| 11 | סקירת טכנולוגיית הפרויקט |
| 12 | תיחום פרויקט |
| 13 | פירוט תיאור המערכת (אפיון) |
| 13 | תיאור מפורט של המערכת |
| 14 | פירוט על יכולות שהיא תעניק לכל סוג משתמש |
| 15 | פירוט בדיקות שמתוכננות לפרויקט |
| 17 | תכנון וניהול לוז |
| 18 | סיכונים |
| 20 | תיאור תחום הידע – פרק מילולי |
| 20 | פירוט מעמיק של היכולות |
| 27 | מבנה / ארכיטקטורה של הפרויקט |
| 27 | תיאור הארכיטקטורה של המערכת המוצעת |
| 27 | תיאור החומרה – רכיבים שונים והקשרים ביניהם |
| 28 | מראה גרפי של קשרי החומרה |
| 29 | תיאור טכנולוגיות רלוונטיות |
| 29 | שפות תכנות |
| 30 | מערכת הפעלה |
| 33 | תחומי עניין |
| 34 | תיאור זרימת המידע במערכת |
| 34 | האזנה שקטה על מחשב המשתמש |
| 35 | איסוף מידע בזמן אמת מן מחשב המשתמש |
| 36 | גיבוי נתונים אצל המשתמש בעת שרת כבוי |
| 37 | שליחת מידע המשתמש אל המנהל |
| 38 | איסוף המידע במחשב השרת ל DB |
| 39 | הצגת המידע באופן נגיש במחשב המנהל |
| 40 | הגדרת רמת בטיחות אצל המנהל |

| | |
|----|---|
| 41 | תיאור האלגוריתמים המרכזיים בפרויקט |
| 41 | ניסוח וניתוח של הבעיה האלגוריתמית |
| 41 | אלגוריתמים קיימים לפתרון הבעיה |
| 42 | הפנייה למקור רלוונטי |
| 42 | סקירת הפיתרון הנבחר |
| 43 | תיאור סביבת הפיתוח |
| 43 | פירוט כלי הפיתוח הדרושים לפיתוח |
| 43 | פירוט הסביבה והכלים הנדרשים לבדיקות |
| 44 | תיאור פרוטוקול התקשורת |
| 44 | תיאור מילולי של פרוטוקול התקשורת |
| 44 | דוגמה להודעות: |
| 45 | פירוט כלל ההודעות הזורמות במערכת |
| 47 | תיאור מסכי המערכת |
| 47 | מסך פתיחה |
| 48 | מסך הגדרות |
| 49 | מסך ראשי |
| 49 | מסך אישי |
| 50 | מסך יציאה |
| 51 | מסך טעינה |
| 51 | מסך שגיאה |
| 52 | מסך 404 |
| 53 | דיאגרמת מסכים |
| 54 | תיאור מבני נתונים |
| 54 | פירוט מבני נתונים |
| 54 | פירוט מאגרי המידע של המערכת |
| 57 | סקירת חולשות ואיומים |
| 57 | שכבת האפליקציה |
| 58 | הצפנה |
| 59 | שכבת התעבורה – פרוטוקול TCP |
| 61 | מימוש הפרויקט |
| 61 | סקירת כל המודולים/מחלקות וקשרי הגומלין ביניהם |
| 61 | מודולים/מחלקות מיובאים |
| 63 | מודולים/מחלקות מקוריים |
| 63 | manager.py |

| | |
|-----|-------------------------------------|
| 66 | server.py |
| 71 | DB.py |
| 75 | process_limit.py |
| 76 | encryption.py |
| 79 | protocol.py |
| 84 | cpu_stats.c |
| 84 | kClientHook.c |
| 86 | mac_find.c |
| 87 | protocol.c |
| 87 | tcp_socket.c |
| 89 | transmission.c |
| 90 | workqueue.c |
| 91 | hide_module.c |
| 91 | hide_tcp_sock.c |
| 92 | file_storage.c |
| 94 | בעיות אלגוריתמיות וקטעי קוד מיוחדים |
| 97 | בדיקות ותוצאות |
| 97 | בדיקות משלב האפיון |
| 99 | בדיקות נוספות למערכת |
| 100 | מדריך למשתמש |
| 100 | עץ קבצים |
| 102 | התקנת מערכת |
| 104 | משתמשי המערכת |
| 104 | מנהל |
| 105 | שרת |
| 106 | לקוח |
| 107 | רפלקציה |
| 108 | ביבליוגרפיה |
| 108 | נספחים |

מבוא

ייזום

תיאור כללי

הפרויקט הינו פלטפורמה שמאפשרת למנהל של חברה להשגיח על פעילות לקוחותיו מרחוק ולוודא שהם מבצעים את העבודה כראוי. הפלטפורמה על המחשבים של העובדים מסתירה את עצמה כך שהעובדים בחברה לא יוכלו להסיר את התוכנית מהמחשב שלהם לבד או לשבש פעילותה. הפרויקט מחולק לשלושה חלקים – העובדים, המנהל והשרת שמנהל את התקשורת כנגד שני הרכיבים האחרים. הפרויקט יעזור למנהל לנתר את פעולות עובדיו ולראות אילו עובדים פועלים בפועל ואילו לא.

בחרתי בפרויקט זה מכיוון שהוא אינו פשוט ויהווה לי אתגר ואף בנוסף הוא עוסק בעיקרו בעולם מערכות ההפעלה, עולם שמרתק אותי. הפרויקט דורש למידה על התנהגות של מערכת ההפעלה והתעסקות עם קוד של מערכת ההפעלה. המטלה הינה לא פשוטה ומביאה עימה מספר אתגרים: הסתרת הליך במערכת הפעלה והאזנה לפעולות שונות מבלי פגע ויכולות של המשתמש לדעת, העברת מידע שנאסף במהלך האזנה למחשב המנהל אשר משמש בתור שרת, תצוגה גרפית בצורה נוחה אצל מחשב המנהל ועוד...

הגדרת לקוח

המערכת פותחה בעיקר בכדי לתת מענה לחברות וארגונים בהם יש צורך לנטר את פעילות מחשבי העובדים במהלך שעות העבודה. הצורך המרכזי נובע מהרצון לוודא שהעובדים עושים שימוש מקצועי ויעיל במחשב שלהם, ולא מבזבזים זמן יקר על עיסוקים שאינם קשורים לעבודה – כלומר, שהעובדים נשארים פרודוקטיביים לאורך היום ואינם מנסים לתחכם את המנהל.

ייחודה של המערכת הוא בכך שהיא הופכת את תהליך הניטור לפשוט, נוח, ובלתי פולשני: אצל העובדים, המערכת פועלת ברקע בצורה שקופה וללא צורך בהתערבות מצדם, ואילו בצד השרת – אין צורך בתחזוקה שוטפת או התעסקות. עם סיום יום העבודה (ואף בזמן אמת, אם יבחר בכך), המנהל יכול לגשת לממשק ייעודי שבו מרוכזות סטטיסטיקות מפורטות שנאספו במהלך היום על פעילות כל אחד מהעובדים.

נתונים אלו מספקים תמונת מצב ברורה לגבי רמת הפעילות והמעורבות של העובדים, ומאפשרים למנהל לקבל החלטות מושכלות – אם בהקשר של מתן משוב וביקורת, ואם בבחירת עובדים שראויים לתגמול והערכה על עבודה רציפה וממוקדת. על אף שהשימוש המרכזי במערכת מיועד עבור משרדים ועסקים עם כוח אדם הפועל מול מחשב, ניתן ליישם את המערכת גם במגוון רחב של תרחישים נוספים: לדוגמה, בבתי ספר שבהם יש מעבדות מחשבים – ניתן לאפשר למורה לעקוב אחר פעילות התלמידים בזמן אמת, או בבתי עסק ציבוריים המציעים גישה למחשב בתמורה כספית, כמו אינטרנט-קפה – שם

המערכת יכולה לשמש לא רק ככלי לניטור אלא גם כאמצעי להבטחת שימוש בטוח ותקין במחשבים, ובעל המקום יכול לבדוק אם יש פעילות חריגה או תהליכים חריגים בהם הלקוחות משתמשים. באמצעות שילוב בין ניטור חכם, פעולה שקטה, ויכולת הצגת נתונים ברורים ומידיים – המערכת מביאה עמה פתרון טכנולוגי מודרני לצורך ניהולי שכיח, תוך שמירה על איזון בין מעקב אפקטיבי לבין פרטיות ותפעול נוח.

בנוסף לכך, המערכת בצד הלקוח היא מערכת חכמה המוטמעת ישירות במחשבי העובדים בצורה שמונעת כל ניסיון לשבש את פעילותה. תכונה זו קריטית במיוחד במקומות בהם קיימת חשש שהמשתמשים במחשבים ינסו לעקוף או להשפיע על הניטור. בזכות הגנה זו, המערכת מבטיחה כי המידע שנאסף ונשלח אל השרת הוא אמין, מדויק ומשקף באופן אמת את השימוש במחשבים, ללא סיכוי למניפולציה או שינוי מצד המשתמשים.

מטרות ויעדים

לפרויקט מספר מטרות מרכזיות אשר מתמקדות בפיתוח מערכת ניטור חכמה ויעילה, המוטמעת במחשבי העובדים בחברה, ומטרתה לוודא שימוש פרודוקטיבי ותקין במחשב במהלך שעות העבודה. מטרה חשובה היא להבטיח שהמערכת תפעל באופן שקוף וחסין בפני העובדים, כך שלא יוכלו לגלות את קיומה, להסירה או לשנותה.

לשם כך, יש צורך במערכת שמוטמעת ברמת מערכת ההפעלה בצורה עמוקה ומאובטחת, אשר מונעת מניסיונות להתערב בפעילותה או לעקוף את הניטור. בנוסף, המערכת צריכה לספק למנהלי החברה גישה נוחה, מהירה וברורה לנתונים שנאספו – תוך הצגה של מידע אמין ומדויק אודות פעילות המחשבים בזמן אמת או בדיעבד.

הנתונים יוצגו בממשק ידידותי, שיקל על המנהלים להבין את רמת הפרודוקטיביות של העובדים ולקבל החלטות מושכלות, בין אם במתן משוב בונה או בתגמול עובדים מצטיינים. הפרויקט שם דגש על איזון עדין בין שתי מטרות מנוגדות לכאורה: מצד אחד, המערכת חייבת להיות נגישה ונוחה לשימוש עבור המנהל, ומצד שני – עליה לפעול במחשבי העובדים בלב מערכת ההפעלה באופן מוסתר, חסין ושקט, מבלי לגרום להפרעה או להיות חשופה להתערבות מצד המשתמשים. תכונות אלו הופכות את המערכת לכלי אמין וחזק, המתאים לשימוש במגוון סביבות עבודה שבהן יש חשיבות לניטור דיסקרטי של פעילות העובדים.

בעיות תועלות וחסכוניות

המערכת באה לחסוך זמנים בשביל מנהלים בחברות. חברות אשר ישתמשו במערכת זו יוכלו באופן נגיש להאזין למחשבים של העובדים בחברה שלהם מבלי שהעובדים יוכלו לעשות דבר כנגד לכך. העובדים לא יוכלו למצוא או לשנות את התוכנית המאזינה בכדי "לעבוד" על המנהל שלהם, המנהל יידע על שלל פעילות המחשב שלהם מבלי יוצא מן הכלל (כמובן הפרטים החשובים, לא האזנה מלאה לכל event). המערכת פותרת את הבעיה שנוצרת אצל עובדים בחברות שאינם מנצלים את זמן העבודה באופן מיטבי ואף גם לעיתים לא מבצעים את העבודה הצפויה מהם במהלך העבודה ומנסים לתחמן את המנהל שלהם בכדי לחשוב שהם עובדים ובפועל הם לא. זוהי הינה בעיה חמורה שמפסידה כסף רב לחברות, וכאן בדיוק מגיעה המערכת שלי.

מבלי ידיעת העובדים המנהל יכול בכל רגע נתון להסתכל על נתוני המחשב של העובד אצלו בתוכנה, ומכך להבין אם העובד מבצע את עבודתו כראוי או שצריך להעיר לו ואף אם הדבר מתמשך לפטר את העובד מהחברה. המערכת מנגישה את המידע על הלקוחות השונים למנהל של החברה בכדי שיוכל לתצפת על עובדיו באופן נוח ויעיל, מבלי שיצטרך להתקשות עם תוכנה מסובכת, כמו שידוע כיום הרבה תוכנות בעלות פונקציונליות רבה מתקשות בלשמוע על ממשק משתמש נוח וקל לשימוש, ולכן המערכת תדע להציג את המידע באופן נוח מבלי שמנהל החברה יצטרך להתקשות איתה.

סקר שוק – חקירת פתרונות קיימים

בעידן הדיגיטלי של ימינו, כאשר רוב העבודה מתבצעת באמצעות מחשבים, נוצר צורך הולך וגובר בקרב מנהלים לנטר ולעקוב אחר פעילות העובדים שלהם. בעיה נפוצה שעמה מתמודדות חברות רבות היא חוסר יעילות בניצול זמן העבודה, כאשר עובדים משתמשים במחשבי החברה לפעילויות אישיות במהלך שעות העבודה. תופעה זו גורמת להפסדים כספיים משמעותיים ומשפיעה על פרודוקטיביות הארגון. כתגובה לבעיה זו, פותחו במהלך השנים האחרונות מגוון פתרונות טכנולוגיים המיועדים לסייע למנהלים בניטור פעילות העובדים. פתרונות אלו נעים מכלים פשוטים למעקב אחר זמן עבודה ועד למערכות מתוחכמות לניתוח התנהגותי מעמיק. במסגרת מחקר זה נבחן את הפתרונות המובילים בשוק, נתמקד ביכולותיהם ובמגבלותיהם, ונבין כיצד הם מתיישבים עם הצרכים הקיימים בשוק.

הפתרונות המובילים בשוק

אחד הפתרונות הבולטים ביותר בתחום הוא Teramind, המוגדר כפתרון מוביל לניטור עובדים. החברה הקנדית מאחורי המוצר פיתחה פלטפורמה מקיפה הכוללת מעקב אחר פעילות המחשב, צילומי מסך אוטומטיים, הקלטות וידאו של המסך, וכלי ניתוח התנהגותי מתקדמים. המערכת מאפשרת למנהלים לקבל דוחות מפורטים על שימוש באפליקציות שונות, זמן שהוסוקט באתרים ספציפיים, והקשות מקלדת מפורטות. עם זאת, אחד החסרונות המשמעותיים של Teramind הוא



שהוא דורש התקנה גלויה על מחשבי העובדים, כלומר העובדים מודעים לכך שהם מנוטרים. כמו כן, המערכת מאופיינת בעלות גבוהה החל מ-10 דולר לחודש למשתמש, וממשק מורכב הדורש הכשרה מקיפה.

פתרון נוסף שצובר פופולריות הוא Hubstaff, המתמקד בעיקר בעקיבת זמן עבודה וניטור פרודוקטיביות עבור צוותים מרוחקים וחברות קטנות עד בינוניות. הכלי מציע יכולות עקיבת זמן מדויקות, צילומי מסך במרווחי זמן קבועים, ומעקב אחר שימוש באפליקציות. אחת מהתכונות הייחודיות של Hubstaff היא יכולת GPS tracking עבור עובדים ניידים, המאפשרת למנהלים לדעת היכן נמצאים העובדים במהלך שעות העבודה. אולם Hubstaff מתמקד בעיקר בעקיבת זמן ופחות בניטור מעמיק של פעילות המחשב, והוא גם אינו מספק אפשרות לפעולה נסתרת מהעובד.



מערכת נוספת שראויה לציון היא ActivTrak, המספקת פיתרון ניטור מתקדם המתמקד בהבנה עמוקה של התנהגות עובדים וזיהוי מגמות פרודוקטיביות לאורך זמן. החברה פיתחה אלגוריתמים מתוחכמים לניתוח התנהגותי המסוגלים לזהות פעילות חשודה או לא רגילה במחשבי העובדים. המערכת מספקת דוחות מפורטים על שימוש באפליקציות, התראות בזמן אמת כאשר מתגלה פעילות חריגה, וממשק משתמש אינטואיטיבי יחסית. למרות יכולותיה המתקדמות, ActivTrak סובלת מעלות גבוהה החל מ-7 דולר למשתמש בחודש, ודורשת הרשאות מפורשות להתקנה על מחשבי העובדים.



כחלק מכלי ניהול זמן פשוטים יותר, בולט Time Doctor, המיועד בעיקר לעסקים קטנים וצוותים מרוחקים. הכלי מתמקד במעקב אחר זמן שמוסוקט בפרויקטים שונים, מספק צילומי מסך בתדירות נמוכה, ומנטר אתרים ואפליקציות בסיסיים. Time Doctor מציע אינטגרציה עם כלי שכר וניהול פרויקטים, מה שהופך אותו לפתרון נוח לחברות קטנות. עם זאת, הממשק הפשוט שלו אינו מתאים לחברות הזקוקות לניטור מתקדם ומעמיק, והוא לא מספק יכולות הסתרה מהעובדים.



בצד השני של הספקטרום נמצא Veriato (שנודע בעבר כ-SpectorSoft), המתמחה בניטור מתקדם וזיהוי איומים פנימיים בארגונים. זהו פתרון ברמה מתקדמת בעיקר באבטחת מידע וביטחון ארגוני. Veriato מציע יכולות ניטור נסתר של פעילות משתמשים, זיהוי איומים פנימיים באמצעות אלגוריתמים מתוחכמים, הקלטת מסכים בפורמט וידאו, וניתוח מתקדם של התנהגות העובדים. המערכת כוללת גם יכולות זיהוי ייחודיות שמסוגלות לזהות התנהגות חריגה או חשודה. אולם העלות הגבוהה של Veriato, המגיעה למעל 100 דולר למשתמש בחודש, והמורכבות הטכנית הגבוהה שלו הופכים אותו לפתרון שמתאים רק לחברות גדולות עם תקציבים נרחבים.



ניתוח מגבלות השוק הקיים

בחינה מעמיקה של הפתרונות הקיימים חושפת מספר פערים משמעותיים בשוק. ראשית, העלות הגבוהה של רוב הפתרונות המתקדמים הופכת אותם לבלתי נגישים עבור חברות קטנות ובינוניות. חברות אלו זקוקות לכלי ניטור יעילים אך אינן יכולות להרשות לעצמן להשקיע עשרות או מאות דולר לחודש עבור כל עובד. שנית, מורכבות הממשק והצורך בהכשרה מקיפה מהווים מכשול נוסף. מנהלים רבים אינם בעלי רקע טכני מתקדם ומעוניינים בפתרון פשוט ונוח לשימוש שלא ידרוש מהם זמן רב של למידה. הפתרונות הקיימים לרוב מתמקדים בפונקציונליות רחבה על חשבון קלות השימוש. פער נוסף הוא בתחום ההסתרה מהעובדים. רוב הפתרונות הקיימים דורשים התקנה גלויה או לפחות הרשאות מפורשות מהעובדים, מה שמפחית את האפקטיביות שלהם.

כאשר עובדים יודעים שהם מנוטרים, הם עלולים לשנות את התנהגותם באופן זמני או למצוא דרכים לעקוף את המערכת. בנוסף, חלק מהפתרונות המתקדמים, כמו Veriato, מתמקדים בעיקר באבטחת מידע וזיהוי איומים פנימיים ופחות בניטור פרודוקטיביות יומיומית. מנהלים רבים מחפשים כלי שיסייע להם להבין האם העובדים שלהם מנצלים את זמן העבודה ביעילות, ולא בהכרח לזהות איומי אבטחה מתוחכמים.

ההזדמנות בשוק

הפערים שזוהו בשוק הקיים יוצרים הזדמנות משמעותית לפתרון חדש המשלב את היתרונות של הכלים הקיימים תוך התמודדות עם המגבלות שלהם. פתרון אידיאלי צריך להיות נגיש מבחינת עלות, פשוט לשימוש, יעיל בהסתרה מהעובדים, ומתמקד באופן ספציפי בניטור פרודוקטיביות. המאפיינים הייחודיים של הפתרון המוצע, כולל הפעולה הנסתר באמצעות KLM, הממשק הפשוט והנוח למנהלים, והעלות האפסית, מציבים אותו כחלופה אטרקטיבית לפתרונות הקיימים. הפתרון ממלא בדיוק את הפער שבין הכלים הפשוטים והזולים לבין המערכות המתקדמות והיקרות, ומציע פשרה אידיאלית עבור חברות הזקוקות לניטור יעיל ללא הסיבוכיות והעלות של הפתרונות לעיל.

סיכום

הסקירה המקיפה של השוק מראה כי קיים צורך ברור בפתרון חדש ומתקדם לניטור עובדים. הפתרונות הקיימים, למרות איכותם, סובלים ממגבלות משמעותיות המקשות על חברות רבות לאמץ אותם. הפתרון המוצע מתמודד עם המגבלות הללו ומציע חלופה יעילה, נגישה ופשוטה לשימוש שתוכל לשרת בצורה מיטבית את הצרכים של מנהלים המעוניינים לשפר את פרודוקטיביות העובדים שלהם.

הפרויקט מציע שימוש נוח ופשוט אצל צד המנהל, אשר על המנהל יש לדעת פרטים מינימליים על מנת להפעיל את המערכת (יפורט בהמשך התיק פרויקט), ולעומת זאת העובדים לא יכולים לשבש את ניטור המידע של התוכנית בניגוד לחלק מהפתרונות הקיימים. זאת בניגוד לפתרונות כמו Time Doctor או Hubstaff, שבהם העובד יכול בקלות לעצור את תהליך הניטור או לשנות את הגדרותיו. במערכות אלו, העובד רואה את אייקון התוכנה בשורת המשימות ויכול לסגור אותה או להשהות את פעילותה בלחיצת כפתור פשוטה. בנוסף, בפתרונות קיימים כמו ActivTrak או Teramind, העובדים לעיתים קרובות מקבלים התראות או הודעות על כך שהם מנוטרים, מה שמאפשר להם להתכונן ולשנות את התנהגותם בהתאם.

יתר על כן, עובדים טכנים יכולים לזהות את התהליכים הפועלים ברקע ולנסות לחסום אותם באמצעות חומת אש או תוכנות אנטי-וירוס. הגישה החדשנית של הפרויקט המוצע מתמקדת ביצירת KLM (Key Logger Module) שפועל ברמה עמוקה יותר במערכת ההפעלה, מה שהופך אותו לבלתי נגיש לעובד הרגיל. המודול מתחבא בין תהליכי המערכת הרגילים ואינו מותיר עקבות גלויים בממשק המשתמש. כך, גם עובדים בעלי ידע טכני מתקדם יתקשו לזהות את קיומו של המערכת ולהשבית אותה.

יש לציין לטובת הפתרונות הקיימים כי הם מציעים תמיכה במגוון רחב של מערכות הפעלה שונות וגרסאות מגוונות. במערכת שלי, הצד המנוטר במערכת מוגבל לגרסה ספציפית של מערכת הפעלה Linux ועל כן הוא לוקה בחסר בחלק זה (חשוב לציין כי הצד הלא מנוטר לא מוגבל למערכת הפעלה ספציפית).

סקירת טכנולוגיית הפרויקט

במערכות הפעלה קיימת שונות רבה ומשמעותית – הן בין מערכות הפעלה שונות לחלוטין, והן בין גרסאות שונות של אותה מערכת הפעלה. שונות זו באה לידי ביטוי בממשקי המערכת, במנגנוני הליבה, בתמיכה בחומרה, ובתצורת ניהול המשאבים. כתוצאה מכך, קיימת חשיבות רבה לסביבה שבה תורץ מערכת כלשהי, שכן ייתכן שתכונות מסוימות שהמערכת נשענת עליהן אינן זמינות או אינן פועלות באותו אופן בכל גרסה או מערכת.

במקרה של המערכת הנוכחית, שמבצעת פעולות קריטיות בליבת מערכת ההפעלה, יש קושי ממשי להפוך אותה לדינמית וגמישה כך שתוכל לרוץ על טווח רחב של גרסאות או מערכות. משום כך, קיימת תלות חזקה בסביבת ההרצה, והמערכת מחויבת לפעול על גרסה מסוימת ומדויקת של מערכת ההפעלה שנבחרה מראש. התאמה זו נדרשת אך ורק בצד הלקוח – יש להדגיש כי צד השרת אינו כפוף לאותן מגבלות, והוא חופשי לפעול על מערכות אחרות. כדי לאפשר הרצה עקבית ונכונה של המערכת, ניתן להשתמש בטכנולוגיות כגון מכונות וירטואליות (VM), שיאפשרו יצירת סביבה זהה בכל הרצה.

מהי מכונה וירטואלית (VM)?

Virtual Machine – VM, היא תוכנה המדמה מערכת מחשוב שלמה, כולל מעבד, זיכרון, דיסק קשיח וכרטיס רשת. בעזרת טכנולוגיה זו ניתן להריץ מערכת הפעלה שלמה בתוך מערכת הפעלה אחרת, באופן מבודד ועצמאי. כך ניתן ליצור סביבת הרצה מדויקת ונשלטת, ללא תלות בחומרת המחשב הפיזי או בהגדרותיו. שימוש במכונה וירטואלית מאפשר לשחזר את תנאי הפיתוח והבדיקה בדיוק רב, מה שמבטיח יציבות, אמינות והתנהגות עקבית של המערכת – גם כאשר היא מועתקת למחשבים שונים.

שימוש במודולים של ליבת מערכת ההפעלה

המערכת הנוכחית עושה שימוש במודול קרנל – רכיב שפועל בתוך ליבת מערכת ההפעלה עצמה ומקבל גישה ישירה למשאבי מערכת רגילים. עבודה ברמת הקרנל מעניקה שליטה מלאה על תהליכים, זיכרון והתקני חומרה, אך גם חושפת את המערכת לתקלות קריטיות במקרה של חוסר תאימות. מכיוון שמודול קרנל תלוי באופן ישיר בגרסה הספציפית של הקרנל ובמבנה הפנימי שלו, כל שינוי או עדכון קטן במערכת ההפעלה עלול למנוע את טעינת המודול או לגרום להתנהגות בלתי צפויה. לכן נדרשת סביבת הרצה מדויקת שתשקף באופן מלא את תנאי הפיתוח – צורך שמכונה וירטואלית עונה עליו בצורה מושלמת. באמצעות VM ניתן להבטיח הרצה עקבית ובטוחה של המודול, ללא תלות בשינויים עתידיים במערכת ההפעלה המארחת.

סיכום

בזכות VM ניתן להבטיח את יציבות המערכת ופעולתה התקינה, תוך בידוד מגורמים חיצוניים שיכולים להשתנות ממחשב למחשב. לפיכך, הפרויקט מחויב לרוץ על אותה גרסה שבה פותח, ואין אפשרות או תמיכה בהרצה על גרסאות אחרות – אפילו אם הן שייכות לאותה מערכת הפעלה. כתוצאה מכך, כל מחשב שעליו תרוץ המערכת (כלקוח) חייב להיות מותאם מראש לגרסת מערכת ההפעלה הספציפית שנבחרה במהלך תהליך הפיתוח.

תיחום פרויקט

הפרויקט עוסק במגוון רחב של תחומים טכנולוגיים, שכל אחד מהם מהווה רכיב מרכזי בתפקוד ובמבנה של המערכת. להלן פירוט התחומים המרכזיים בהם יעסוק הפרויקט:

- ❖ מערכות הפעלה: הפרויקט מעמיק בעבודה מול מערכת ההפעלה, תוך שימוש בטכניקות מתקדמות הכוללות ביצוע hooking לפונקציות קריטיות במערכת. טכניקות אלו יאפשרו ירוט ושינוי של התנהגות הפונקציות על פי צרכי המערכת. בנוסף, תתבצע הסתרה מכוונת של מידע מפני תהליכים אחרים, במטרה לשמור על חשאיות הפעולה של המערכת. ביצוע פעולת hooking מאפשר בנוסף קבלת מידע מאותם פונקציות קריטיות ושליחתו אל השרת.
- ❖ רשתות תקשורת: כחלק מתהליך העברת המידע מהצד של המשתמש אל צד השרת ובתקשורת רציפה בין המנהל והשרת, המערכת תשתמש בפרוטוקולי רשת, ובפרט בפרוטוקול TCP. בחירה בפרוטוקול זה נובעת מהצורך בהעברת מידע אמינה ומהימנה, תוך הבטחת שלמות הנתונים בעת ההעברה. המידע שייאסף ממערכת ההפעלה ישודר בצורה סדירה ומבוקרת אל מחשב השרת, אשר יקלוט, יאחסן ויעבד את הנתונים בהתאם לדרישות המערכת.
- ❖ הצפנה ואבטחת מידע: לצורך שמירה על סודיות ואבטחת המידע בזמן ההעברה, המערכת תשתמש בשיטות הצפנה שונות. הצפנה זו נועדה למנוע חשיפת המידע על ידי גורם שלישי שינסה ליירט את התעבורה בין מחשב הלקוח לשרת. השימוש בהצפנה יבטיח כי המידע הרגיש הנאסף לא ייחשף לגורמים בלתי מורשים, וכי התקשורת בין רכיבי המערכת תתבצע בצורה מאובטחת.
- ❖ נושאים בהם הפרויקט לא עוסק: יש להדגיש כי המערכת אינה עוסקת בתחום של אבטחת מערכות מחשב או בהגנה מפני תוכנות זדוניות, אנטי-וירוסים, פיירוולים וכדומה. כמו כן, המערכת לא נועדה לשם ניהול סיסמאות או שמירתן במסדי נתונים (DB). כלומר, אין מדובר בפתרון אבטחה כולל, אלא במערכת ניטור ומעקב פנימית, הפועלת באופן דיסקרטי למטרות ניהול ובקרה מצד גורמים בעלי הרשאה.

פירוט תיאור המערכת (אפיון)

תיאור מפורט של המערכת

המערכת המדוברת היא מערכת האזנה שקטה, אשר פועלת על מחשבי המריצים את מערכת ההפעלה לינוקס. מטרת העל של המערכת היא לפקח על פעולתם התקינה והשוטפת של המחשבים, תוך שמירה על סודיות מוחלטת וללא כל אפשרות למשתמשי המערכת לזהות את נוכחותה. המערכת פועלת ברקע ומנטרת בצורה שקטה מתודות שונות המבוצעות על ידי תוכניות שונות שרצות על המחשב, וזאת במטרה לאפשר מעקב רציף, מדויק ויסודי אחרי כל פעולה המתבצעת במערכת.

באמצעות מנגנוני האזנה מתקדמים, המערכת יכולה לעקוב אחרי מגוון רחב של פעולות, ביניהן הרצת תוכניות חדשות, יצירת ושימוש בחיבורים לרשת, ועוד. המידע שייאסף יאפשר לקבל תמונה מלאה, מקיפה ורציפה על כל מה שמתרחש במחשב בזמן אמת, כך שכל פעילות תדווח למחשב השרת והמנהל יוכל לבחון זאת בעצמו ולהחליט על עבודת העובד.

הייחודיות של המערכת טמונה בכך שהיא לא רק מבצעת מעקב אלא גם שומרת על עצמה מוסתרת לחלוטין. גם אם עובד יידע מראש על קיומה, הוא לא יוכל לזהות אותה, להסיר אותה או להפריע לפעולתה. לשם כך, המערכת משתמשת בטכניקות חכמות ומתקדמות, כמו שינוי התנהגות של מערכת ההפעלה, שינוי דינמי של קריאות מערכת והתחמקות מגילוי או אמצעי ניטור אחרים.

המערכת עובדת באופן קבוע לאיסוף מידע, אשר ייארז בצורה מסודרת, מאורגנת ויעילה, תוך שמירה על הפרדה בין משתמשים שונים. המידע נשלח למחשב שרת מרכזי שאיתו המנהלים יכולים ליצור תקשורת ולהשיג את הנתונים ממנו השמורים על כל מחשב ומחשב, הנתונים שמורים בצורה ברורה, נוחה ונגישה. המידע יוצג לפי משתמשים ומחשבים, עם אפשרויות סינון מתקדמות, הצגת נתונים בזמן אמת, כולל הפקת דוח מפורט לפי הצורך.

באופן כללי, המערכת מהווה פתרון טכנולוגי מתקדם שמפשט ומייעל את ניהול העובדים והמשאבים הדיגיטליים של הארגון. בזכות אוטומציה של תהליכי הפיקוח ושמירה על דיסקרטיות מוחלטת, המערכת מאפשרת למנהל לקבל שליטה מלאה, לצפות בהתנהלות העובדים, לאתר בעיות במהירות, וכל זאת תוך שמירה על רמת ביצועים גבוהה מאוד וללא השפעה ניכרת על המערכת הנבדקת.

פירוט על יכולות שהיא תעניק לכל סוג משתמש

היכולות העיקריות שמספק הפרויקט ממוקדות בצד המנהל ולא בצד המשתמש (העובד). כפי שהוסבר קודם, מדובר בפרויקט שמטרתו לאסוף מידע על פעילות מחשבים של עובדים בארגון, תוך הסתרת פעילותו ככל האפשר. כלומר, אחד מיעדי הליבה של הפרויקט הוא לגרום לכך שהתוכנה תרוץ ברקע מבלי שהעובדים יהיו מודעים לקיומה. גם במקרה שהעובד כן מבחין בסימנים כלשהם המעידים על קיום התוכנה – לא תהיה לו היכולת להשפיע על פעולתה או להסירה.

לכן, מבחינת העובד, אין כל ממשק או תועלת ישירה מן המערכת, והיא אינה מספקת עבורו כל פונקציונליות. לעומת זאת, מנהל העובדים הוא הגורם שמפיק את מרב התועלת מהפרויקט. למנהל יש גישה למגוון רחב של יכולות, אשר מאפשרים לו לצפות בפעילות מחשבי העובדים בכל רגע נתון. הוא יכול לבדוק האם עובדיו פועלים בהתאם להוראות העבודה, לזהות דפוסי התנהגות חריגים או בלתי יעילים, ואף להסיק מסקנות לגבי איכות הביצוע של כל עובד.

הגישה לנתונים מתקבלת בצורה נוחה וישירה על מחשב המנהל, כשהמידע נשלח ומעודכן בזמן אמת, מה שמאפשר תגובה מיידי או תכנון שיפורים בתהליכי העבודה. יכולת זו תורמת תרומה משמעותית ליעול ניהול העובדים ולשיפור אפקטיביות העבודה בארגון.

בנוסף, מכיוון שהעובדים אינם מודעים למעקב, הם אינם יכולים לדעת מתי ובאיזה אופן מתבצעת הבקרה על פעילותם. מצב זה מעניק למנהל יתרון משמעותי: הוא מקבל תמונה אותנטית של ההתנהלות בפועל, ללא ניסיון של העובדים להסתיר או לזייף ביצועים. באמצעות הנתונים הללו, המנהל יכול לבצע הערכה אובייקטיבית של תרומת העובדים לארגון, לקבל החלטות מבוססות מידע לגבי שיפור נהלים, ארגון מחדש של משימות ואף קבלת החלטות קשות כגון פיטורים – במקרים בהם מתגלה כי עובדים מסוימים אינם ממלאים את תפקידם כנדרש לאורך זמן.

להלן הכותרות של היכולות שהמערכת תעניק לכל סוג משתמש –

1. האזנה שקטה על מחשב המשתמש.
2. איסוף מידע בזמן אמת מן מחשב המשתמש.
3. גיבוי נתונים אצל המשתמש בעת שרת כבוי.
4. שליחת מידע של מחשב המשתמש אל מחשב השרת.
5. איסוף המידע במחשב השרת אל תוך DB.
6. הצגת המידע באופן נגיש במחשב המנהל.
7. הגדרת רמת בטיחות אצל המנהל.

פירוט בדיקות שמתוכננות לפרויקט

בפרויקט הזה חשוב לבצע מספר בדיקות כדי לוודא שהמערכת מתפקדת כפי שתוכננה. הבדיקות מתמקדות בהעברת נתונים וביכולת המעקב אחר פעולות המשתמש, תוך שמירה על יציבות המערכת. ביצוע הבדיקות הינו שלב הכרחי בפיתוח הפרויקט שבלעדיו לא ניתן להכריע אם הפרויקט עובד כראוי.

1. בדיקת העברת נתונים לשרת המרכזי

| | |
|-----------|---|
| מטרה | לבדוק שהנתונים מהמחשב של העובד מועברים לשרת של המנהל באופן אמין |
| איך אבדוק | אגדיר שרת מדומה ואעקוב אחרי זרימת הנתונים ממחשבי העובדים. אשתמש בכלים להסנפה (Wireshark) לבדיקת חבילות רשת כדי לוודא שהמידע מגיע בשלמותו ובתזמון הנכון. |

2. בדיקת אמינות הנתונים

| | |
|-----------|--|
| מטרה | לוודא שהנתונים הנאספים ממחשבי העובדים משקפים את הפעולות שבוצעו במדויק. |
| איך אבדוק | אבצע פעולות מוגדרות מראש במחשב העובד, כמו פתיחת תוכנות ועבודה על מסמכים, ואשווה את הנתונים שנשלחו לשרת לאירועים בפועל. כך אוכל לראות שהמערכת עוקבת בצורה מדויקת אחר הפעולות. |

3. בדיקת השפעה על ביצועי המערכת

| | |
|-----------|---|
| מטרה | לבדוק שהמערכת לא פוגעת בביצועים של מחשבי העובדים |
| איך אבדוק | אבצע מדידת ביצועים (CPU, זיכרון) לפני התקנת המערכת ואחריה, ואשווה את התוצאות. כך אוכל לוודא שהמערכת לא מכבידה על המשאבים של המחשב ומאפשרת עבודה חלקה. |

4. בדיקת אבטחת הנתונים

| | |
|-----------|---|
| מטרה | לוודא שהתקשורת בין המנהל לשרת מוגנת. |
| איך אבדוק | אשתמש בהצפנה להעברת המידע ואבחן פרוטוקולי תקשורת מאובטחים (DH AES) על מנת לוודא שהנתונים נשארים חסויים. ומוגנים בפני האזנות של מחשבים ברשת. |

5. בדיקת החבאתו של תוכנת הלקוח

| | |
|-----------|--|
| מטרה | לוודא שהתוכנה המופעלת אצל הלקוח הינה מוסתרת מן הלקוח ואין ביכולתו לשבש את פעילותה |
| איך אבדוק | אפעיל את התוכנה אצל הלקוח ואבדוק על ידי תוכנות ופקודות שונות אם התוכנה נראית ללקוח – Wireshark, netstat, lsmod. החבאה מתוכנות ופקודות אלו ימנע מן הלקוח לשבש את פעילות התוכנה (לדוגמה - ללמוד איך התוכנית עובדת לפי הפקטות היוצאות ולחכות את התנהגות התוכנה ובכך לשבש את התוכנית). |

תכנון וניהול לוז

| מספר | פעילות | תאריך יעד | תאריך בפועל | הערות |
|------|---------------------------------|------------|-------------|--|
| 1 | הצעת רעיון פרויקט גמר | 17.10.2024 | 14.10.2024 | הצעת הפרויקט הינה פרויקט זה, לא התבצעו שינויים ברעיון הפרויקט המקורי |
| 2 | הצעה לפרויקט גמר | 30.10.2024 | 24.10.2024 | תכנון ראשוני של הפרויקט, הכלים והדרך ביצוע. |
| 3 | מסמך אפיון – הגשת ביניים ראשונה | 10.11.2024 | 6.11.2024 | הגשה חלקית של אפיון הפרויקט. |
| 4 | מסמך אפיון – הגשת ביניים שנייה | 20.11.2024 | 14.11.2024 | הגשה חלקית של אפיון הפרויקט. |
| 5 | מסמך אפיון – הגשה סופית | 30.11.2024 | 22.11.2024 | הגשה סופית של אפיון הפרויקט המלא. |
| 6 | הוכחת יכולת | 15.01.2025 | 12.01.2025 | ביצוע מעקב אחר פעולות מרכזיות ושליחה לשרת. |
| 7 | ממשק גרפי | 31.01.2025 | 30.01.2025 | ממשק גרפי מלא אצל המנהל. |
| 8 | פעילות מלאה של המנהל | 01.03.2025 | 27.02.2025 | המנהל יכול לעקוב אחרי העובדים ולנתר את עבודתם. |
| 9 | פעילות מלאה של הלקוח | 20.03.2025 | 22.03.2025 | פיתוח מלא של קוד הלקוח כולל כל הפיצ'רים. |
| 10 | סיום כתיבת הפרויקט | 31.03.2025 | 28.3.2025 | פיתוח מלא של הפרויקט ללא באגים. |
| 11 | סיום בדיקות הפרויקט | 17.04.2025 | 9.04.2025 | בדיקות מלאות של הפרויקט כמו שצוין באפיון. |
| 12 | תיק פרויקט | 30.04.2025 | 20.05.2025 | סיום כתיבת תיק פרויקט מלא. |
| 13 | הגשת פרויקט גמר | 24.05.2025 | 24.05.2025 | הגשת פרויקט מלא ותיק פרויקט. |

סיכונים

| הסיכון | תיאור הסיכון | רמת סיכון | תיאור דרכי התמודדות | מה בוצע בפועל |
|-----------------------------------|---|-----------|---|--|
| אי עמידה בלוח הזמנים | הגשה מאוחרת וחוסר עקביות בלוח זמנים המתוכנן. אי עמידה בלוח הזמנים משפיע על כל תהליך הפרויקט | קל | לדאוג לעקוב אחר הלוח זמנים. | דאגתי לעקוב אחר הלוח זמנים. |
| המידע שמועבר אינו מוצפן | המידע שמועבר מן המנהל לשרת אינו מוצפן ולכן אנשים שאינם חלק מהמערכת יכולים לפענח את ההודעות ולהשתמש בהם לטובתם | קשה | להשתמש בהצפנות שונות, הצפנה אסימטרית להעברת המפתחות בשביל הצפנה סימטרית בשביל העברת ההודעות | השתמשתי בהצפנות בין המנהל לשרת. הסכמה על המפתחות מתבצעת בתחילת התקשורת בין השניים. |
| ממשק משתמש לא יציב ולא נוח לשימוש | ממשק המשתמש GUI אינו נוח לשימוש מה שמקשה רבות על השימוש בפרויקט מצד השרת. | בינוני | שינוי ממשק המשתמש כך שגם מי שאינו מבין את תוכן הפרויקט יוכל להבין איך להשתמש בממשק המשתמש, בדיקה כנגד אנשים חיצוניים | יצרתי ממשק משתמש שנוח לשימוש ואף נראה נוח לעין, אינו מסובך ויוצר הגיון בקרב משתמשים אפילו כשאינם מבינים בעולם המחשבים. |
| קריסת השרת | השרת נפל בשל סיבה לא ידועה כזו או אחרת | קשה | שימוש באמצעי זהירות בכדי למנוע את קריסתו של השרת (try/except), מזעור של קריסת השרת כך כאשר באמת יקרוס השרת כבר אז תהייה בעיה חיצונית שהפרויקט לא אחראי עליה | קוד השרת ממזער את קריסתו על ידי כתיבתו כך שאם נתקל בבעיה כלשהי כזו או אחרת בזכות מנגנון try/except יתפוס את התקלה וידפיסה למסך. בנוסף הקוד הראשי של השרת מופרד מההתנהלות נגד כל לקוח בזכות threads, כלומר אם thread שמטפל בלקוח קורס, השרת ימשיך לפעול ולקבל לקוחות אחרים. |
| עומס יתר על השרת | השרת מוצף בלקוחות ומנהל מול מספר רב של לקוחות session מה שמכביד עליו ויקשה | קשה | הגבלתם של מספר הלקוחות שיכולים להתחבר לשרת, מספר קבוע מקסימלי שיקבע מספר גג של | כאשר השרת מופעל, המפעיל את השרת יגדיר כמות דיפולטיבית של מקסימום לקוחות שיכולים להתחבר, |

| | | | | |
|---|--|--------|---|---|
| לאחר מכן בעת התחברות המנהל יוכל אף הוא בעצמו להגביל את כמות הלקוחות (ההגבלה גם היא בטווח, בין 1-40 לקוחות). | לקוחות שיכולים בו זמנית לנהל session מול השרת | | על מחשב השרת לתפקד כראוי | |
| במקום לשמור כל הודעה המתקבלת על ידי הלקוח כתבתי את הקוד כך שימספר כל הודעה, מה שמצד אחד חוסך כמויות גדולות של אחסון, ומצד שני חוסך גם כוח מחשב כאשר מנסים לשלוח מידע מן DB. | מחיקת מידע אצל כל משתנה כאשר כמות המידע השמורה אצלו בטבלה עברה מכסה מסוימת שנקבעה מראש | קשה | ה DB שהשרת שומר בזמן אמת על לקוחותיו יגדל בצורה כה משמעותית מה שיכביד על השרת וביצועיו | DB של הלקוחות גדול מדי |
| הסתרתי את התוכנית בפני המשתמש, אין ביכולת המשתמש לבצע פעולות כנגד התוכנית משום שאין לו שום מידע עליה (הלקוח לא יכול לדעת על איזה פורט השרת מתקשר וסוג ההודעות הנשלחות אל השרת). | הסתרת התוכנית כך שתעבוד מאחורי הקלעים, עבודה עם ה kernel בכדי להחביא את התוכנית. בנוסף תדאג להסתרת כל ביצועיה, שליחת המידע למחשב חיצוני גם כן הוא יוסתר על ידי התהליך. | קשה | הלקוח הצליח לשנות את המערכת ששולחת מידע מהמחשב אל השרת, מה שיפגע בתפקוד הפרויקט | הלקוח משנה את המערכת אצלו במחשב |
| השתמשתי בפרוטוקול TCP, המידע מגיע באופן אמין כאשר אין שיבושים ברשת. | שימוש בפרוטוקול אמין TCP על מנת להבטיח שהמידע שיועבר יגיע בצורה אמינה. | קשה | הלקוח והשרת מתקשרים באופן לא אמין, לא כל המידע שהלקוח שולח מגיע בשלמותו אל השרת | תקשורת לא אמינה |
| כאשר מריצים את התוכנה אני מקצה מספיק משאבים למכונה הוירטואלית (משתנה בין מחשב למחשב, לרוב המחשבים שנבדקו בשביל ריצה מלאה צריך 2 מעבדים (RAM MB2048 | נתינה של מספיק משאבים לכל מכונה וירטואלית (RAM, NETWORK, CPU STORAGE) | בינוני | המשאבים שניתנו למכונה הוירטואלית עליה רץ הפרויקט לא מספיקים, ולכן מתקשה המכונה לעבוד לפי צרכי הפרויקט | מכונה וירטואלית אינה מקבלת מספיק משאבים מהמערכת |

תיאור תחום הידע – פרק מילולי

פירוט מעמיק של היכולות

| שם היכולת | האזנה שקטה על מחשב המשתמש |
|--|---|
| מהות היכולת | <p>האזנה שקטה על מחשב המשתמש מאפשרת לכך שאין באפשרות העובדים למחוק/לשבש דברים בתוכנה בזמן פעילותה, כלומר התוכנה תמיד תפעל ואך ורק המנהל הוא בעל היכולת לשנות דברים בתוכנה. בכך המנהל יכול בכל זמן נתון לתצפת על עובדיו מבלי שיוכלו "לעבוד" על התוכנית ולגרום לה לשלוח מידע לא אמין.</p> |
| אוסף פעולות/יכולות הנדרשות ליכולת הנ"ל | <p>אצל הלקוח</p> <ul style="list-style-type: none"> ○ ביצוע hook שונים על פעולות מבלי לפגוע במהות הפעולות ○ הסתרת התהליך ○ האזנה "קלה" כך שאינה משפיע על ביצועי המערכת של מחשב המשתמש |
| אובייקטים נחוצים | <ul style="list-style-type: none"> ○ KLM – Kernel Loadable Module, כלומר תהליך בהרשאות הגבוהות ביותר במחשב – קרנל ○ מחשב לקוח (עם מערכת הפעלה לינוקס) ○ הרשאות של המנהל על מחשב הלקוח |

| שם היכולת | איסוף מידע בזמן אמת מן המחשב המשתמש |
|--|--|
| מהות היכולת | <p>איסוף מידע בזמן אמת מן המחשב של המשתמש מאפשרת ליכולות אחרות בפרויקט זה, אסיפת המידע היא יכולת קריטית אצל מחשבי המשתמשים, והיא חלק גדול מפרויקט זה. איסוף המידע בזמן אמת יאפשר למנהל לתצפת על עובדיו בזמן אמת (ולא רק לקבל סיכום), כלומר בכל רגע נתון המנהל בעל יכולת לבדוק את עובדיו ואת עבודותיהם. איסוף המידע מתכוון ליכולת של התוכנית לשמור את המידע שניתן להשיג מן ההאזנה לתוך buffer'ים בצורה ברורה.</p> |
| אוסף פעולות/יכולות הנדרשות ליכולת הנ"ל | <p>אצל הלקוח</p> <ul style="list-style-type: none"> ○ ביצוע hook'ים שונים על פעולות מבלי לפגוע במהות הפעולות ○ שמירת המידע בצורת טקסט שניתן לקרוא ולהבין בצורה מונגשת ○ יצירת סדר במידע, כל פעולה שונה במחשב המשתמש עליה להיות מסודרת לפי סוג הפעולה. |
| אובייקטים נחוצים | <ul style="list-style-type: none"> ○ KLM – Kernel Loadable Module, כלומר תהליך בהרשאות הגבוהות ביותר במחשב – קרנל ○ מחשב לקוח ○ קבצים/באפרים לאחסון המידע באופן זמני |

| שם היכולת | גיבוי נתונים אצל המשתמש בעת שרת כבוי |
|--|---|
| מהות היכולת | <p>בזכות היכולת של הלקוח לאסוף בזמן אמת נתונים, עליו גם לאחסן אותם אצלו כאשר קיימות תקלות כאלו ואחרות בתקשורת או אפילו מחשב השרת נכבה. יכולת זה מכפה על תקלות כאלו שאינן בשליטת הפרויקט ובאות לפצות עליהן. גיבוי הנתונים אינו אינסופי אך מוגבל בגודלו על ידי המנהל בכדי לא להעמיס על מחשבי הלקוחות, הגודל נקבע לפי רצון המנהל. אם מחשב הלקוח נכבה אף הוא בעצמו מבלי לסגור את התוכנית באופן שלם, התוכנית תדע בעצמה לחזור למצב המקורי של גיבוי המידע, כלומר אין איבוד מידע גם כאשר מחשב הלקוח בעצמו נכבה בבורטליות.</p> |
| אוסף פעולות/יכולות הנדרשות ליכולת הנ"ל | <p>אצל הלקוח</p> <ul style="list-style-type: none"> ○ ביצוע hook שונים על פעולות מבלי לפגוע במהות הפעולות ○ שמירת המידע בצורת טקסט שניתן לקרוא ולהבין בצורה מונגשת ○ יצירת סדר במידע, כל פעולה שונה במחשב המשתמש עליה להיות מסודרת לפי סוג הפעולה. ○ התממשקות עם קבצים – פתיחה, כתיבה, קריאה, סגירה (ותוך כדי להישאר במצב Kernel). |
| אובייקטים נחוצים | <ul style="list-style-type: none"> ○ KLM – Kernel Loadable Module, כלומר תהליך בהרשאות הגבוהות ביותר במחשב – קרנל ○ מחשב לקוח ○ קבצים לאחסון המידע באופן זמני |

| שם היכולת | העברת המידע של המשתמשים למנהל |
|--|--|
| מהות היכולת | <p>העברת המידע של המשתמשים אל מחשב השרת מאפשרת למנהל לאחר מכן לקבל את המידע על עובדיו מן השרת ובכך מקשר את מחשבי העובדים למחשב השרת ואת השרת למנהל. המידע שנאסף על המחשבים נשלח בזמן אמת אל השרת ובכך התוכנית יכולה לבצע יכולות רבות על המידע שמתקבל מן המשתמשים ולאחר מכן לשלוח אותו למנהל כך שיוכל לראות אותו באופן נגיש.</p> |
| אוסף פעולות/יכולות הנדרשות ליכולת הנ"ל | <p>אצל הלקוח</p> <ul style="list-style-type: none"> ○ תוכנית שמאזינה לפעולות הלקוח. ○ שליחת המידע מן מחשב המשתמש לשרת. <p>אצל השרת</p> <ul style="list-style-type: none"> ○ קבלת המידע מן הלקוח ופענוח המידע. ○ שמירת המידע. ○ שליחת המידע לאחר הצפנה אל מחשב המנהל. <p>אצל המנהל</p> <ul style="list-style-type: none"> ○ בקשת מידע מן השרת. ○ קבלת המידע ופענוח המידע. |
| אובייקטים נחוצים | <ul style="list-style-type: none"> ○ מחשב לקוח ○ Socket ○ מחשב שרת ○ מחשב מנהל ○ מפתחות הצפנה – מנהל ושרת ○ קבצים/באפרים לאחסון המידע באופן זמני |

| שם היכולת | איסוף המידע במחשב השרת אל תוך DB |
|--|--|
| מהות היכולת | <p>איסוף המידע במחשב השרת אל תוך DB מאפשר למחשב השרת לשמור את המידע הנשלח מן הלקוחות בזמן אמת ולהפריד אותו לכל לקוח בנפרד. כלומר כל מחשב של עובד בחברה יהיה שמור בתוך ה DB של כלל העובדים, כך שיהיה ניתן להפריד בין העובדים השונים במערכת.</p> |
| אוסף פעולות/יכולות הנדרשות ליכולת הנ"ל | <p>אצל השרת</p> <ul style="list-style-type: none"> ○ איסוף המידע מן הלקוח. ○ פענוח המידע לפי הפרוטוקול ○ הכנסה ל DB את הערכים של הלקוח ממנו נשלחה ההודעה. |
| אובייקטים נחוצים | <ul style="list-style-type: none"> ○ מחשב לקוח ○ Socket ○ מחשב שרת ○ טבלאות DB אצל השרת |

| שם היכולת | הצגת המידע באופן נגיש במחשב המנהל |
|--|--|
| מהות היכולת | <p>הצגת המידע באופן נגיש במחשב המנהל מאפשרת למנהל של העובדים באופן יעיל להסתכל על המידע של מחשבי העובדים ובכך לקבוע את יעילות עבודתם ואם הם עובדים כראוי או שאינם מבצעים את עבודתם כלל, ולכן חשוב להציג את המידע באופן מסודר כך שהמנהל יוכל להסתכל בנוחות ובזריזות על המידע.</p> |
| אוסף פעולות/יכולות הנדרשות ליכולת הנ"ל | <p>אצל המנהל</p> <ul style="list-style-type: none"> ○ איסוף המידע מן השרת. ○ פענוח המידע לפי ההצפנה המוסכמת על ידי השרת והמנהל. ○ פענוח המידע לפי הפרוטוקול. ○ הצגה באופן ויזואלי על ידי GUI את נתוני ה DB בזמן אמת. |
| אובייקטים נחוצים | <ul style="list-style-type: none"> ○ מחשב לקוח ○ Socket ○ מחשב שרת ○ מחשב מנהל ○ טבלת DB אצל השרת ○ GUI במחשב המנהל ○ מפתחות הצפנה |

| שם היכולת | הגדרת רמת בטיחות אצל המנהל |
|--|---|
| מהות היכולת | <p>כאשר המנהל מתחבר לשרת לאחר ביצע תהליך Login עליו להגדיר את רמת הבטיחות שהינו רוצה בעת הפעלת המערכת אצלו, הכוונה היא שביכולתו להגביל את כמות הלקוחות שמחוברים בו זמנית לשרת ואף לתת לשרת רמת בטיחות שהשרת יפעל על ידו – רמת בטיחות תהיה כמות ההודעות שהשרת מסכים שאינן עובדות לפי הפרוטוקול שאיתו השרת מתקשר לפני שהשרת מנתק את לקוח זה.</p> |
| אוסף פעולות/יכולות הנדרשות ליכולת הנ"ל | <p>אצל המנהל</p> <ul style="list-style-type: none"> ○ הצגת המידע באופן ויזואלי – GUI ○ התחברות ותקשורת עם השרת ○ הצפנת הודעות מקצה לקצה עם השרת <p>אצל השרת</p> <ul style="list-style-type: none"> ○ הצפנה הודעות מקצה לקצה עם המנהל ○ הגבלת כמות לקוחות ○ הגבלת כמות הודעות "בלתי חוקיות" לפי רמת בטיחות |
| אובייקטים נחוצים | <ul style="list-style-type: none"> ○ Socket ○ מחשב שרת ○ מחשב מנהל ○ GUI במחשב המנהל ○ מפתחות הצפנה ○ אובייקטים של לקוחות בכדי שהשרת יוכל לספור לכל לקוח כמה הודעות "בלתי חוקיות" נשלחו ממנו |

מבנה / ארכיטקטורה של הפרויקט

תיאור הארכיטקטורה של המערכת המוצעת

תיאור התומרה – רכיבים שונים והקשרים ביניהם

בפרויקט ישנם רכיבים שונים שפועלים ביחד על מנת להביא את הפרויקט לעבוד בצורה מלאה ולאפשר למנהלים שונים לעקוב אחרי עבודת עובדיהם ועל אמינות עבודתם. ראשית נגדיר את מחשבי הלקוחות, אשר מהווים את אבני היסוד של הפרויקט. הלקוחות ישלחו את המידע אל מחשב השרת.

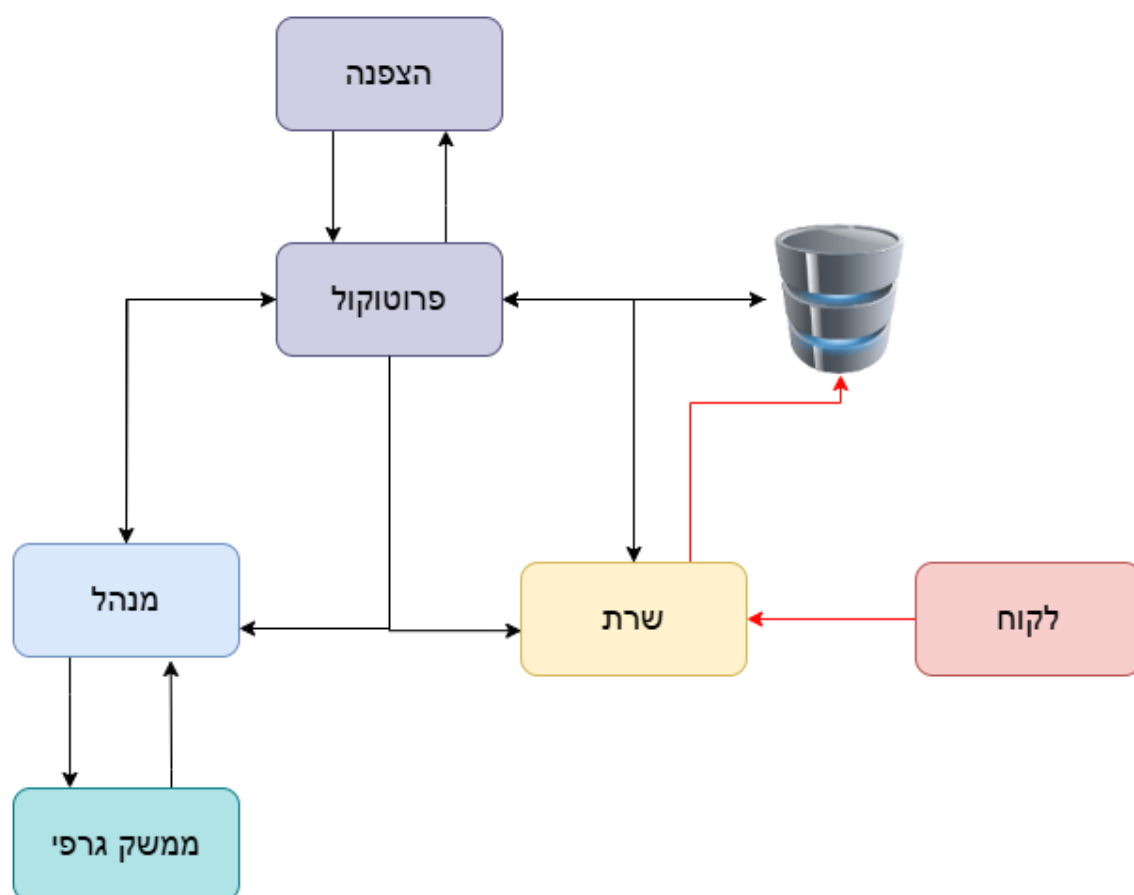
השרת יקבל את המידע מן הלקוחות דרך הרשת, המידע יועבר בצורה מוצפנת, הן בשימוש הצפנה אסימטרית להעברת המפתח והן בשימוש בהצפנה סימטרית בכדי להעביר מידע מוצפן עם המפתחות שהוחלפו בעזרת ההצפנה האסימטרית. מחשבי הלקוחות מתחברים לרשת באמצעות כרטיסי רשת המאפשרים להם לשלוח ולקבל נתונים בצורה מאובטחת.

חיבור הרשת בין הרכיבים מתבצע דרך פרוטוקול TCP אמין, המבטיח שהנתונים יישלחו בצורה יציבה ומוגנת. מחשב השרת, מקבל את המידע לפי הפרוטוקול המוסכם מהלקוחות ומעבד אותו בצורה מאובטחת. המחשב מצויד במערכת לניהול הצפנה המפענחת את המידע שהתקבל. המידע שנשלח מן הלקוחות אל השרת מפוענח אצל השרת ומתועד ישירות לתוך מסד הנתונים.

תוך כדי, השרת בתקשורת עם המנהל ישלח לו בצורה מוצפנת את המידע כאשר המנהל מבקש. המנהל מציג את המידע בצורה גרפית בממשק משתמש נוח וברור, תוך שמירה על הצפנה ואבטחת המידע לאורך כל התהליך.

להלן שרטוט המציג את הקשרים הללו -

מראה גרפי של קשרי התומרה



תיאור טכנולוגיות רלוונטיות

שפות תכנות

שפת תכנות C:

שפת C היא הבחירה המרכזית לפיתוח תוכנות ברמת ליבת מערכת ההפעלה. השפה מאפשרת גישה ישירה למשאבי המערכת ול-API של הליבה אשר משתמשים בה לצורך כתיבת מערכות הפעלה, מה שמעניק שליטה מלאה על תהליכים, זיכרון ורכיבי מערכת קריטיים. בעזרת C ניתן לבצע פעולות כמו עבודה עם מצביעים, ניהול זיכרון ישיר ותקשורת עם רכיבי חומרה או חלקים אחרים של המערכת.

שפת תכנות Python:

Python יכולה לשמש ככלי עזר לפיתוח סביבת העבודה של הפרויקט. היא אידיאלית לאוטומציה של בדיקות, איסוף נתונים ויצירת ממשקים לתקשורת עם רכיבי המערכת. Python מצטיינת בפשטות שלה וביכולות המתקדמות שלה בטיפול ברשתות, ניתוח קבצים ולוגים, ובניהול מהיר של משימות מורכבות.

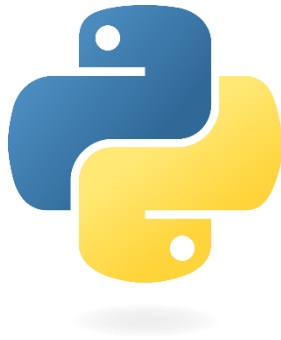
שפת תכנות SQL:

SQL היא כלי מרכזי לניהול ואחסון נתונים בצורה מסודרת. היא מאפשרת ליצור מסדי נתונים, לשמור מידע, ולעבוד איתו בקלות דרך שאילתות מובנות. בעזרת SQL ניתן לארגן מידע בטבלאות עם מבנה ברור, להגדיר קשרים בין נתונים שונים, ולשלוף בדיוק את המידע הדרוש בצורה מהירה ויעילה. השפה מתאימה לניהול כמויות גדולות של נתונים ומשמשת בסיס לרוב מערכות אחסון הנתונים המודרניות.

השילוב של C, Python ו-SQL נותן גמישות עצומה בעבודה. כל אחת מהשפות מביאה יתרונות ייחודיים, והעבודה ביניהן מאפשרת ליצור מערכת חזקה ומותאמת אישית. שפת C מאפשרת לעבוד ישירות עם המערכת, עם שליטה מלאה על הזיכרון, התהליכים והמשאבים של המחשב. ניתן לכתוב קוד יעיל שמבצע את הפעולות הקריטיות באופן מהיר ומדויק.

Python, לעומת זאת, נותנת דרך פשוטה ואינטואיטיבית ליצור סקריפטים שמנהלים תקשורת עם חלקי המערכת, מעבדים נתונים או מבצעים בדיקות. היכולת שלה לעבוד עם ספריות חזקות לרשתות ולעיבוד נתונים מקלה עליו לפתח חלקים מורכבים בצורה מהירה יותר.

SQL משתלבת כדי לאפשר לשמור נתונים בצורה מסודרת ולשלוף אותם בקלות לפי הצורך. במקום להתעסק עם אחסון נתונים גולמיים, אפשר להשתמש במבנה של מסדי נתונים כדי לשמור על סדר ולייעל את העבודה שלי. השילוב בין השפות האלה מאפשר לעבוד עם שכבת הליבה, ולאחסן ולשלוף את המידע בצורה מאורגנת ונגישה תוך כדי תקשורת בין הרכיבים השונים במערכת. כל שפה תורמת לתפקיד שונה, וביחד הן יוצרות מערכת מאוזנת ויעילה.



מערכת הפעלה

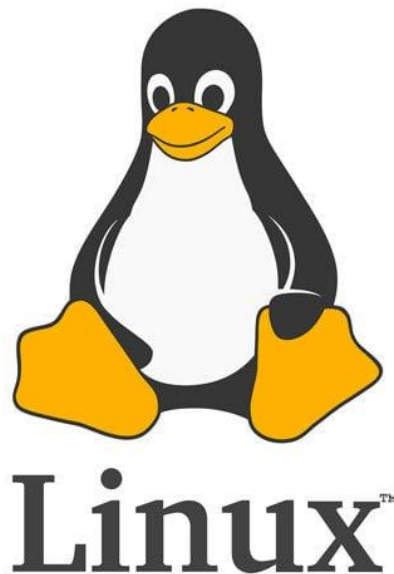
לינוקס (Linux) היא מערכת הפעלה מבוססת קוד פתוח, כלומר, הקוד שלה זמין לציבור וניתן לשנות אותו, להפיץ אותו ולהשתמש בו ללא מגבלות מסחריות. זו אחת הסיבות לכך שלינוקס נפוצה מאוד בקרב מפתחים, חוקרי אבטחת מידע וסטודנטים שמחפשים ללמוד ולהתנסות במערכת הפעלה ברמה עמוקה.

לינוקס בנויה במודל מודולרי, כך שהרכיבים שלה מחולקים לגרעין ספריות, ותוכנות מערכת. הגרעין אחראי על אינטראקציה עם החומרה, ניהול תהליכים, זיכרון ומערכות קבצים. העובדה שהקוד של הגרעין זמין מאפשרת למפתחים לחקור אותו, להבין את אופן הפעולה של המערכת, ואפילו לכתוב תוספות או שינויים כמו דרייברים.

מכיוון שלינוקס ניתנת להתאמה אישית מלאה, קל יחסית ליצור פרויקטים כמו פרויקט זה. ניתן לשנות את קוד הגרעין כדי להוסיף פונקציונליות או להתקין מודול קרנל KLM שמבצע פעולות מתקדמות, כמו גישה לזיכרון, ניהול תהליכים, או עקיפה של בקורות גישה. בנוסף, קהילת המפתחים של לינוקס גדולה, ויש תיעוד רב שיכול לסייע בלמידה ובהתמודדות עם אתגרים טכניים. הגמישות והנגישות של לינוקס הופכות אותה לפלטפורמה אידיאלית לפרויקטים חינוכיים ומחקריים, במיוחד בתחום אבטחת המידע והסייבר.

מכונה וירטואלית היא סביבה שמדמה מחשב עצמאי בתוך מחשב פיזי. בעזרתה ניתן להריץ מערכת הפעלה (כמו לינוקס) בתוך מערכת הפעלה אחרת, כך שיש בידנו סביבה מבודדת לחלוטין, שמנצל את המשאבים של המחשב הפיזי כמו מעבד, זיכרון ואחסון, ומחלק אותם בין מכונות וירטואליות שונות.

בפרויקטים שמצריכים עבודה על פיתוח ברמה נמוכה, כמו שינוי גרעין המערכת או עבודה עם מודולים קרנל, מכונה וירטואלית חשובה מאוד. היא מאפשרת ניסוי עם הקוד בסביבה מבודדת, כך שאין חשש לשבש את מערכת ההפעלה הראשית או את המחשב הפיזי. זה גם מאפשר ביצוע בדיקות וחקירות, תוך שמירה על הסביבה האמיתית מפני נזקים אפשריים.



תקשורת

התקשורת בפרויקט זה היא חלק מרכזי ומהותי בהצלחתו. המטרה היא לאפשר העברת מידע בין המערכת שבה רץ הקוד לבין מחשב אחר בצורה יעילה, אמינה ובטוחה. התקשורת מהווה את הגשר בין המידע שנאסף לבין היכולת לנתח ולהשתמש בו, ולכן תכנון נכון של מנגנון זה הוא חיוני. בפרויקט קיים שימוש בפרוטוקול TCP (Transmission Control Protocol) לצורך התקשורת.

TCP נבחר בשל היתרונות המשמעותיים שלו באמינות ובניהול מסרים. זהו פרוטוקול שמבטיח שכל המידע הנשלח מגיע ליעדו בדיוק כפי שנשלח, ובסדר הנכון. אמינות זו קריטית במיוחד במערכת שמטרתה להעביר נתונים מדויקים. TCP פועל באמצעות יצירת חיבור מבוקר בין שני מחשבים, תהליך המתחיל ב"לחיצת יד משולשת" (Three-Way Handshake) שמבטיחה ששני הצדדים מוכנים לתקשורת. תהליך זה מאפשר גם התמודדות עם בעיות כמו אובדן נתונים או הפרעות בתקשורת, כיוון שהפרוטוקול כולל מנגנוני תיקון וידוא.

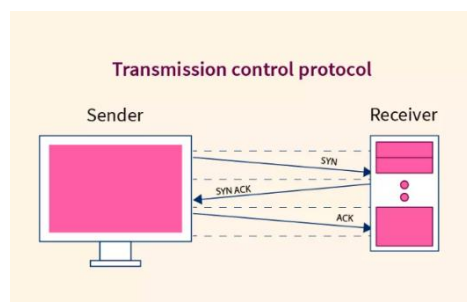
מרכיב חשוב נוסף בתקשורת בפרויקט הוא שימוש בנתונים מוצפנים. הצפנה היא כלי מרכזי להבטחת אבטחת המידע שנשלח, ובמיוחד כאשר מדובר בתקשורת שעשויה לכלול מידע רגיש או קריטי.

ההצפנה מגנה על הנתונים מפני גישה לא מורשית, כך שגם אם צד שלישי מצליח ליירט את התקשורת, הוא לא יוכל לפענח את המידע שנשלח.

בפרויקט, יש שילוב של מנגנוני הצפנה חזקים, כמו שימוש באלגוריתמים סטנדרטיים כגון DH ו AES, כדי להבטיח רמת אבטחה גבוהה. הנתונים המוצפנים יישלחו במסגרת חיבור ה-TCP, כך שאפשר להיות בטוחים שהמידע מוגן ומועבר בצורה בטוחה בין הצדדים.

שילוב הצפנה בתקשורת מבטיח שהמערכת לא רק פועלת ביעילות אלא גם משמרת את אבטחת המידע. האיזון בין אמינות, יעילות ובטיחות הוא קריטי להצלחת הפרויקט, והתכנון המוקפד של התקשורת נועד להשיג מטרה זו בצורה מיטבית.

הפרויקט פועל בתוך רשת מקומית. רשת מקומית היא מערכת תקשורת המחברת מספר מחשבים ומכשירים אחרים בתוך שטח גיאוגרפי מוגבל, כמו בית, משרד או מוסד חינוכי. ה-LAN מתאפיינת במהירות תקשורת גבוהה יחסית וביכולת לשלוט ברשת באופן ישיר, מה שהופך אותה לפתרון אידיאלי לפרויקט זה.



תחומי עניין

הפרויקט עוסק במגוון תחומי עניין טכנולוגיים המשלבים ידע תאורטי עם יישום מעשי. להלן תחומי העניין המרכזיים בפרויקט:

רשתות מחשבים ותקשורת נתונים

תכנון מנגנוני התקשורת בפרויקט נעשה תוך שימוש בפרוטוקול TCP, שמספק אמינות וניהול מסרים יעיל. נושא זה כולל הבנה מעמיקה של ניהול חיבורים, טיפול בהפרעות ושמירה על תקשורת תקינה בין רכיבי המערכת.

אבטחת מידע והצפנה

אחד ההיבטים המרכזיים בפרויקט הוא הגנה על הנתונים המועברים בזמן אמת באמצעות הצפנה. שילוב אלגוריתמים חזקים כמו AES ו DH מבטיח הגנה מפני גישה לא מורשית ושמירה על פרטיות המידע, גם אם הוא מיירט על ידי צד שלישי.

מערכות הפעלה וניהול משאבים

פיתוח מודול לייב בלינוקס מחייב ידע מתקדם במבנה מערכת ההפעלה וניהול משאבים. התחום עוסק בשימוש יעיל במנגנונים של המערכת והתממשקות עם לב מערכת ההפעלה מבלי לגרום להפרעה.

תכנון מערכות מורכבות

הפרויקט משלב בין מספר תחומי טכנולוגיה, מה שמחייב תכנון מקיף וחשיבה אסטרטגית. המערכת מתוכננת כך שתהיה יעילה, מאובטחת וקלה לתפעול, תוך שילוב פתרונות יצירתיים לאתגרים הנדסיים.

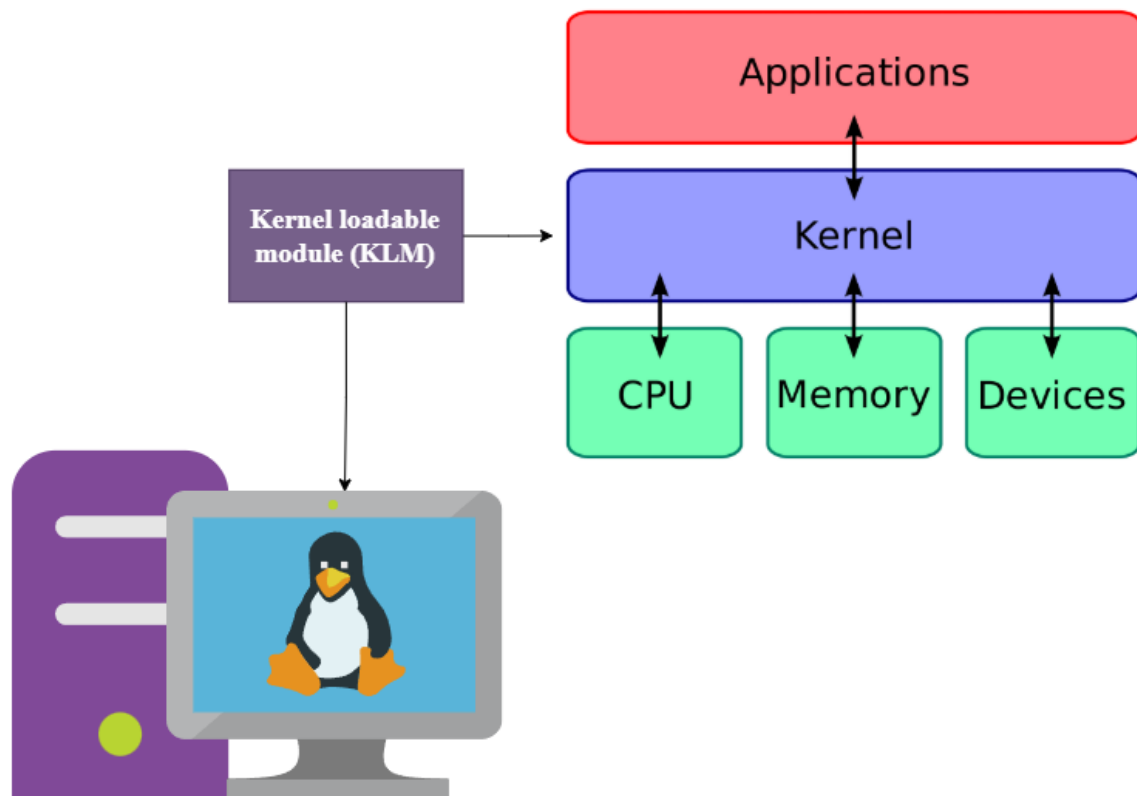
תצוגת נתונים בזמן אמת

אחד ההיבטים החשובים בפרויקט הוא הצגת הנתונים הנאספים באופן גרפי ברור ואינטואיטיבי בזמן אמת. המידע מוצג בממשק משתמש ידידותי, שמאפשר ניטור והבנה מיידיים של פעולת המערכת. תחום זה כולל שימוש בטכנולוגיות גרפיות ותכנון חוויית משתמש, תוך התחשבות על גמישות והתאמה למשתמשים.

תחומי העניין הללו מאפשרים ליצור פרויקט עשיר ומאתגר, שמחבר בין ידע טכנולוגי מתקדם לבין יישומים מעשיים בתחום מערכות ההפעלה, אבטחת המידע והנדסת התוכנה.

תיאור זרימת המידע במערכת

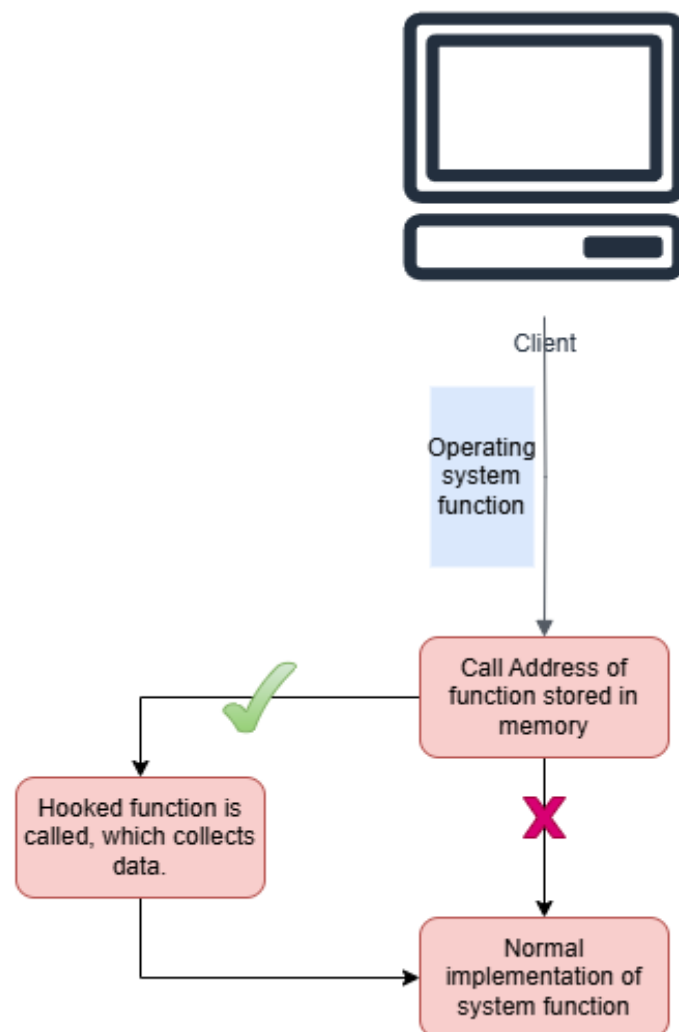
האזנה שקטה על מחשב המשתמש



תיאור הגרף:

גרף זה מציג את היכולת "האזנה שקטה על מחשב המשתמש". יכולת זו יכולה אך ורק לבוא לידי ביטוי על ידי שיתוף פעולה עם מערכת ההפעלה. ההאזנה השקטה מתבצעת באפשרות החבאת מידע מן המשתמש, שדבר זה אפשרי אך ורק בעזרת קוד מערכת הפעלה. דרך נגישה לעשות זאת מבלי לשנות את קוד מערכת ההפעלה עצמה (פעולה אפשרית, אך דורשת מכל מחשב שרוצה להריץ את פרויקט זה לשנות קוד במערכת ההפעלה שלו. כלומר פעולה ארוכה ולא יעילה, וגם מסוכנת אשר מדובר פה במערכת הפעלה, שהיא התוכנית הכי רגישה על המחשב, וכל שינוי מיותר בה יכול לגרור לסיכונים לכלל התנהלות המערכת) הינה על ידי השתמשות ב Kernel loadable module במערכת הפעלה לינוקס. לינוקס כמערכת הפעלה מאפשרת להוסיף אליה קוד אף תוך כדי זמן ריצה (מפה מגיעים המילים loadable module), הוספת אותו קוד הינו כלי חזק אשר התוכנית רצה בזיכרון של מערכת ההפעלה ובעלת גישה לנתונים רבים שאינם נגישים מהמשתמש. כפי שאפשר לראות בגרף, אפליקציות המשתמש יכולות רק לתקשר עם הקרנל, אך לא בעלות שינוי שלו. הוספת מודול משלנו למערכת הפעלה מאפשר לשנות את הדרך ואת המידע שאליו אפליקציות המשתמש חשופות אליו.

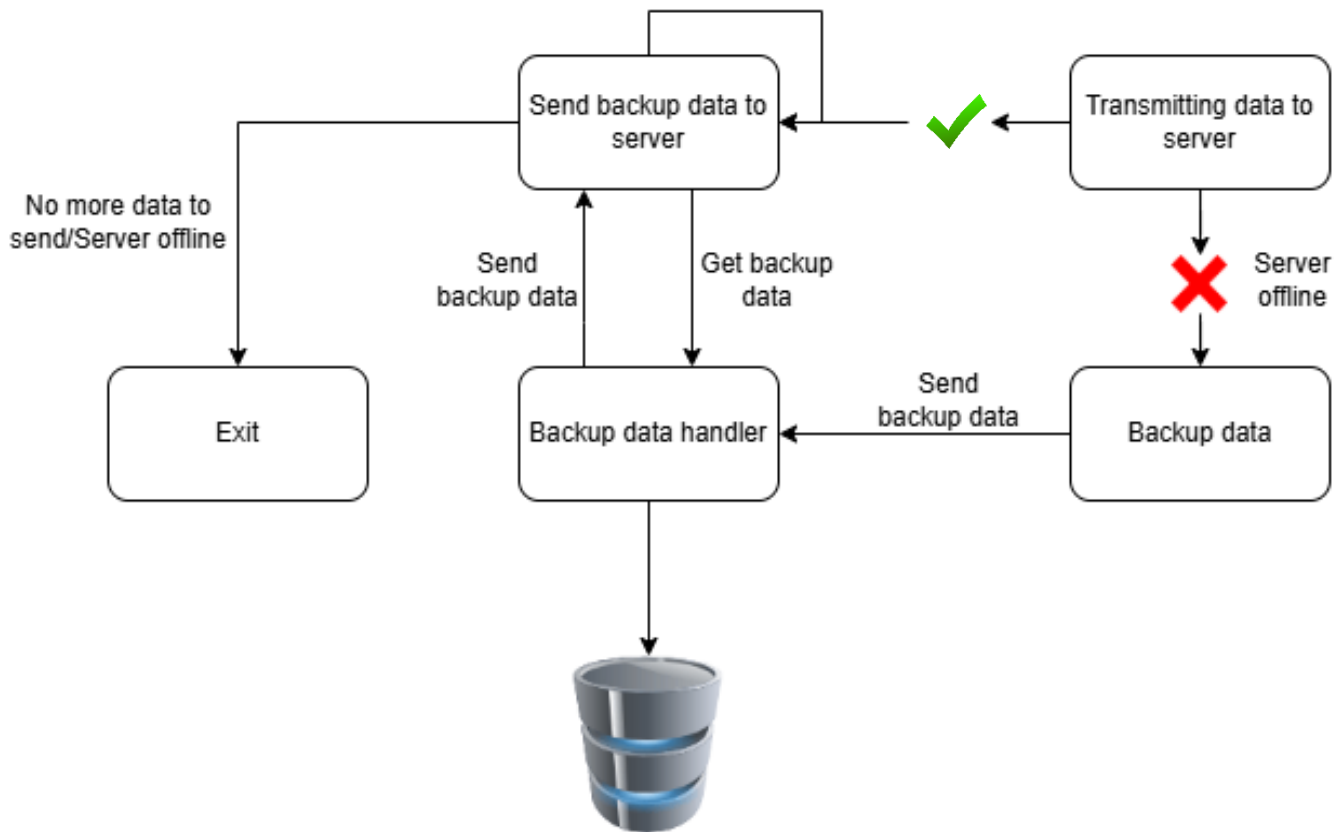
איסוף מידע בזמן אמת מן מחשב המשתמש



תיאור הגרף:

גרף זה מציג את היכולת "איסוף מידע" במחשבי המשתמשים. איסוף המידע מתבצע באמצעות מונח הנקרא Hooking, מונח זה הינו תהליך שבו אנו יוצרים ניתוב מחדש של תהליך זרימת הקוד ולמעשה מנתבים אותו דרך פונקציות שלנו שבעזרתן נוכל להחליט לבצע האזנה בסתר על המידע המועבר במערכת הפעלה כחלק מקריאת הפונקציות. כמו שניתן לראות גם לאחר שינוי הפעולה בכדי שנוכל להאזין, הפעולה המרכזית עדיין נקראת, כלומר לא התבצעה פגיעה בפעולת מערכת, אלא רק האזנה מאחורי הקלעים.

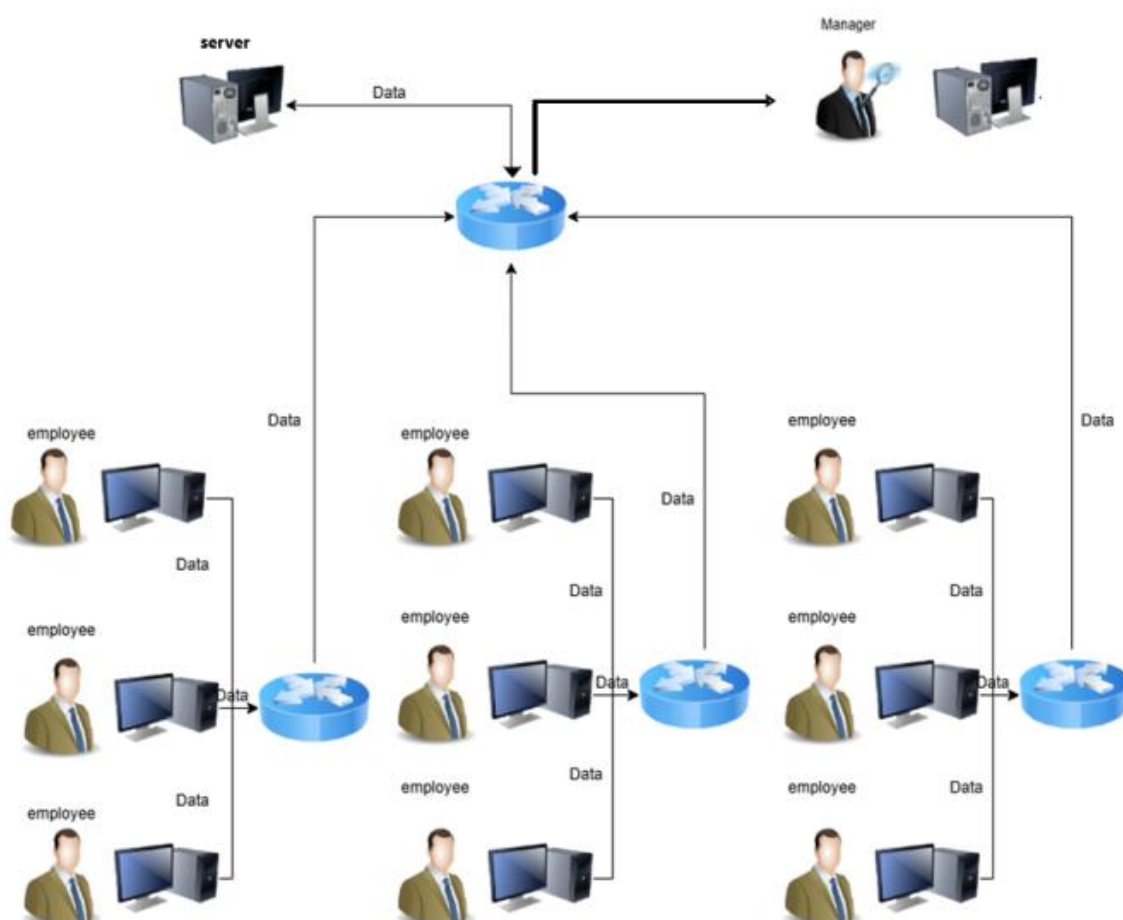
גיבוי נתונים אצל המשתמש בעת שרת כבוי



תיאור הגרף:

גרף זה מציג את היכולת "גיבוי נתונים" במחשבי המשתמשים. כאשר הלקוח מנסה לשלוח הודעה אל השרת והשרת איננו זמין (או כלשהי בעיה אחרת בשליחת ההודעה), הלקוח יגבה את המידע אצלו במחשב על ידי שימוש בממשק שידוע להתנהל כנגד מאגר הנתונים. כאשר הלקוח כן מצליח לתקשר עם השרת הוא יודא כי אין מידע ששמור במאגר גיבוי המידע, ולכן ינסה לרוקנו כל פעם שיש חיבור מוצלח עם השרת.

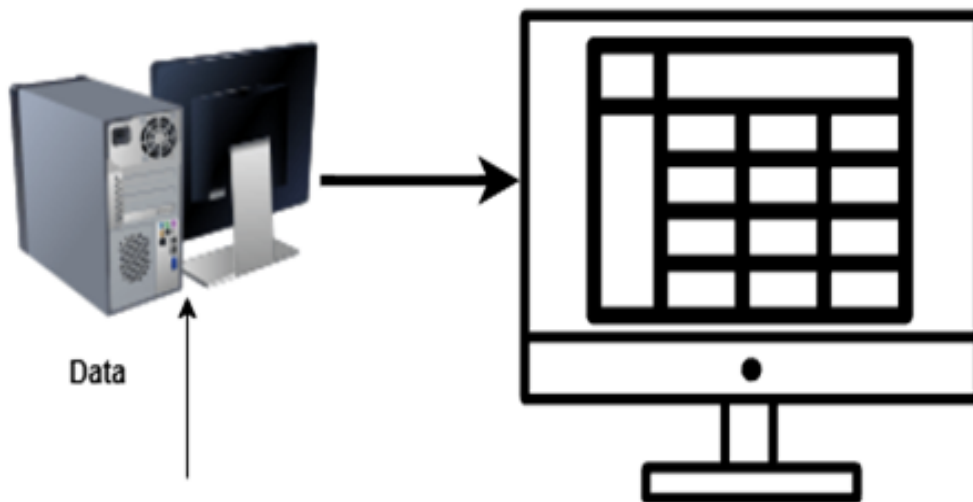
העברת המידע של המשתמשים למנהל



תיאור הגרף:

גרף זה מציג את היכולת "העברת המידע של המשתמשים למנהל". השליחה מתבצעת על ידי איסוף המידע מהמחשבים של העובדים והעברתו דרך רכיבי אינטרנט. כל עובד מחובר דרך הרשת לרכיב אינטרנט, אשר מרכז את המידע מכל תחנות העבודה. לאחר מכן, המידע נשלח ישירות למחשב של השרת, שם הנתונים נאספים ועוברים עיבוד לצורך ניתוח או הפקת דוחות, לאחר מכן נשלח למנהל. פעולה זו מאפשרת לארגון לרכז את המידע במקום אחד ולהבטיח שהמנהל מקבל גישה למידע חיוני בזמן אמת.

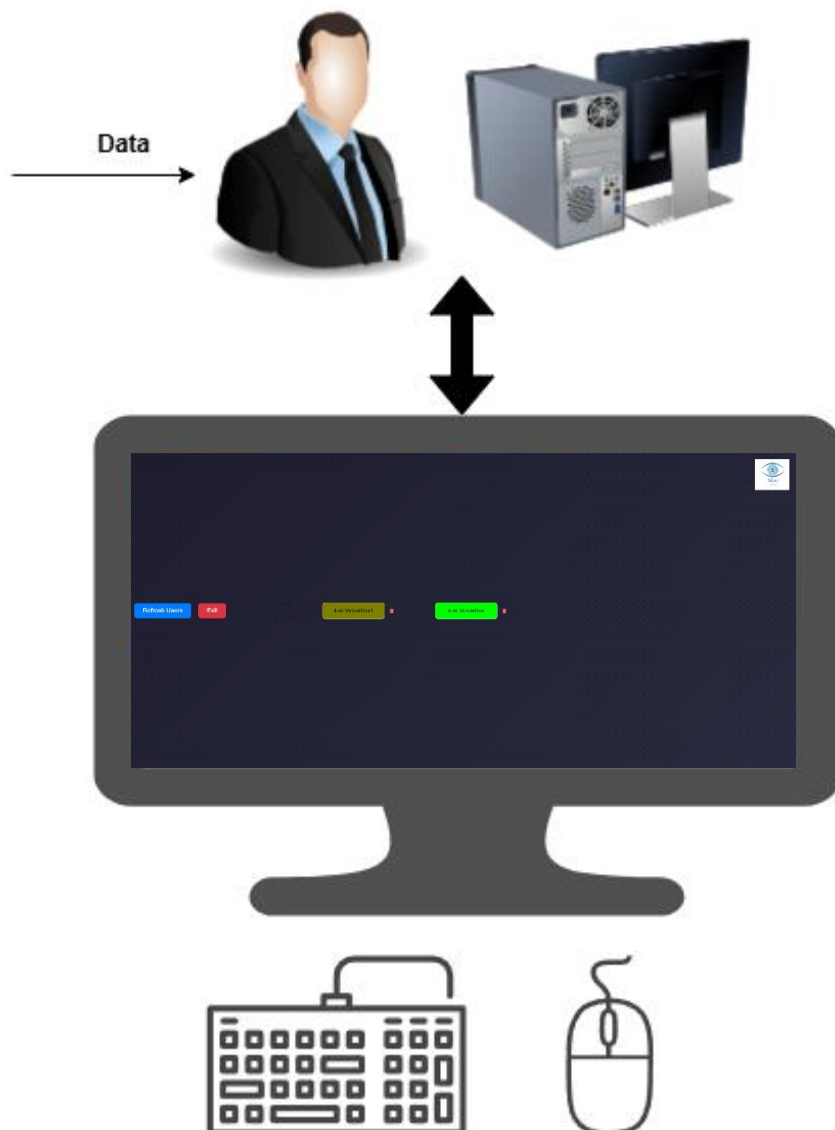
איסוף המידע במחשב השרת ל DB



תיאור הגרף:

גרף זה מציג את היכולת "איסוף המידע במחשב השרת ל DB". איסוף המידע מתבצע על ידי שליחת נתונים מהמחשבים המקומיים של העובדים אל השרת, שם הוא שומר את המידע בצורה מסודרת בתוך DB, ה DB יפריד בין המידע שהוא מקבל מהמשתמשים, וכך יוכל המנהל להבחין בין סוגי המידע שנשלחים אליו. תהליך זה מתחיל באיסוף הנתונים, מהתוכנות או מהפעילות של המשתמשים במחשבים שלהם. הנתונים מועברים דרך רכיב אינטרנט באופן מאובטח, ולאחר מכן מגיעים לשרת המרכזי, שם הם עוברים עיבוד ונשמרים בטבלאות מסודרות בבסיס הנתונים. תהליך זה מאפשר לארגון לנהל כמויות גדולות של מידע בצורה יעילה, להבטיח אחסון מאובטח של הנתונים, ולספק גישה מהירה לצורך ניתוח, קבלת החלטות או שיפור תהליכים עסקיים.

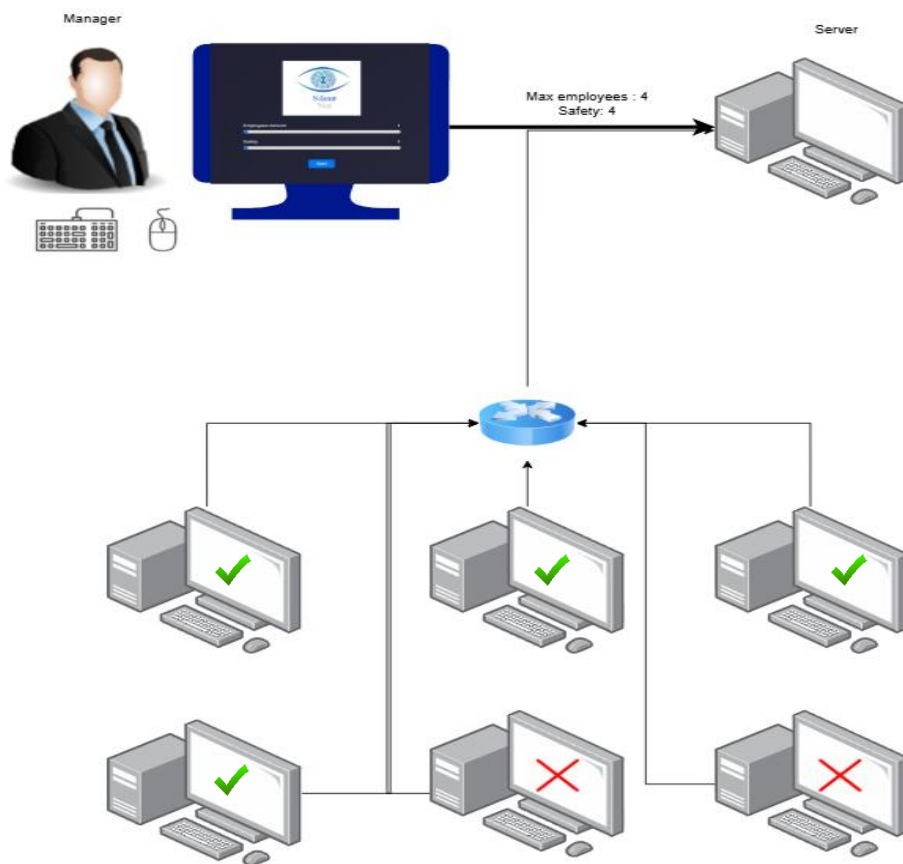
הצגת המידע באופן נגיש במחשב המנהל



תיאור הגרף:

גרף זה מציג את היכולת "הצגת המידע באופן נגיש במחשב המנהל". תהליך זה מתאר את הדרך שבה הנתונים שנשמרו בשרת מוצגים בצורה נוחה וקלילה למנהל. המידע שנשמר בבסיס הנתונים במחשב השרת ונשלח אל המנהל אשר מציג בצורה מאורגנת על מסך, למשל כטבלה או גרף, כך שמנהלים יכולים לגשת אליו בקלות, להבין אותו בצורה מהירה ולבצע עיבודים נוספים אם נדרש. המידע שמוצג אצל המנהל הינו דו"ח מפורט על מחשבי הלקוחות, ובכך המנהל יכול לנתח את פעילות העובדים ולדאוג שהם עובדים כראוי בצורה פשוטה.

הגדרת רמת בטיחות אצל המנהל



תיאור הגרף:

גרף זה מציג את היכולת "הגדרת רמת בטיחות אצל המנהל". הגרף מתאר איך המנהל יכול להגדיר אצל השרת עד כמה לקוחות הוא מרשה ומכך השרת מגביל את כמות הלקוחות המחוברים אליו בזמנית. בשל קושי ההמחשה בגרף, לא היה ניתן להמחיש את רמת הבטיחות שהמנהל יכול להגדיר, אך גם היא פרמטר שהמנהל מעביר לשרת בו הוא מחליט על כמות ההודעות הלא חוקיות שהשרת מרשה לכל לקוח ברצף. כפי שניתן לראות בגרף המנהל החליט כי כמות מקסימלית של לקוחות תהיה 4, ולכן רק ארבעה מחשבים הצליחו להתחבר אל השרת, בעוד השניים האחרים לא יכולים להתחבר לשרת ולא מקיימים session עימו.

תיאור האלגוריתמים המרכזיים בפרויקט

ניסוח וניתוח של הבעיה האלגוריתמית

הבעיה האלגוריתמית המרכזית בפרויקט שלי היא ניהול יעיל של אחסון הודעות, כך שלא תיגרם הצפה של המערכת במספר רב מדי של הודעות. הבעיה מתעוררת כאשר אין בקרה על כמות ההודעות, מה שעלול להוביל לשימוש מוגזם במשאבי מערכת כגון זיכרון או דיסק, ולגרום לקריסות או האטה משמעותית של התוכנה. דוגמה לכך היא כאשר הודעות נאגרות בקובץ או בזיכרון בקצב מהיר מהיכולת של המערכת לעבד ולמחוק אותן, וכתוצאה מכך גודל הקובץ או מבנה הזיכרון הולך וגדל ויוצר עומס הולך ומתגבר.

אלגוריתמים קיימים לפתרון הבעיה

הפתרון הנפוץ לבעיה זו הוא שימוש במבנה נתונים מעגלי (Circular Buffer), שבו האחסון מתנהל בצורה מעגלית: כאשר מגיעים לסוף המקום הפנוי, המצביע חוזר להתחלה ומתחיל לדרוס נתונים ישנים. כך, נשמרת שליטה על כמות הנתונים המאוחסנים, והמערכת לא מתמלאת ללא הגבלה. קיימות גישות שונות לניהול Circular Buffer: למשל, שימוש בשני מצביעים (ראש ו-זנב), ניהול מונה הודעות, ובדיקת מצבי מלא/ריק.

אפשר גם להגדיר מדיניות כמו דריסת הודעות ישנות אוטומטית או חסימת כתיבה כאשר אין מקום חדש עד שהתפנה מקום. לכל גישה יתרונות וחסרונות — דריסת נתונים ישנים מתאימה כאשר חשוב לשמור על זרימה רציפה, ואילו חסימה מתאימה כאשר לא רוצים לאבד מידע כלל.

מעבר לשימוש ב-Circular Buffer, קיימים פתרונות נוספים:

1. תור דינמי (Dynamic Queue): במקום להגדיר גודל קבוע מראש, אפשר להשתמש בתור מבוסס רשימה מקושרת או מבנה נתונים מתרחב, כך שהמערכת מוסיפה מקום לפי הצורך. כאשר מספר ההודעות חורג מסף שנקבע מראש, ניתן למחוק הודעות ישנות (למשל, בשיטת FIFO) או להפעיל תהליך ניקוי. פתרון זה גמיש יותר אך דורש מעקב מדויק אחרי כמות ההודעות ותחזוקה שוטפת של הזיכרון.

2. מבנה נתונים עם עדיפות (Priority Queue): במקרים שבהם יש חשיבות שונה להודעות, ניתן לאחסן אותן במבנה מסוג Priority Queue, כך שהודעות חשובות יקבלו עדיפות בטיפול או שמירה. זה מאפשר להיפטר קודם מהודעות פחות קריטיות כאשר המקום מתמלא. פתרון זה מתאים למערכות שבהן לא כל ההודעות שוות ערך, אך מורכב יותר מבחינת מימוש וניהול.

3. סבב קבצים (Log Rotation): כאשר ההודעות נשמרות בקובץ, אפשר ליישם מנגנון של סבב קבצים – כאשר קובץ הלוג מגיע לגודל מסוים, נפתח קובץ חדש, והקובץ הישן נמחק או נדחס. כך, שומרים על נפח דיסק מוגבל בלי לאבד שליטה על גודל הקבצים. פתרון זה נפוץ במערכות לוגים ומעקב, אך דורש תיאום בין מנגנוני הכתיבה והסבב, דבר המקשה על כתיבת הפרויקט במיוחד כאשר הוא מודול קרנלי.

הפנייה למקור רלוונטי

כותבי ויקיפדיה. (2025). Circular buffer. ויקיפדיה.
https://en.wikipedia.org/wiki/Circular_buffer

כותבי geeksforgeeks. (2022). Implement dynamic queue using templates class and a circular array geeksforgeeks.

<https://www.geeksforgeeks.org/implement-dynamic-queue-using-templates-class-and-a-circular-array/>

כותבי geeksforgeeks. (2025). What is Priority Queue | Introduction to Priority Queue. geeksforgeeks.

[/http://geeksforgeeks.org/priority-queue-set-1-introduction](http://geeksforgeeks.org/priority-queue-set-1-introduction)

כותבי ויקיפדיה. (2021). Log rotation. ויקיפדיה.

https://en.wikipedia.org/wiki/Log_rotation

סקירת הפיתרון הנבחר

הפתרון שבחרתי — Circular Buffer בקובץ עם דריסה אוטומטית — מאפשר שמירה על ביצועים גבוהים ללא צורך בתחזוקה מורכבת. באמצעות ניהול פשוט של מצביעי קריאה וכתיבה, ניתן לוודא שהקובץ לעולם לא יגדל מעבר לגבול מוגדר, וכך למנוע קריסה או האטת המערכת. נבחר לא להשתמש בגישות אחרות, כיוון שהן מוסיפות מורכבות מיותרת ואינן מתאימות לצורך הבסיסי של שליטה קלה וגמישה בגודל הנתונים. התממשקות עם קובץ שמתנהג כמקום אחסון מעגלי מביאה עימה את היכולת לאחסן בצורה יעילה ללא שימוש בפקודות שונות של מערכת ההפעלה הקשורות לקבצים. מימוש בצורה נכונה של אלגוריתם זה מייעל את הליכי הפרויקט ואף מבטיח על כמות מוגדרת מראש של מידע שלא תלך לאיבוד כאשר השרת לא פעיל אך הלקוח מבצע את עבודתו. מוטב לציין כי בפרויקט יש לשמור בתוך הקובץ עצמו את שני המצביעים של הקובץ, אשר יש להתחשב במקרה כאשר מחשב הלקוח קורס ויש מידע בתוך קובץ הגיבוי, לכן לאחר כל קריאה/כתיבה נעדכן את מצביעים אלו בסוף הקובץ.

תיאור סביבת הפיתוח

פירוט כלי הפיתוח הדרושים לפיתוח

במהלך פיתוח הפרויקט יש שימוש בסביבה שמבוססת על מכונה וירטואלית, המאפשרת להפעיל גרסה ספציפית של מערכת לינוקס הדרושה להרצת הקוד. כך ניתן ליצור סביבה מבודדת, עצמאית וגמישה, שמאפשרת לבצע פיתוח בצורה מבוקרת וללא השפעה על מערכת ההפעלה הראשית של המחשב. לצורך כתיבת הקוד אני משתמש בשני עורכי טקסט עיקריים:



VIM – עורך טקסט פופולרי ויעיל במיוחד בסביבת לינוקס, המאפשר עבודה מהירה וקלה עם קבצים ללא צורך בממשק גרפי, ומיועד למי שמכיר את הפקודות שלו.



VSCode – עורך מודרני ומתקדם, עם אינטגרציה מלאה לכלים נוספים כמו ניהול גרסאות, דיבאגינג ותוספים מגוונים, שמקלים משמעותית על תהליך הפיתוח.

בנוסף, נעשה שימוש ב-Git ו-GitHub לניהול גרסאות הקוד.



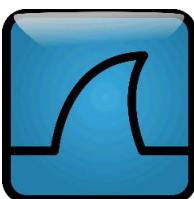
Git מאפשר לעקוב אחר כל שינוי בקוד, לשמור גרסאות שונות ולנהל את תהליך הפיתוח בצורה מסודרת ויעילה.

GitHub משמש כפלטפורמה לשיתוף הקוד, עבודה שיתופית עם צוותים, וגיבוי הקוד בענן.

פירוט הסביבה והכלים הנדרשים לבדיקות



סביבה בה יש שימוש בפרויקט הינה מבוססת על מכונה וירטואלית, שמאפשרת להגדיר ולהפעיל גרסה מסוימת של מערכת לינוקס לפי הצורך, מבלי להשפיע על המערכת הראשית. מכונות וירטואליות כמו VirtualBox או VMware מאפשרות לנהל את הסביבה בצורה גמישה ובטוחה, ולבצע ניסויים ובדיקות בלי חשש.



לניתוח התקשורת בין רכיבי המערכת, אני משתמש בכלי Wireshark – כלי מצוין ללכידת וניתוח תעבורת הרשת. בעזרתו ניתן לבדוק את זרימת המידע בין השרתים ללקוחות, לאתר בעיות תקשורת, לבחון שימוש נכון בכתובות IP, ולוודא שהקוד פועל כשורה בכל ההיבטים של התקשורת.

בנוסף, לצורך איתור שגיאות ופתרון בעיות בפרויקט בחלק שנכתב בשפת python, נעשה שימוש ב-Python Debugger דרך VSCode. הדיבאגר מאפשר לעקוב אחרי ביצוע, לבדוק ערכים בזמן ריצה ולזהות נקודות תורפה מדויקות.

תיאור פרוטוקול התקשורת

תיאור מילולי של פרוטוקול התקשורת

החשיבות של פרוטוקול תקשורת בפרויקט שלי היא מכרעת, שכן הוא מבצע את התיאום וההבנה בין כל הרכיבים במערכת. פרוטוקול תקשורת מספק את הכללים וההנחות שדרכם רכיבי המערכת יכולים להבין אחד את השני ולהעביר מידע בצורה מסודרת וברורה. אם הפרוטוקול מוגדר בצורה טובה, הוא הופך את התקשורת לפשוטה יותר ומובנת לכל הצדדים המעורבים. בפרויקט זה פרוטוקול התקשורת הינו פרוטוקול סינכרוני בין רכיבי התקשורת כאשר קיימת תלות בין סוגי ההודעות בתקשורת בין המנהל לשרת, כלומר לסדר של ההודעות. (בין העובד לשרת לאחר התחברות ראשונה אין תלות).

מבנה הפרוטוקול תקשורת בו אשתמש בפרויקט הינו במבנה הבא:

1. אורך הודעה (כארבעה תווים שמהווים מספר)

2. סוג הפעולה שהתבצעה (המתודה שנקראה/איזה פעולה המשתמש ביצע)

3. המידע הקשור לאותה פעולה (פרמטרים)

התו המפריד להודעות יהיה הבית '1f' (בזכות היותו בית שלא נמצא באף הודעה בין חלקי הפרויקט השונים, לכן לא יתערבב עם חלקי הודעה שונים). חשוב להתייחס לכך שאם התו המפריד יהיה חלק מהחלק השני של ההודעה, תיווצרנה בעיה בפרוטוקול, לכן יש הימנעות מלשלוח הודעה שכוללת בחלק השני שלה את התו הנ"ל. לא תהיינה בעיה שהתו יופיע כחלק מהחלק השלישי של ההודעה, אשר ברגע שנמצא התו המפריד הראשון, שם נדע לבצע את ההפרדה בין חלקי ההודעה. כאשר יהיה שימוש בתו המפריד בחלק השלישי של ההודעה דבר זה יהיה בכדי לבצע הפרדה בין הפרמטרים של פקודה.

12.

דוגמה להודעות:

0006CIE\1f48 (שימוש באינפוט של חומרה)

0010CPO\1fchrome (פתיחת תהליך).

ועוד...

פירוט כלל ההודעות הזורמות במערכת

| שם ההודעה | מבנה שדות ההודעה | נשלח מ-/אל- | תיאור |
|-----------|---|-------------|--|
| CAU | תעודת זהות כרטיס רשת (MAC) ולאחר מכן שם הוסט (hostname) | לקוח אל שרת | הודעה הנשלחת כל תחילת תקשורת בין לקוח לשרת |
| CPO | שם תהליך | לקוח אל שרת | שם תהליך שנפתח על ידי הלקוח |
| CCU | מספרי ליבה ואחוזי שימוש בה מופרדים עם תו ההפרדה בין כל ליבה. לאחר הליבה האחרונה יופיע גם השעה בה נמדדו נתונים אלו – 0,56\x1f1,98,datetime | לקוח אל שרת | אחוזי שימוש בכל ליבה במעבד של המחשב |
| COT | קטגוריה של סוג אפליקציה | לקוח אל שרת | אפליקציות אליהן הלקוח מתקשר ברשת |
| CIE | מספר | לקוח אל שרת | קלט מכל חומרה המתקבל במחשב הלקוח מתקבל כקוד, אותו קוד נשלח לשרת |
| MME | אין (חוץ מסוג ההודעה ואורכה) | מנהל אל שרת | המנהל מודיע אל השרת על סיום החיבור ביניהם |
| MST | כמות לקוחות מקסימלית ולאחר מכן רמת בטיחות | מנהל אל שרת | לאחר התחברות המנהל אל השרת הוא ישלח לו את ההגדרות הרצויות למנהל – כמות לקוחות מקסימלית המחברים לשרת ורמת בטיחות של השרת כנגד כל לקוח |
| MGC | אין (חוץ מסוג ההודעה ואורכה) | מנהל אל שרת | המנהל מבקש מן השרת את הנתונים הכלליים על הלקוחות המחברים |
| MGD | שם הלקוח | מנהל אל שרת | המנהל מבקש את שלל הנתונים על לקוח ספציפי |
| MDG | שם הלקוח | מנהל אל שרת | המנהל שולח לשרת שם של לקוח שהוא רוצה למחוק את הנתונים שלו מהשרת |
| MCC | אין (חוץ מסוג ההודעה ואורכה) | מנהל אל שרת | המנהל בודק אם השרת פתוח לתקשורת, אחרת המנהל לא יהיה שימוש במערכת של המנהל |
| MCN | השם הנוכחי של הלקוח והשם החדש של הלקוח | מנהל אל שרת | המנהל מבקש להחליף שם של לקוח |
| MCH | אין (חוץ מסוג ההודעה ואורכה) | שרת אל מנהל | השרת מעדכן את המנהל כי עדכון השם הצליח |
| MMP | אין (חוץ מסוג ההודעה ואורכה) | מנהל אל שרת | המנהל מעדכן את השרת שהוא הולך לשלוח לו את הסיסמה |

| | | | |
|-----|---|----------------------|--|
| MIC | אין (חוץ מסוג ההודעה ואורכה) | שרת אל מנהל | השרת מעדכן את המנהל שהסיסמה ששלח לא נכונה |
| MVC | אין (חוץ מסוג ההודעה ואורכה) | שרת אל מנהל | השרת מעדכן את המנהל שהסיסמה ששלח תקפה והחיבור יוצא לפועל |
| EXH | ערכי DH – יכול להיות גם g/p וגם המפתח הציבורי | שרת/מנהל אל מנהל/שרת | החלפת מפתחות שמתחרשת בתחילת התקשורת בין השרת למנהל |

סוגי שגיאות:

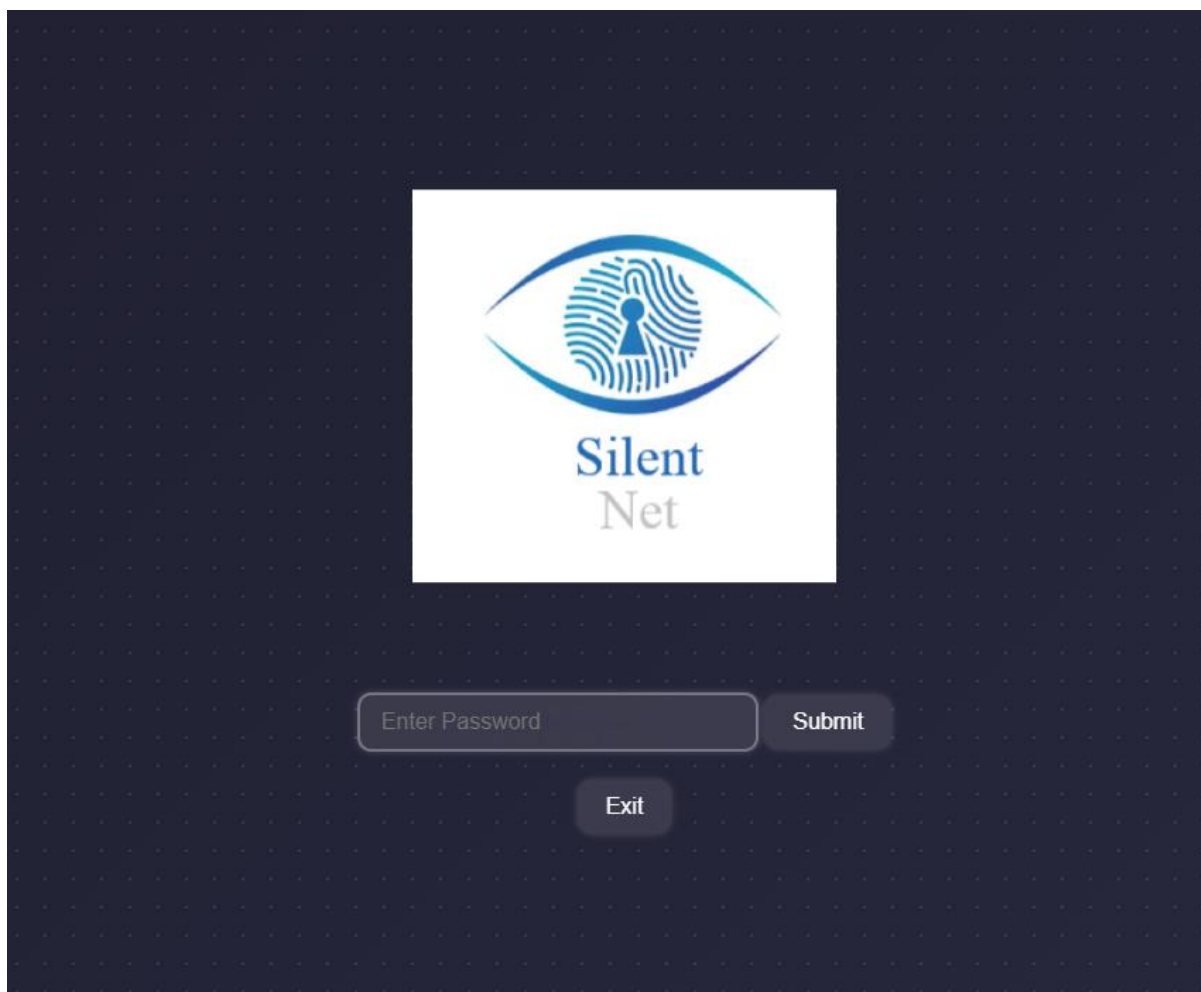
MIH (שרת אל מנהל) - השרת מעדכן את המנהל כי עדכון השם לא חוקי.

MAC (שרת אל מנהל) - השרת מעדכן את המנהל כי קיים כבר מנהל מחובר לשרת ועל כך לא ייתן לעוד מנהל להתחבר.

MNF (שרת אל מנהל) – השרת מעדכן את המנהל כי השם של הלקוח אותו המנהל מבקש לקבל עליו סטטיסטיקות לא נמצא במערכת.

תיאור מסכי המערכת

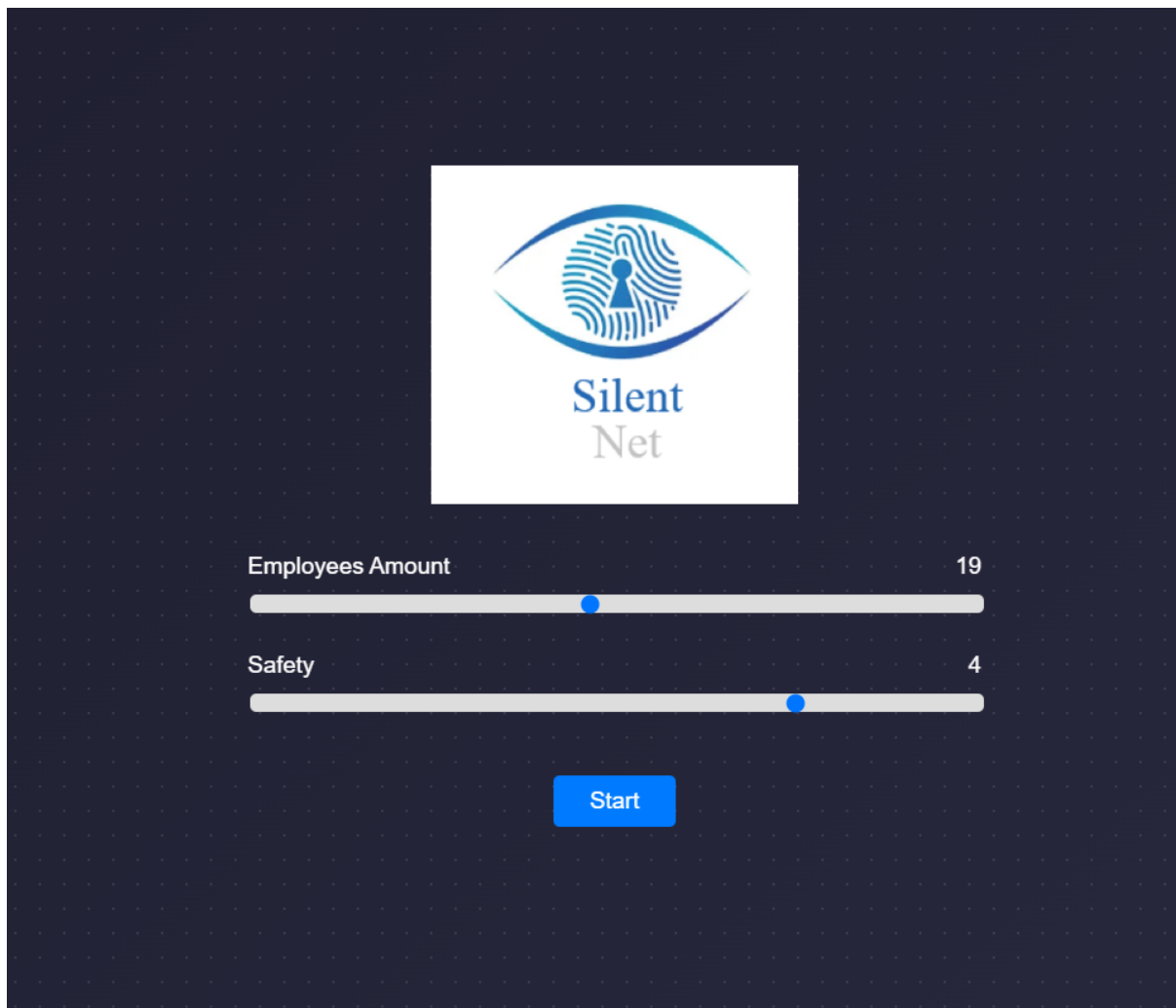
מסך פתיחה



מהות מסך:

מסך זה הינו המסך הראשון ביותר שיופיע במחשב המנהל כאשר הוא פותח את התוכנית. מסך זה הינו מסך פתיחה, כלומר מחכה לקלט המשתמש על מנת להמשיך הלאה. כפי שאפשר לראות במסך זהו בעצם הלוגו של המערכת "Silent Net". במסך זה על המנהל להכניס סיסמה שידועה לו מראש, במידה והסיסמה נכונה ימשיך למסך הבא. במידה והסיסמה לא נכונה יופיע לו הודעה על כך ולא ימשיך למסך הבא. (במידה וכבר מחובר מנהל אחר, המנהל יופנה למסך הטעינה).

מסך הגדרות



Employees Amount 19

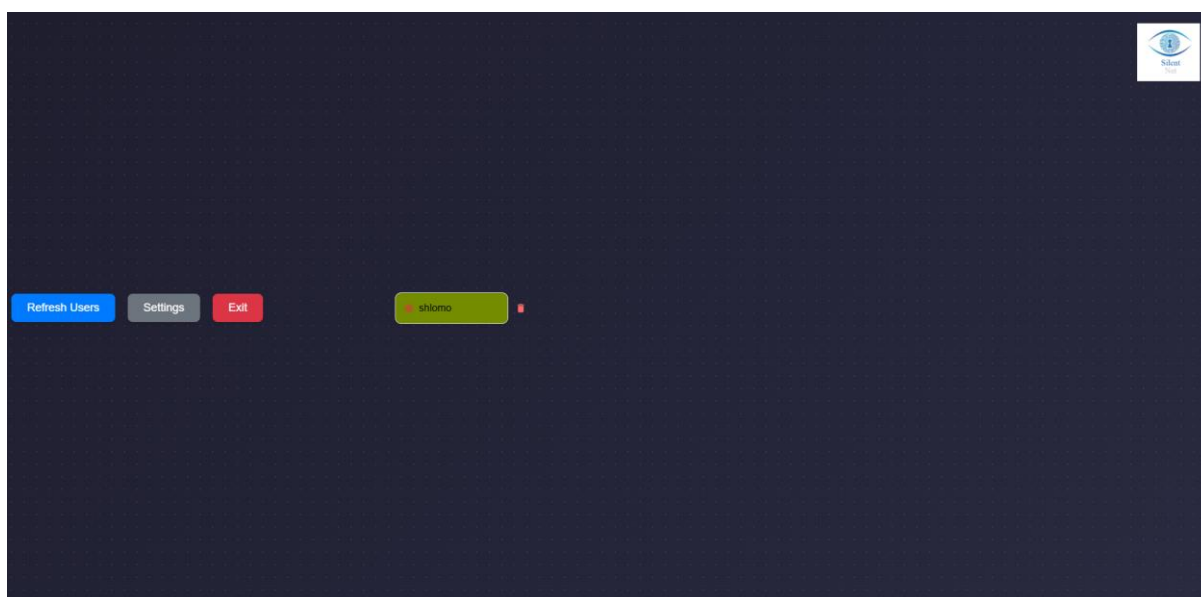
Safety 4

Start

מהות מסך:

מסך זה הינו המסך השני אשר מוצג במחשב המנהל, מסך זה מבקש מן המנהל עצמו להכניס באופן ידני את ההגדרות הראשוניות ביותר לתוך הפרויקט. אחת מן ההגדרות (הראשונה) מבקשת את הכמות המקסימלית של לקוחות שיכולים להתחבר למחשב השרת. הגדרה זאת מונעת מן מחשב השרת לקרוס בעת עומס, בנוסף על כך מונעת איום שידוע בשם DOS/DDOS בו מתחברים כמויות גדולות של לקוחות אל מחשב השרת במטרה להפיל אותו בשל עומס. ההגדרה השנייה קובעת את כמות הבטיחות של הפרויקט, כלומר הכוונה היותר מדויקת היא כמה מחשב השרת משאיר קשר עם מחשב מסוים, עד שהוא מנתק אותו בשל היות אותו קשר עם מחשב שמטרותיו זדוניות. רמת הבטיחות היא בסופו של דבר כמה מחשב השרת משאיר קשר עם מחשב לקוח לאחר שמחשב הלקוח שוב ושוב שלח מידע שלא ניתן לפיענוח ולא עומד בפרוטוקול. אם מחשב לקוח שולח באופן עקבי מידע לא אמין, רמת הבטיחות תקבע מתי כבר להפסיק להאמין לאותו לקוח ולנתק את הקשר איתו.

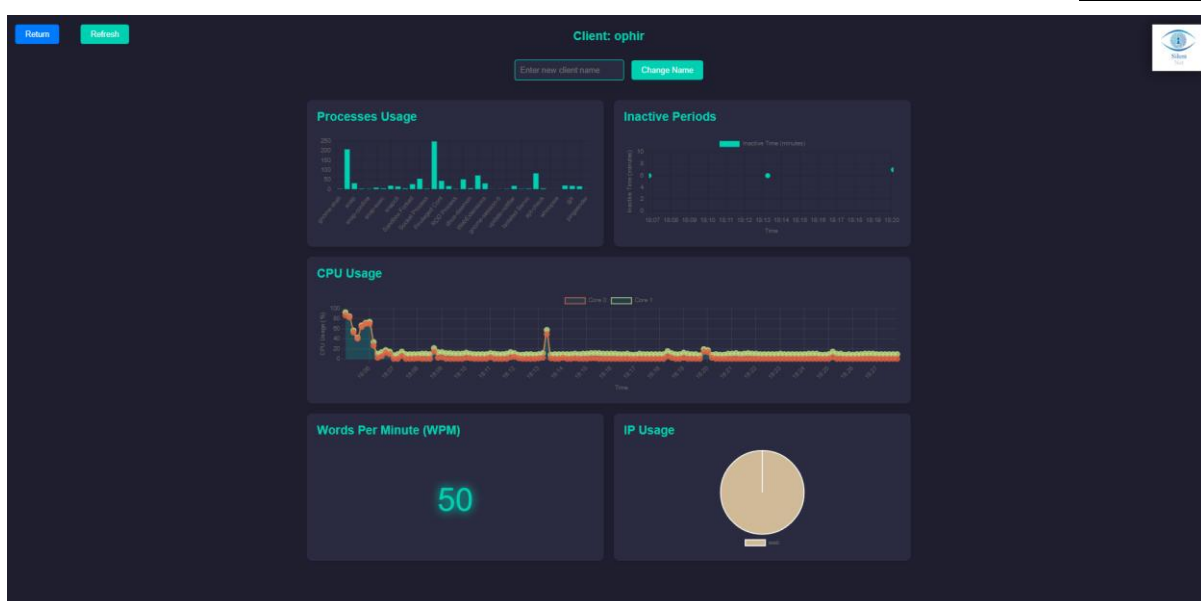
מסך ראשי



מהות מסך:

מסך זה הינו המסך השלישי אשר מוצג במחשב המנהל, מסך זה מראה בזמן אמת את המחשבים המחוברים כרגע למחשב השרת ואשר מנהלים איתו קישור. אותם מחשבים שולחים מידע אל מחשב השרת. שמות המחשבים בדוגמה זו הינם שמות של משתמשים אמיתיים, אך בעת חיבור ראשוני שם המחשב יהיה כשם ה hostname (אשר אין שם אחר שייצג את מחשב זה והוא קריא).

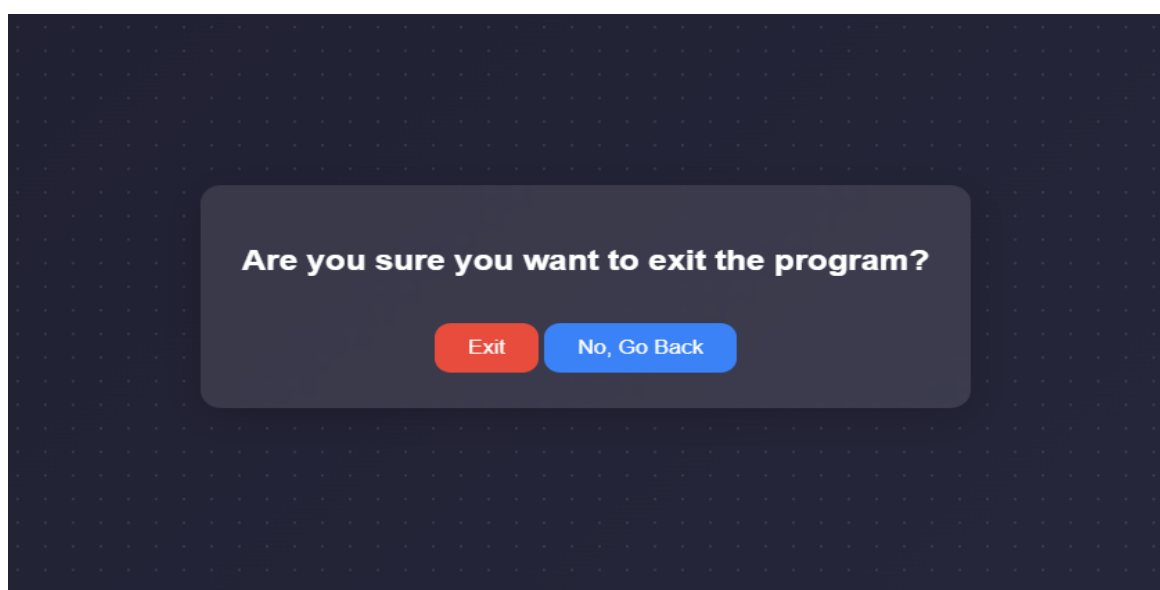
מסך אישי



מהות מסך:

מסך זה הינו המסך הרביעי אשר מוצג במחשב המנהל. מסך זה מראה בזמן אמת את המידע אשר מתקבל מן אותו מחשב בעל כתובת מסוימת. ניתן להגיע למסך זה על ידי לחיצה על אחד השמות הקיימים במסך הראשי. ניתן לראות במסך את הנתונים השונים כמו תהליכים שנפתחו במחשב הלקוח, שימוש בליבות השונות, כמות מילים ממוצעת לדקה, זמנים בהם הלקוח לא היה פעיל, וסוגי IP אליהם פנה הלקוח.

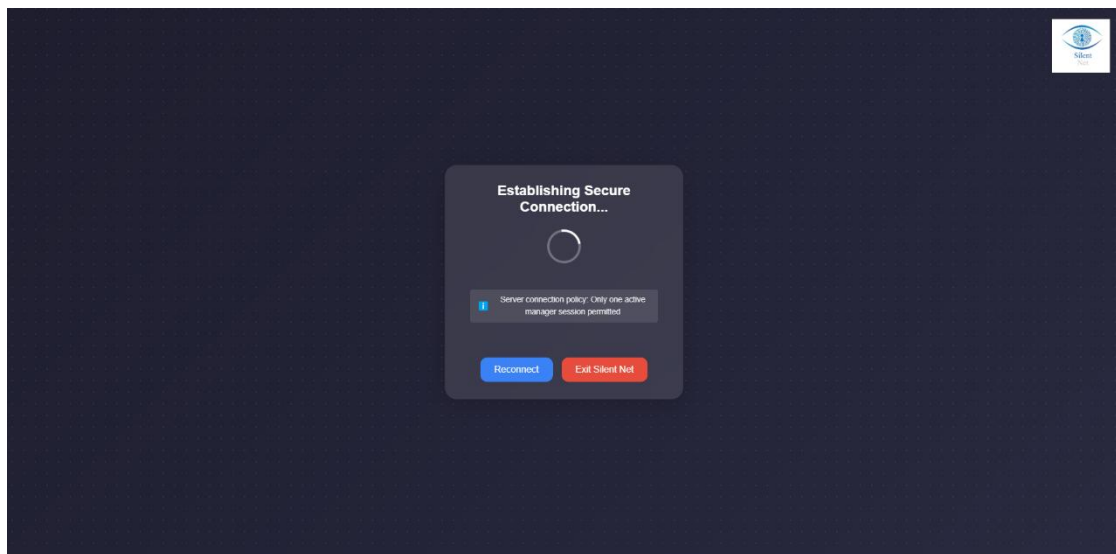
מסך יציאה



מהות מסך

מסך יציאה, זהו המסך שמופיע למנהל לאחר שהמנהל רוצה לסגור את התוכנית. המסך בא לוודא שהמנהל לא רצה לצאת בטעות מן המערכת, ולכן שואל אותו שוב אם ברצונו לסגור ולצאת מן המערכת.

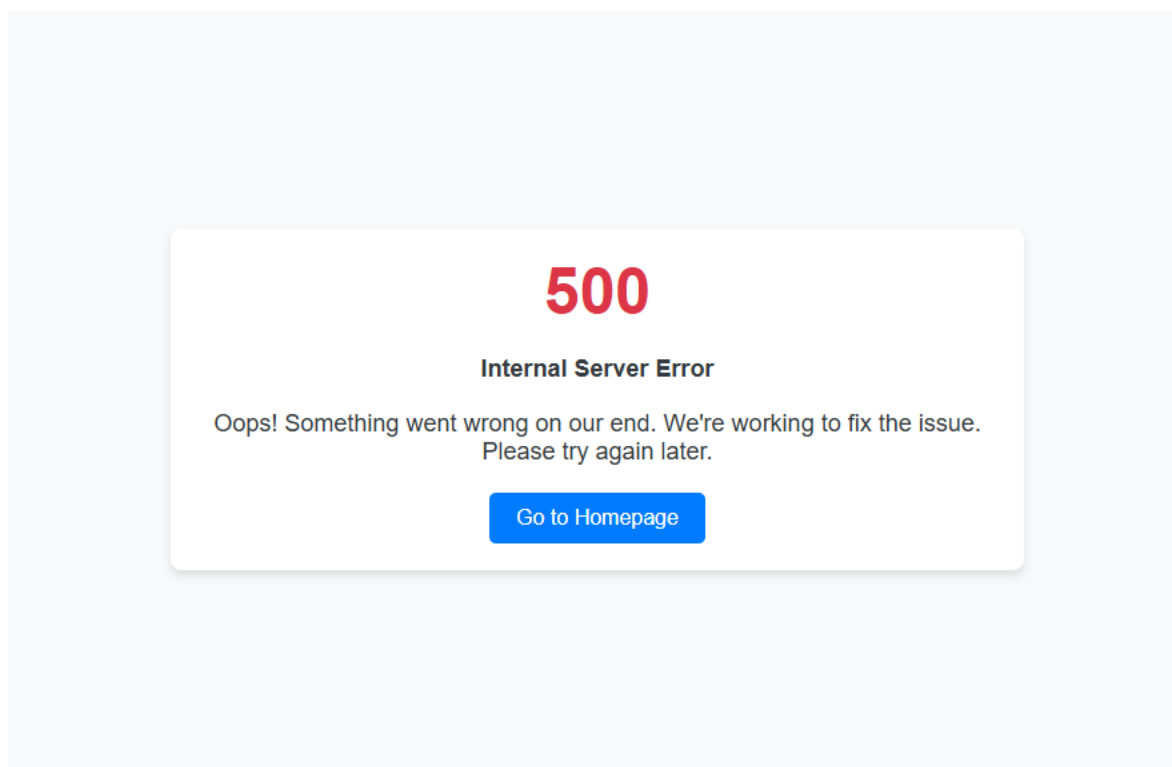
מסך טעינה



מהות מסך

מסך זה מופיע כאשר אבד החיבור עם השרת, ולכן המנהל מופנה למסך זה בו הוא יכול לנסות להתחבר אל השרת כאשר ירצה, אם החיבור יצליח המנהל יופנה למסך ההגדרות, אם לא, יישאר במסך זה.

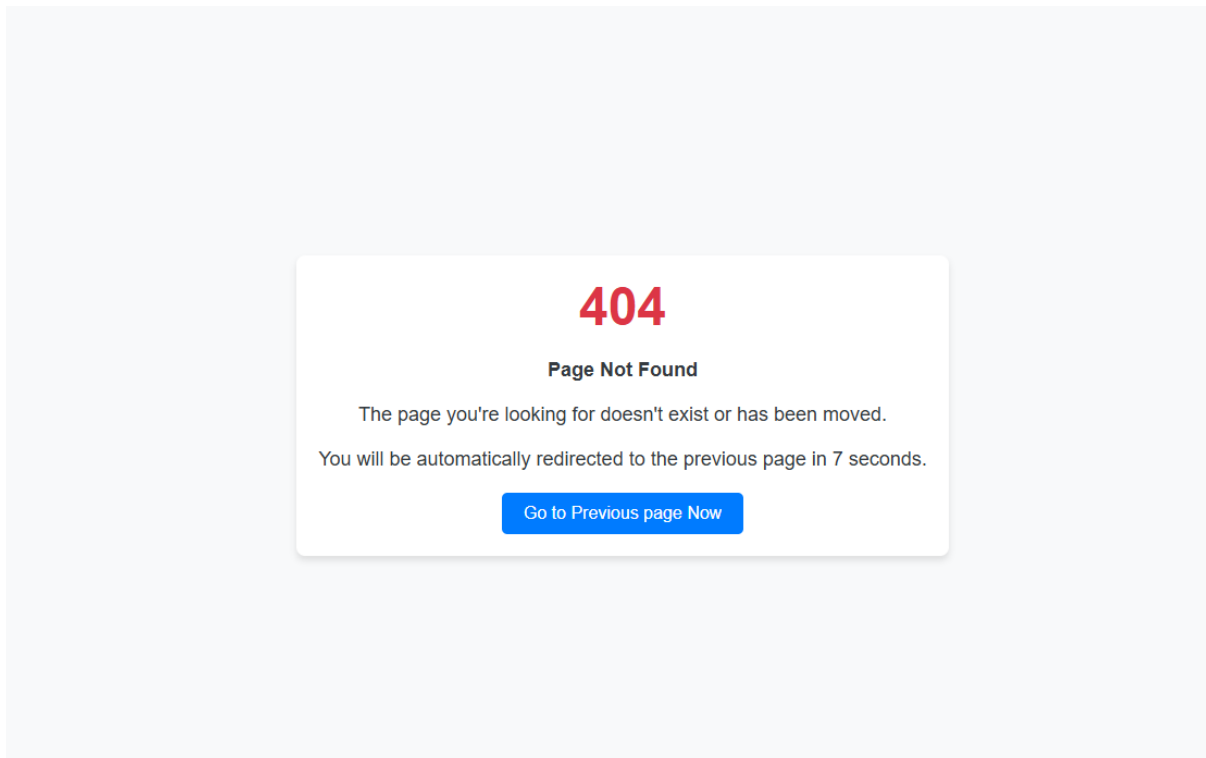
מסך שגיאה



מהות מסך

מסך שגיאה, מופיע כאשר מתרחשת שגיאה אצל הקוד של התוכנית של המנהל. המסך מוביל למסך הבית שהוא מסך הטעינה.

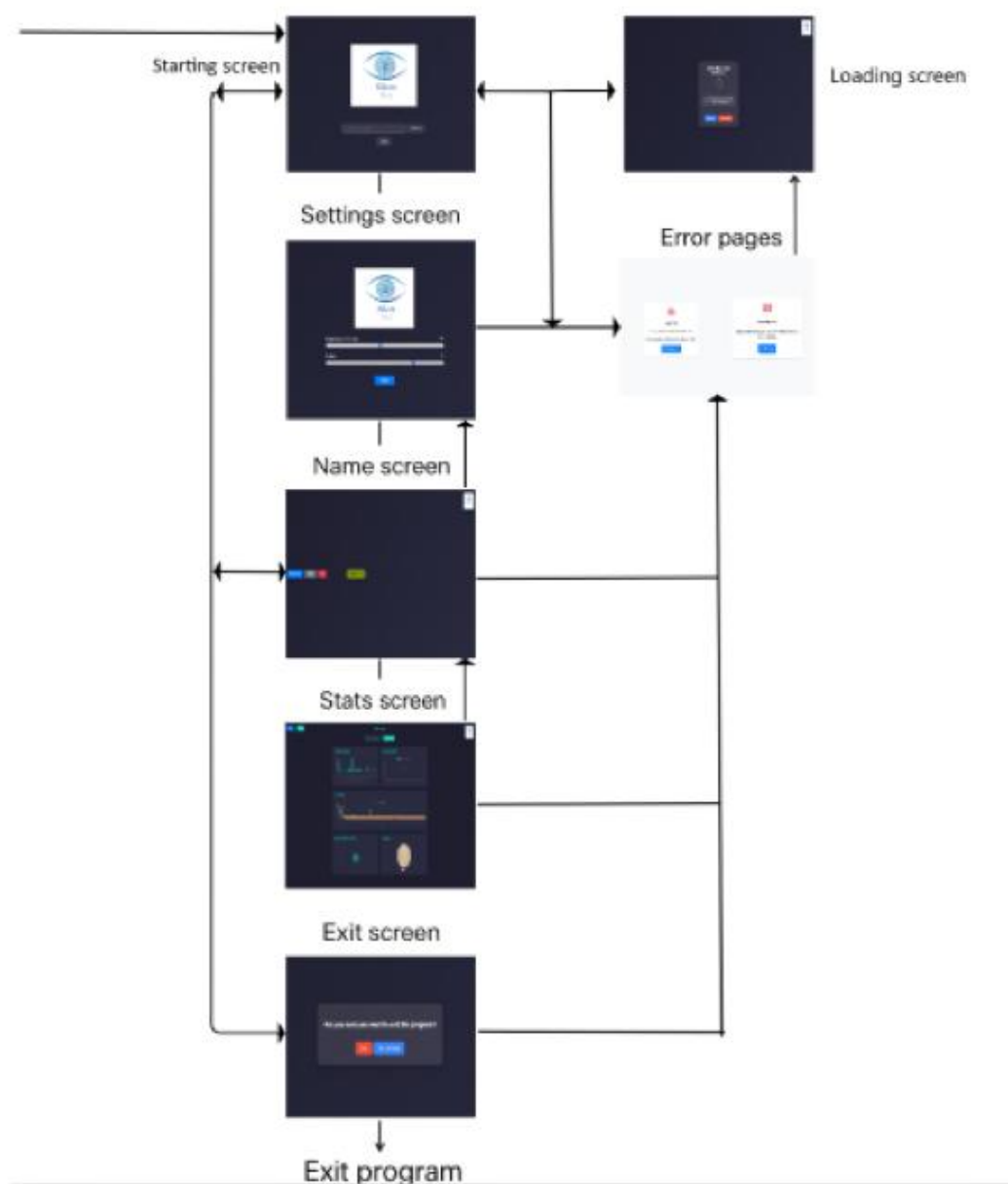
מסך 404



מהות מסך

כאשר המנהל ינסה לשנות את ה URL לכתובת לא חוקית (בתוך התוכנית), יופנה אל המנהל העמוד הזה שמוביל גם כן הוא אל מסך הבית – מסך הטעינה.

דיאגרמת מסכים



תיאור מבני נתונים

פירוט מבני נתונים

הפרויקט עוסק בשלל מבני נתונים שונים, המרכזיים ביניהם בהם עושה שימוש הם שתי טבלאות מרכזיות כמסד נתונים (בצד השרת) ובקובץ מרכזי (בצד הלקוח).

פירוט מאגרי המידע של המערכת

צד שרת – מסד נתונים server db

טבלה ראשונה - logs

המידע שנשלח מהמחשב יכול לכלול נתונים שונים, קלט מחומרה, שמות תהליכים, ערכי CPU ועוד... כל מחשב שולח את המידע האחרון שנשלח, והמערכת בצד השרת מעדכנת באופן רציף בטבלה הראשונה. במערכת שלי, כל id ניתן למשתמש, ה id משמשת כמפתח והערכים ששמורים על אותו id מייצגים את כל המידע שנשמר על אותו משתמש. המידע יכול להיות טקסטואלי (כמו שם תהליך), מספרי (כמו קוד של קלט מחומרה), או בייטים (כמו תו מפריד). כל עדכון מידע ממחשב ספציפי מעדכן את המספר של כמות הפעמים של אותו מידע. הגדרת הטבלה -

| | | |
|-------|---------|------------------------------------|
| id | INTEGER | "id" INTEGER NOT NULL |
| type | TEXT | "type" TEXT NOT NULL |
| data | BLOB | "data" BLOB NOT NULL |
| count | NUMERIC | "count" NUMERIC NOT NULL DEFAULT 1 |

השדות לפי התמונה: id הינו שדה מסוג מספר וסוג ההודעה type הוא שדה מסוג טקסט, שדה המידע עצמו הוא שדה המוגדר כ BLOB – מידע בינארי. השדה האחרון הינו שדה שמונה את כמות הפעמים של ההודעה הנוכחית, וכמובן יהיה שדה מסוג מספר.





כלומר בטבלה זו נשמרים הנתונים שהשרת צריך בכדי לקבל תמונת מצב מלאה על הלקוחות ולבנות מהם את הנתונים שהמנהל בסופו של דבר יראה על מסכו כאשר יבקש מן השרת. מספור ההודעות (שדה אחרון) מייעל לשרת כמות גדולה של זיכרון, במקום לתעד כל log בנפרד, נמספר את כמות הפעמים שהודעה הגיעה מן הלקוח. בכך במקום שמירת כמויות עצומות של מידע ניתן לשמור על כל לקוח פחות ממאה לוגים שונים ועדיין לקבל עליו תמונת מצב מלאה. יש לשים לב כי אין בטבלה זו שדה שמוגדר כ Primary Key, כלומר אין מפתח ראשי בטבלה זו. זאת מכיוון שגם id אשר הוא השדה שמפריד בין לקוחות שונים, יכול להופיע יותר מכמה פעמים, ועל כך לא ניתן להגדיר אותו כמפתח.

דוגמה למידע בטבלה –

| id | type | data | count |
|--------|--------|----------------------------------|--------|
| Filter | Filter | Filter | Filter |
| 4 | CFE | 2025-05-23 21:56:18 | 1 |
| 4 | CLE | 2025-05-23 21:58:12 | 1 |
| 4 | CIN | | 1 |
| 4 | CCU | 0,46□1,3,2025-05-23 21:56:36 ... | 1 |
| 4 | CIE | 8 | 14 |

טבלה שנייה – uid

כל תקשורת בין לקוח לשרת מתחילה כאשר הלקוח שולח לשרת את שם השם של המחשב/משתמש וכתובת ה MAC של כרטיס רשת. מידע זה עוזר לשרת להבדיל בין הלקוחות השונים שמחוברים אליו. מידע זה נכתב בטבלה נפרדת בה נשמרים שני הנתונים הללו, השם המקורי בעת התחברות ו id - שמוצר על ידי השרת בעת הכנסת המשתמש למערכת. המידע נשמר בטבלה נפרדת בשביל הפרדה לוגית ובשל שוני ערכי הנתונים שצריך לשמור. הצורך בשמירת שם המחשב הוא בכדי שכאשר המנהל מבקש את רשימת המחשבים המחוברים הוא יוכל להסתכל עליהם לפי שמות ואף גם לשנות אותם לשמות אחרים, דבר שיקל עליו מאשר להסתכל על כל מחשב ככתובת ה MAC שלו. מכיוון שהמנהל יכול לשנות את שמות המשתמשים, יש צורך לשמור את השם המקורי איתו התחברו לראשונה, בכדי שאם ינסו להתחבר מאוחר יותר המערכת תדע לזהות אותם (אשר הם ינסו להתחבר עם השם המקורי שלהם). בעת התחברות לקוח למערכת, המערכת בודקת שאין את אותו שם כבר בתוך המערכת, במידה ויש היא תוסיף מספרים לאחר השם בכדי ליצור ייחודיות בין השמות (בכדי שהמנהל ידע להפריד בין המשתמשים השונים). בנוסף על כן, בכדי לקבוע אם אותו משתמש מתחבר שוב למערכת, המערכת בודקת אם כבר יש את אותה כתובת ה MAC ואותו שם שמורים בטבלה באותו id. זאת בכדי לאפשר למשתמשים שונים מאותו מחשב להישמר בצורה שונה, מבלי שיתערבבו בשרת. כלומר אם יש שני שמות שונים שמתחברים מאותה כתובת ה MAC המערכת תפריד ביניהם ואכן תהיה הפרדה בדבר אצל מחשב השרת. הגדרת הטבלה -

| | | |
|---|---------|---------------------------------|
|  id | INTEGER | "id" INTEGER |
|  mac | TEXT | "mac" TEXT NOT NULL |
|  hostname | TEXT | "hostname" TEXT NOT NULL UNIQUE |
|  original_hostname | TEXT | "original_hostname" TEXT |

השדות לפי התמונה: מספר id שניתן ללקוח (ניתן לראות כי הוא מוגדר כ primary key זאת בכדי שלא יהיה ערבוב בין ה id's השונים מכיוון שמשתמשים ב id בטבלה הקודמת בכדי להפריד בין הלקוחות), טקסט בשם ה MAC שמייצג את הכתובת של כרטיס הרשת של המחשב, טקסט בשם hostname שמייצג את השם של הלקוח, בנוסף על כך יש את השדה original_hostname שגם הוא טקסט שמייצג את השם המקורי איתו הלקוח התחבר (במידה והמנהל החליט לשנות את השם שלו לשם אחר, בכדי לזהות את אותו לקוח כאשר יתחבר בעתיד, נשמור גם את השם המקורי שלו).

דוגמה למידע בטבלה –

| id | mac | hostname | original_hostname |
|--------|-------------------|------------------|-------------------|
| Filter | Filter | Filter | Filter |
| 4 | 08:00:27:f3:5a:f9 | itzik-VirtualBox | itzik-VirtualBox |

צד לקוח – קובץ לגיבוי מידע

בצד הלקוח, מיושם מנגנון גיבוי מקומי המיועד לשמירה זמנית של נתונים אשר לא עלה בידם להישלח אל השרת הייעודי עקב תקלות שרת או הפרעות בתקשורת הרשת. מנגנון זה ממומש באמצעות קובץ גיבוי בעל מבנה מעגלי (Ring Buffer) וגודל אחסון מוגדרת מראש.

בעת אירוע הדורש גיבוי, הלקוח מבצע העתקה מדויקת של המידע המיועד לשליחה אל קובץ זה, תוך שמירה על פורמט הנתונים המקורי כפי שהוא מוגדר בפרוטוקול התקשורת עם השרת. עם חידוש התקשורת התקינה עם השרת, הלקוח יוזם תהליך של שליחת הנתונים השמורים בקובץ הגיבוי אל השרת. מאחר וגודלו של קובץ הגיבוי הינו קבוע ומוגבל, מנגנון הדריסה המחזורית הינו חלק בלתי נפרד מפעולתו.

עם התמלאות הקובץ, נתונים חדשים יכתבו על גבי הנתונים הוותיקים ביותר, ובכך יאפשרו המשך פעולה רציפה של מנגנון הגיבוי גם תחת עומס נתונים מתמשך. השימוש בקובץ גיבוי בעל גודל קבוע הינו דרישה פונדמנטלית של הפרויקט. החלטה זו נובעת מן הצורך לצמצם את ההשפעה על משאבי המחשב של הלקוח, ובפרט זיכרון ומשאבי דיסק.

באמצעות הגבלת גודל הקובץ, מתאפשרת פעולה יציבה וצפויה של מנגנון הגיבוי ברקע, באופן שקוף למשתמש הקצה וללא פגיעה בביצועי המערכת הכוללים. חשוב להדגיש כי המידע המאוחסן בקובץ הגיבוי משקף באופן ישיר את מבנה הנתונים המוגדר בפרוטוקול התקשורת בין הלקוח לשרת.

הקובץ אינו מבצע כל שינוי או עיבוד של המידע, ותפקידו היחיד הוא שימור זמני של נתונים טקסטואליים או בינאריים המיועדים לשליחה עתידית. ארכיטקטורה זו מבטיחה נאמנות לנתונים המקוריים ומפשטת את תהליך השחזור והשליחה מחדש בעת חידוש הקישוריות עם השרת. ועל כן, המידע בתוך הקובץ יהיה הודעות של הפרוטוקול אחת לאחר השנייה. סוג המידע הינו בתיים (האומנם בשפות low level אין ממש הבדל בין המונחים סטרינג ובתיים, אשר סטרינג הוא אוסף של בתיים. דבר השונה משפות גבוהות כמו python בהם יש התייחסות שונה לסוגים אלו. אפילו שההגדרה של סטרינג היא אוסף של בתיים וכך גם המחשב מתייחס אליהם, python מבצע הפרדה בין המונחים). דוגמה למידע בקובץ-

0010CPO\x1fchrome0006CIE\x1f48

סקירת חולשות ואיומים

בפרויקט זה העוסק במגוון נושאים מעולם הסייבר, אחד הנושאים שצריך לקחת בחשבון הוא ייצור קוד בטוח אשר יודע להתגונן נגד איומים שונים בכדי להבטיח ריצה חלקה ומלאה של הפרויקט ללא הפרעות ומבלי חשש לחדירה לנתונים פרטיים. הפרויקט נמנע מאיומים שונים ומתגונן בפני חולשות בנושאים השונים ובדרכים הבאות –

שכבת האפליקציה

Web

בפרויקט קיימת עבודה עם Web לצורך הצגה נוחה של GUI לצד המנהל. בכדי להימנע מאיומים שונים הפרויקט נמנע מלתת למנהל (אשר הוא הרכיב היחיד בפרויקט בעל GUI) גישה להכנסת קלט כצורת טקסט (בכדי להימנע מאיומים דומים ל XSS – פעולה ידועה שמנצלת את העובדה שאתרים לא בודקים או מנקים נכון קלט של משתמש, ומאפשרים להכניס קוד ישירות לדף).

בנוסף, איום ידוע נוסף הוא כאשר האתר אינו בודק כראוי ואוכף את הגישה לקבצים שונים, כלומר המשתמש יכול לפי ה URL לשנות אותו ולהגיע לקבצים שלא היו בכוונת כותב השרת ציבוריים. הפרויקט נמנע מאיום זה על ידי שהוא נותן גישה אך ורק ל URLים ספציפיים, ואם השם המבוקש לא נמצא בהם שגיאה 404 תופיע למנהל.

אומנם הפרויקט מנסה להימנע מלתת גישה לקלט טקסט מן המשתמש ב GUI אך כן יש מקרים בודדים בהם קיים שימוש והם כאשר המנהל מתבקש להכניס סיסמה בכדי להיכנס למערכת וכאשר המנהל רוצה להחליף שם לאחד המחשבים המחוברים. הסכנה פה היא פעולת sql injection - טכניקת שבה תוקף מזריק פקודות SQL זדוניות לשדה קלט במטרה לשנות את שאלת המסד נתונים ולבצע פעולות לא מורשות.

כאשר מכניסים סיסמה אין אפשרות לבצע איום אשר בצד השרת בשלב זה עוד לא ניגשים אל מסד הנתונים. אך כאשר מחליפים שם ניתן לבצע פעולה ולפגוע במסד הנתונים, אך הפרויקט נמנע מאיום זה על ידי בדיקת תווים מיוחדים בשמות שניתנו, אם המנהל נתן שם עם תווים אלו שיכולים לבצע פגיעה, הפרויקט לא ישלח את השם אל השרת.

תהליך login – צד לקוח וצד שרת

צד שרת:

כחלק מהליך ה login המנהל צריך לשלוח סיסמה שרק היא ידועה למנהלים, כלומר לא כל אדם יכול להשיג את הנתונים שהשרת מספק למנהלים, אלא רק מי שידע את הסיסמה איתה השרת התחיל.

צד לקוח:

כחלק מהליך ה login הלקוח צריך לשלוח את הנתונים שיעזרו לשרת לזהות את לקוח זה – כתובת ה MAC של כרטיס הרשת ושם של המשתמש של המחשב.

שני תהליכי login אלו קריטיים אשר מנהל/לקוח שלא יבצע אותם כמו שצריך לא ימשיך ב session עם השרת, כלומר לא ניתן לשלוח הודעות לשרת שמדמות הודעות שנשלחות באמצע session ולהשיג מידע/לשלוח מידע (שישמר בשרת) מבלי לבצע את התהליך ההתחלתי. הדבר מונע התחזות לאנשים.

הצפנה

כמו שצוין בחלק הקודם, המנהל שולח סיסמה אל השרת בתחילת ה session כחלק מתהליך ה login שלו. כמצופה, לא ניתן לשלוח סיסמה מבלי להצפין אותה אחרת האזנה פשוטה על ידי פעולת MITM (Man in the middle – מחשב שמאזין למידע הנשלח בין שני מחשבים אחרים מבלי ידיעתם) יכולה לגרום להפצת הסיסמה ואנשים שהם לא המנהל יוכלו להשיג את המידע ולמחוק אותו!

בפרויקט נעשה שימוש בהצפנה שמתנהלת בין המנהל לשרת (עוד לפני שליחת הסיסמה). ההצפנה מונעת מכל מחשב שמאזין להבין את המידע הנשלח בין מחשב המנהל למחשב השרת ולא יהיה לו תועלת מכך. סוגי ההצפנות ופרוטוקולים שהפרויקט משתמש בהם הם:

DH (Diffie Hellman) – פרוטוקול להחלפת מפתחות ידוע שמסתמך על קושי פתרון בעיית הדיסקרט לוגריתם (Discrete Logarithm Problem), הפרוטוקול מאפשר לשני אנשים להגיע למפתח פרטי על ערוץ ציבורי (כך שאם מישהו מאזין לתקשורת הוא אינו יכול להגיע למפתח הפרטי). הפרויקט משתמש במספרים גדולים וחזקים שיקחו זמן רב בשביל מחשב מאזין לפענח בכדי להגיע למפתח הפרטי.

לאחר ששני הצדדים בעלי המפתח הפרטי ניתן כעת להשתמש בהצפנה המידע עם אותו מפתח פרטי שמשותף רק לשניהם. בשלב הזה הפרויקט משתמש בהצפנה סימטרית AES (מצב CBC) – אלגוריתם להצפנה סימטרית שפועל במצב שרשור בלוקים (Cipher Block Chaining) כדי לשפר את האבטחה על ידי קישור כל בלוק מוצפן לבלוק הקודם, הצפנה חזקה שמקשה על מי שינסה להאזין לתעבורה.

חשוב לציין כי יש חשיבות גדולה לשימוש ב TCP עם מצב CBC בהצפנה AES, אשר מכיוון שההצפנה הינה הצפנת בלוקים, כל בלוק מסתמך על הבלוק הקודם כחלק מההצפנה, ולכן גם הקידוד יעבוד באותה צורה, כלומר אם בלוק אחד הולך לאיבוד או אפילו רק חלק ממנו, דבר שמאוד יתכן בפרוטוקולים כמו UDP, לא יהיה ניתן לפענח את המסרים מהצד השני.

שכבת התעבורה – פרוטוקול TCP

TCP מספק חיבור אמין בין שני צדדים באמצעות מנגנון של לחיצת יד משולשת (Three-Way Handshake). במהלך לחיצת היד, שני הצדדים מאמתים אחד את השני ומסכימים על פרמטרים בסיסיים של החיבור. התהליך מבטיח שהחיבור נוצר בכוונה תחילה ולא נוצר בטעות או בפעולת זיוף פשוטה (כגון IP spoofing). בנוסף, TCP כולל מנגנוני בקרת שגיאות - לכל מקטע (segment) נוסף Checksum — מספר שמייצג את תוכן המקטע.

כשמקבלים מקטע, בודקים אם החישוב מתאים. אם יש טעות, המקטע נזרק והצד השולח ישלח אותו שוב, סידור מנות - לפעמים מנות מגיעות בסדר שונה מזה שבו הן נשלחו. TCP נותן לכל מקטע מספר סידורי (Sequence Number), וכשהצד המקבל מקבל מנות, הוא מסדר אותן לפי המספרים כדי לשחזר את הנתונים בדיוק כמו שנשלחו, ובקרת זרימה - TCP מוודא שהשולח לא יציף את המקבל ביותר מדי נתונים.

לשם כך המקבל מודיע לשולח כמה מקום פנוי יש לו בזיכרון (בחלון שנקרא Window Size). כך, השולח מתאים את קצב השליחה למה שהמקבל מסוגל לקלוט ולעבד, שמגינים על שלמות הנתונים ומקטינים סיכון לפגיעות כתוצאה מהעברת נתונים חלקית או שגויה.

DDOS/DOS

אחד האיומים שקלים לביצוע אך אפקטיבי מבניהם. האיום מסתמך ישירות על כך שמשאבים של מחשב אינם אינסופיים, ולכן אם רכיב אחד מקבל כמות עצומה של משימות בבת אחת הוא ככל הנראה יקרוס/לא יתפקד כמצופה. הפרויקט נמנע מאיום זה בכמה דרכים שונות:

- הגבלת כמות הלקוחות אצל השרת, השרת פשוט לא ייצור session עם לקוחות כאשר כבר קיימים כמות לקוחות מחוברים ככמות שהמנהל הגביל את השרת. בנוסף המנהל יכול גם כן להגביל את השרת לכמות בעצמה מוגבלת של לקוחות (המנהל לא יכול לתת מעל 40 לקוחות, כמות גדולה אך מעליה השרת מתקשה בביצוע פעילותו).
- הגבלת כמות המנהלים שיכולים להיות מחוברים לשרת בו זמנית. בפרויקט רק מנהל אחד יכול לנהל session עם השרת (אפילו שהמנהל מקבל יחס עליון – גם אם כמות הלקוחות הגיעה לכמות הלקוחות המקסימלית המנהל עדיין יכול להתחבר).
- הגדרת רמת בטיחות ללקוחות. המנהל יכול להגדיר לשרת את רמת הבטיחות שהוא רוצה כאשר הוא מחובר לשרת, רמת הבטיחות היא מקושרת ישירות לכמות ההודעות שהשרת לא יכול לפענח שמגיעה מן הלקוחות, לאחר כמות ההודעות האלו שמגיעה ברצף השרת סוגר את session עם הלקוח. הגדרה זו מונעת מלקוח שכן התחבר כראוי לשרת לבצע פעולת DOS על השרת.

Timing attack

טכניקה שבה התוקף מנתח את הזמן שלוקח למערכת לבצע פעולות שונות, כדי לגלות מידע סודי כמו סיסמאות, איום זה מסתמך על ההתנהגות הטבעית של המחשב בביצוע פעולות שונות. אם צד אחד יודע את הדרך בה מתבצעת פעולה מסוימת הוא יכול למדוד זמנים כמה זמן לוקח לכל קלט להחזיר תשובה מהשרת.

הדרך שבה השרת בודק אם הסיסמה נכונה היא על ידי השוואת אות אות, ברגע שאות לא טועמת את האות בסיסמה השמורה, הוא מסיים את הבדיקה ומחזיר FALSE, עם ידע זה, הצד המתקיף יכול לנסות סטרינגים שונים ולראות מה לוקח לשרת יותר זמן להגיב, ולפי זה הוא יכול לדעת אם הוא בכיוון לסיסמה הנכונה לא.

ניתן לבצע פעולה זו אם המנהל בטעות השאיר את המחשב שלו פתוח ועובד השיג גישה למחשב המנהל (בהסתמך שהתוכנית סגורה כאשר המנהל לא על המחשב, אם המנהל לא סוגר את התוכנית כשהוא לא על המחשב ועובדים יכולים להשיג גישה למחשב שלו זה כבר מעבר לתחום הפרויקט), העובד יכול לנסות לשלוח סיסמאות שונות בדף פתיחה אל השרת ולראות מה לוקח יותר זמן.

הפתרון לבעיה זו הוא דווקא פשוט למימוש בצד השרת – לאחר כל פעם שהשרת מאמת סיסמה שנשלחה אליו הוא מחכה למשך זמן רנדומלי (עד לשנייה) ורק לאחר מכן מגיב בחזרה לצד השני. ביצוע פעולה זו גורם לכך שאין באפשרות הצד השני לעולם לדעת אם הסיסמה מקיימת בתוכה חלק מתוך הסיסמה האמיתית ובכך לא יקבל רמזים לגבי הסיסמה האמיתית.

מימוש הפרויקט

סקירת כל המודולים/מחלקות וקשרי הגומלין ביניהם

מודולים/מחלקות מיובאים

מנהל

sys - משמש להוספת תיקיית האב ל- sys.path כדי לאפשר ייבוא של קבצים משותפים כמו protocol, encryption.

webbrowser - פותח את ממשק הניהול בדפדפן ברירת המחדל כאשר האפליקציה מופעלת.

os - משמש להרצת פעולות מערכת כמו קבלת נתיב קובץ נוכחי ושליחת אות סיום לתהליך. (os.kill).

signal - משמש לשליחת אות SIGINT כדי לעצור את התהליך בצורה נקייה בעת יציאה מהמערכת.

Json - משמש לפענוח מידע בפורמט JSON שמתקבל מהשרת, למשל נתוני סטטיסטיקה של עובדים.

functools.wraps - דקורטור שעוזר לשמור על המידע של הפונקציה המקורית כאשר עוטפים אותה בפונקציה אחרת (כמו בדקורטור check_screen_access).

Flask מחלקת – (Flask משמשת ליצירת אפליקציית ווב בפלאסק).

redirect (פונקציה) – מחזירה תגובת הפניה לכתובת אחרת (למשל לאחר התחברות או שגיאה).

render_template (פונקציה) – מציגה דף HTML מתוך תיקיית התבניות (templates) עם אפשרות לשלוח נתונים לדף.

request (אובייקט) – מייצג את הבקשה שנשלחה על ידי המשתמש, כולל מידע כמו פרמטרים, טפסים ו-JSON.

jsonify (פונקציה) – ממירה מבני נתונים של פייתון (כמו dict או list) לתגובה בפורמט JSON.

url_for (פונקציה) – יוצרת כתובת URL לפונקציה לפי שמה, בצורה בטוחה ודינמית.

שרת

sys – משמש לקבלת פרמטרים מהשורה (כמו מספר לקוחות וסיסמה) ולהוספת תיקיות ל sys.path.

threading – משמש לניהול חיבורים מרובים בו-זמנית באמצעות יצירת תהליכונים (threads) לכל לקוח.

os – משמש לגישה למערכת הקבצים, קבלת נתיבים ל-DB, ופעולות מערכת כלליות.

json – משמש להמרה של מידע ל-JSON כדי לשלוח אותו למנהל דרך הפרוטוקול.

time.sleep – משמש להשהיית פעולה לזמן רנדומלי כדי למנוע פעולות תזמון (timing attacks).

random.uniform – מגריל מספרים עשרוניים לטווח מסוים, משמש גם כן למניעת פעולות תזמון.

keyboard.on_press_key – מקשיב ללחיצות מקשים (q, e) להפסקת השרת או למחיקת הלוגים.

socket.timeout – מטפל במצבים שבהם מתבצעת קריאה לחיבור שחרג מהזמן שהוקצה לו.

traceback – משמש להדפסת מידע מפורט על שגיאות שהתגלו במהלך ריצת התוכנית לצורכי ניפוי שגיאות (debugging).

לקוח

- types – מספק טיפוסים נתונים כמו uint32_t ו-int64_t לניהול נתונים יעיל בליבה.
- uaccess – מאפשר גישה מאובטחת לזיכרון משתמש להעתקת נתונים בין מרחבי כתובות.
- fs – מספק פונקציות לניהול קבצים במערכת, כולל קריאה, כתיבה וסגירה.
- init – מאפשר הגדרת פונקציות אתחול (init__) וניקוי (exit__) עבור המודול.
- input – מספק מבנים ופונקציות לטיפול באירועי קלט ממקלדת ועכבר.
- kernel – מכיל פונקציות ליבה בסיסיות כמו printk לניפוי שגיאות.
- kernel_stat – מספק גישה לסטטיסטיקות מעבד, כגון זמני פעילות ומנוחה.
- kprobes – מאפשר הוספת נקודות תצפית (hooks) לפונקציות ליבה.
- module – הכרחי להגדרת מודול ליבה, כולל רישוי, מחבר ותיאור.
- mutex – מספק נעילות לסנכרון תהליכים במצבים מרובי-תהליכונים.
- netdevice – מאפשר גישה לרשימת התקני רשת וכתובות MAC
- sched – מספק גישה למבנה current לזיהוי תהליכים ומשתמשים.
- slab – מספק פונקציות להקצאת ושחרור זיכרון בליבה.
- utsname – מאפשר קבלת שם המחשב (hostname) לזיהוי המערכת.
- workqueue – מאפשר ניהול משימות רקע (background tasks) בצורה אסינכרונית.
- fttrace – משמש להסתרת חיבורי רשת על ידי שינוי התנהגות פונקציות ליבה.
- in – מכיל פונקציות כמו in_aton להמרת כתובות IP
- inet – מספק פונקציות רשת כמו htons להמרת פורטים.
- net – מכיל הגדרות בסיסיות לחיבורי רשת ומבני נתונים כמו struct sock
- ptrace – מאפשר גישה לרגיסטרים במהלך הוקים של פונקציות ליבה.
- seq_file – משמש להסתרת חיבורי רשת בעת הצגתם בפקודות כמו netstat
- tcp – מספק פונקציות ומידע על חיבורי TCP לניהול תקשורת.
- dcache – מאפשר גישה למבני נתונים של מערכת הקבצים לניהול קבצים נסתרים.
- err – מכיל קודי שגיאה סטנדרטיים לטיפול בתקלות.
- mount – מספק פונקציות לניהול נקודות עגינה (mount points) במערכת הקבצים.
- stat – מכיל הגדרות הרשאות ופונקציות לבדיקת תכונות קבצים.

מודולים/מחלקות מקוריים

manager.py

המודול נעזר ב protocol.py

Class: SilentNetManager

תפקיד: מחלקה זו היא הלב של אפליקציית האינטרנט בצד המנהל. היא משתמשת ב Flask - כדי ליצור את ממשק המשתמש, לטפל בניתוב ולתקשר עם השרת. היא מנהלת את המסכים השונים של האפליקציה (כניסה, הגדרות וכו') ואת הזרימה ביניהם.

תכונות:

- Flask (app) מופע אפליקציית Flask.
- manager_socket (client או None): אובייקט socket לתקשורת עם השרת.
- is_connected (bool) דגל המציין אם המנהל מחובר לשרת.
- screens (dict) מילון הממפה נתיבי URL לרמות היררכיית מסך.
- current_screen (str) ה URL של המסך המוצג כעת.
- previous_screen (str) ה URL של המסך שהוצג קודם לכן.

פעולות:

- `__init__:`
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: מאתחל את אפליקציית הניהול, מגדיר את אפליקציית Flask, הסוקט, היררכיית המסכים והנתיבים.
- `setup_routes:`
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: מגדיר את כל הנתיבים של Flask עבור אפליקציית האינטרנט.
- `setup_error_handlers:`
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: מגדיר מטפלי שגיאות עבור שגיאות HTTP 404 ו-500.
- `check_screen_access:`
 - טענת כניסה: פונקציית תצוגה ניתנת לקריאה של Flask (f).
 - טענת יציאה: פונקציה עטופה ניתנת לקריאה שאוכפת היררכיית מסכים.
 - תיאור: דקורטור שבודק אם למשתמש יש הרשאה לגשת למסך בהתבסס על היררכיה מוגדרת.

- disconnect:
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: מתנתק מהשרת ומנקה את הסוקט.
- exit_program:
 - טענת כניסה: אין
 - טענת יציאה: תגובה ריקה עם קוד סטטוס 204.
 - תיאור: מטפל ביציאה מהאפליקציה, מתנתק מהשרת ומסיים את התהליך.
- page_not_found:
 - טענת כניסה: אובייקט שגיאה (בדרך כלל 404).
 - טענת יציאה: תבנית HTML מעובדת עבור דף השגיאה 404.
 - תיאור: מעבד את דף השגיאה 404.
- internal_error:
 - טענת כניסה: אובייקט שגיאה (בדרך כלל 500).
 - טענת יציאה: תבנית HTML מעובדת עבור דף השגיאה 500.
 - תיאור: מעבד את דף שגיאת השרת הפנימית 500.
- exit_page:
 - טענת כניסה: אין
 - טענת יציאה: תבנית HTML מעובדת עבור מסך אישור היציאה.
 - תיאור: מעבד את דף אישור היציאה.
- loading_screen:
 - טענת כניסה: אין
 - טענת יציאה: תבנית HTML מעובדת עבור מסך הטעינה.
 - תיאור: מעבד את מסך הטעינה בזמן ניסיון להתחבר לשרת.
- start_screen:
 - טענת כניסה: אין
 - טענת יציאה: תבנית HTML מעובדת עבור מסך הכניסה הראשוני, פוטנציאלית עם דגל המציין סיסמה שגויה.
 - תיאור: מעבד את מסך הכניסה הראשוני.
- check_password:
 - טענת כניסה: סיסמה מטופס הכניסה.

- טענת יציאה: תגובת הפניה, או למסך ההגדרות (בכניסה מוצלחת), למסך הטעינה (אם החיבור נכשל), או למסך הפתיחה (אם הסיסמה שגויה).
- תיאור: מאמת את סיסמת המנהל מול השרת.
- settings_screen:
 - טענת כניסה: אין
 - טענת יציאה: תבנית HTML מעובדת עבור מסך הגדרות השרת.
 - תיאור: מעבד את מסך הגדרות השרת.
- submit_settings:
 - טענת כניסה: נתוני טופס המכילים את מספר העובדים והגדרות הבטיחות.
 - טענת יציאה: תגובת הפניה למסך העובדים.
 - תיאור: מעדכן את הגדרות השרת עם הערכים שסופקו.
- employees_screen:
 - טענת כניסה: אין
 - טענת יציאה: תבנית HTML מעובדת עבור מסך רשימת העובדים, המכילה רשימה של סטטיסטיקות עובדים.
 - תיאור: מעבד את מסך רשימת העובדים, ושולף נתונים מהשרת.
- delete_client:
 - טענת כניסה: נתוני JSON המכילים את שם הלקוח למחיקה.
 - טענת יציאה: תגובת JSON המציינת הצלחה או כישלון של המחיקה.
 - תיאור: מטפל בבקשה למחיקת לקוח מהשרת.
- manual_connect:
 - טענת כניסה: אין
 - טענת יציאה: תגובת JSON המציינת את סטטוס החיבור.
 - תיאור: מנסה להתחבר לשרת באופן ידני ומחזיר את סטטוס החיבור.
- update_client_name:
 - טענת כניסה: נתוני JSON המכילים את השם הישן והשם החדש של הלקוח.
 - טענת יציאה: תגובת JSON המציינת הצלחה או כישלון של עדכון השם.
 - תיאור: מטפל בבקשה לעדכון שם של לקוח.
- stats_screen:
 - טענת כניסה: שם של לקוח.
 - טענת יציאה: תבנית HTML מעובדת עבור מסך הסטטיסטיקות המפורטות, המכילה נתוני סטטיסטיקה של הלקוח.

○ תיאור: מעבד את מסך הסטטיסטיקות המפורטות עבור לקוח מסוים, ושולף נתונים מהשרת.

• `connect_to_server`:

○ טענת כניסה: אין

○ טענת יציאה: אין

○ תיאור: מנסה להתחבר לשרת.

• `run`:

○ טענת כניסה: אין

○ טענת יציאה: אין

○ תיאור: מריץ את אפליקציית Flask.

server.py

המודול נעזר ב `process_limit.py` | `DB.py` | `protocol.py`

Class: SilentNetServer

תפקיד: זוהי אפליקציית ליבת השרת. היא מנהלת את כל חיבורי הלקוח והמנהל, מקבלת ומעבדת נתונים, ויוצרת אינטראקציה עם מסד הנתונים. היא אחראית לפעולה הכוללת של מערכת "Silent Net".

תכונות:

- `max_clients (int)` המספר המרבי של לקוחות עובדים המורשים להתחבר.
- `safety (int)` סף בטיחות לטיפול בהודעות לא חוקיות.
- `password (str)` הסיסמה הנדרשת לאימות מנהל.
- `proj_run (bool)` דגל המציין אם השרת פועל.
- `manager_connected (bool)` דגל המציין אם מנהל מחובר כעת.
- `clients_connected (list)` רשימה של טאפלים, שכל אחד מהם מכיל אובייקט ת'רד ואובייקט לקוח עבור לקוחות עובדים מחוברים.
- `macs_connected (list)` רשימה של כתובות MAC של לקוחות עובדים מחוברים.
- `clients_recv_event (threading.Event)` אירוע המשמש לסנכרון קבלת נתוני לקוח.
- `log_data_base (UserLogsORM)` מופע של המחלקה UserLogsORM לניהול לוגים.
- `uid_data_base (UserId)` מופע של המחלקה UserId לניהול זהויות משתמשים.
- `server_comm (server)` אובייקט שרת לתקשורת רשת.

פעולות:

• `__init__`:

○ טענת כניסה: אין

○ טענת יציאה: אין

- תיאור: מאתחל את השרת עם הגדרות ברירת מחדל.
- start:
- טענת כניסה: אין
- טענת יציאה: אין
- תיאור: מפעיל את השרת עם ההגדרות שהוגדרו.
- load_configuration_:
- טענת כניסה: אין או ארגומנטים משורת הפקודה (מספר לקוחות מקסימלי, רמת בטיחות וסיסמה).
- טענת יציאה: אין
- תיאור: טוען את הגדרות השרת משורת הפקודה או משתמש בברירות מחדל.
- initialize_databases_:
- טענת כניסה: אין
- טענת יציאה: אין
- תיאור: מאתחל חיבורים למסדי הנתונים.
- setup_keyboard_shortcuts_:
- טענת כניסה: אין
- טענת יציאה: אין
- תיאור: מגדיר קיצורי מקלדת לשליטה בשרת.
- run_server_:
- טענת כניסה: אין
- טענת יציאה: אין
- תיאור: לולאת השרת הראשית לקבלה וטיפול בחיבורי לקוחות.
- accept_clients_:
- טענת כניסה: אין
- טענת יציאה: אין
- תיאור: מקבל ומנהל חיבורי לקוחות נכנסים.
- handle_client_connection_:
- טענת כניסה: אובייקט לקוח.
- טענת יציאה: אין
- תיאור: קובע את סוג הלקוח ומנתב למטפל המתאים.
- determine_client_type_:
- טענת כניסה: אובייקט לקוח, סוג הודעה, תוכן הודעה.

- טענת יציאה: ערך בוליאני: True אם הלקוח היה מנהל, False אחרת.
- תיאור: קובע אם הלקוח הוא מנהל או עובד ומטפל בהתאם.
- `:handle_manager_connection`
 - טענת כניסה: אובייקט לקוח, הודעת סיסמה.
 - טענת יציאה: ערך בוליאני: תמיד מחזיר True (חיבור מנהל מטופל).
 - תיאור: מטפל באימות וחיבור של מנהל.
- `:handle_employee_connection`
 - טענת כניסה: אובייקט לקוח, הודעת אימות.
 - טענת יציאה: אין
 - תיאור: מטפל באימות וחיבור של עובד.
- `:remove_disconnected_client`
 - טענת כניסה: אובייקט לקוח להסרה.
 - טענת יציאה: אין
 - תיאור: מסיר לקוח מנותק מרשימת הלקוחות המחוברים.
- `:quit_server`
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: מסיים את פעולת השרת.
- `:erase_all_logs`
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: מוחק את כל הלוגים ממסד הנתונים.

Class: ManagerHandler

תפקיד: מחלקה זו, הפועלת בצד השרת, מטפלת בתקשורת ספציפית מלקוח מנהל מחובר. היא מקבלת בקשות מהמנהל (למשל, עבור נתוני לקוח, מחיקת לקוח) ומעבדת אותן, לעתים קרובות באינטראקציה עם מסד הנתונים. משתמשת ב `protocol.py` ו `encryption.py` על מנת לקיים תקשורת מוצפנת ואמינה עם המנהל.

תכונות:

- `server (SilentNetServer)` הפניה למופע השרת הראשי.
- `client (client)` חיבור ה socket-למנהל.

פעולות:

- `:__init__`

- טענת כניסה: מופע של SilentNetServer, אובייקט לקוח.
- טענת יציאה: אין
- תיאור: מאתחל את המטפל במנהל עם השרת ואובייקט הלקוח המתאים.
- process_requests:
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: לולאה ראשית לעיבוד בקשות ממנהל.
- handle_client_request_:
 - טענת כניסה: סוג הודעה, תוכן הודעה.
 - טענת יציאה: תוכן תגובה.
 - תיאור: מנתב בקשות לקוח לפונקציות טיפול מתאימות.
- get_client_data_:
 - טענת כניסה: שם לקוח.
 - טענת יציאה: נתוני סטטיסטיקה של לקוח.
 - תיאור: אוסף נתוני סטטיסטיקה מפורטים עבור לקוח מסוים.
- handle_name_change_:
 - טענת כניסה: פרמטרי הודעה (שם קודם ושם חדש).
 - טענת יציאה: סוג הודעה המציין הצלחה או כישלון.
 - תיאור: מטפל בבקשה לשינוי שם לקוח.
- delete_client_:
 - טענת כניסה: פרמטרי הודעה (שם לקוח).
 - טענת יציאה: אין
 - תיאור: מטפל בבקשה למחיקת לקוח.
- handle_unsafe_message_:
 - טענת כניסה: אין
 - טענת יציאה: ערך בוליאני המציין אם לנתק את המנהל.
 - תיאור: מטפל בהודעות לא בטוחות/לא חוקיות מהמנהל.

Class: ClientHandler

תפקיד: מחלקה זו, הפועלת בצד השרת, מטפלת בתקשורת עם לקוח (עובד). היא מקבלת נתונים מהלקוח (כגון שימוש במעבד, תהליכים פתוחים, הקלדות) ומעבדת אותם, ומאחסנת את המידע הרלוונטי במסד הנתונים.

תכונות:

- server (SilentNetServer) הפניה למופע השרת הראשי.
- client (client) חיבור ה socket-ללקוח.
- mac_address (str) כתובת ה MAC-של הלקוח המשויך למטפל זה.

פעולות:

- `__init__`:
 - טענת כניסה: מופע של SilentNetServer, אובייקט לקוח, כתובת MAC של לקוח.
 - טענת יציאה: אין
 - תיאור: מאתחל את המטפל בלקוח עם השרת, אובייקט הלקוח וכתובת ה-MAC המתאימה.
- `process_data`:
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: לולאה ראשית לעיבוד נתונים מלקוח.
- `handle_client_data`:
 - טענת כניסה: סוג נתונים, נתוני הודעה.
 - טענת יציאה: אין
 - תיאור: מנתב נתוני לקוח לפונקציות טיפול מתאימות.
- `handle_ip_data`:
 - טענת כניסה: נתוני הודעה (כתובות IP).
 - טענת יציאה: אין
 - תיאור: מטפל בנתוני כתובות IP מהלקוח.
- `handle_process_data`:
 - טענת כניסה: נתוני הודעה (שמות תהליכים).
 - טענת יציאה: אין
 - תיאור: מטפל בנתוני שמות תהליכים מהלקוח.
- `handle_cpu_data`:
 - טענת כניסה: נתוני הודעה (נתוני שימוש במעבד).
 - טענת יציאה: אין
 - תיאור: מטפל בנתוני שימוש במעבד מהלקוח.
- `handle_keyboard_data`:
 - טענת כניסה: נתוני הודעה (אירועי לוח מקשים).
 - טענת יציאה: אין

- תיאור: מטפל בנתוני אירועי לוח מקשים מהלקוח.

DB.py

המודול נעזר במחלקת MessageParser מ protocol.py

Class: DBHandler

תפקיד: מחלקת בסיס לטיפול בפעולות מסד נתונים. היא מספקת את הבסיס לחיבור, שאלות וניהול מסד נתונים. SQLite היא נועדה להיות מחלקה ממנה יורשות מחלקות אחרות שצריכות גישה למסד נתונים.

תכונות:

- DB_NAME (str): קבוע ברמת המחלקה המגדיר את שם קובץ מסד הנתונים ("server_db.db").
- conn (sqlite3.Connection): משתנה מופע המייצג את החיבור למסד הנתונים. SQLite.
- cursor (sqlite3.Cursor): משתנה מופע המייצג את אובייקט הסמן המשמש לביצוע שאלות. SQL.
- table_name (str): משתנה מופע המאחסן את שם הטבלה הראשית שהמטפל משויך אליה.
- _lock (threading.Lock): נעילת ת'רד כדי להבטיח פעולות מסד נתונים בטוחות לת'רד.

פעולות:

- init:
 - טענת כניסה: אובייקט חיבור למסד נתונים, אובייקט סמן למסד נתונים, שם הטבלה.
 - טענת יציאה: אין.
 - תיאור: מאתחל חיבור למסד נתונים באמצעות אובייקטי חיבור וסמן קיימים.
- connect_DB:
 - טענת כניסה: שם מסד הנתונים.
 - טענת יציאה: טאפל המכיל אובייקט חיבור ואובייקט סמן.
 - תיאור: מקים חיבור למסד נתונים ומחזיר אובייקט חיבור וסמן.
- close_DB:
 - טענת כניסה: אובייקט סמן, אובייקט חיבור.
 - טענת יציאה: אין.
 - תיאור: סוגר את החיבור למסד הנתונים.
- clean_deleted_records_DB:
 - טענת כניסה: אין.
 - טענת יציאה: אין.
 - תיאור: מנקה את כל הרשומות שנמחקו מהטבלה.

• delete_all_records_DB:

- טענת כניסה: אין.
- טענת יציאה: אין.
- תיאור: מוחק את כל הרשומות מהטבלה ומנקה את מסד הנתונים.

• commit:

- טענת כניסה: פקודת SQL, ארגומנטים לפקודה (אופציונלי).
- טענת יציאה: תוצאות השאילתה (ייתכן מבנה נתונים מורכב בהתאם לשאילתה).
- תיאור: מבצע פקודת SQL במסד הנתונים עם נעילה להבטחת סנכרון.

Class: UserLogsORM

תפקיד: מחלקה זו (בדומה ל-ORM) מרחיבה את DBHandler כדי לנהל רישום של נתוני פעילות משתמש. היא מספקת שיטות להוספת לוגים, אחזור סטטיסטיקות (כגון WPM, שימוש במעבד) וניתוח התנהגות משתמש. היא משתמשת בתבנית Singleton כדי להבטיח שקיים רק מופע אחד. משתמשת ב protocol.py על מנת לפרק הודעות לפי הפרוטוקול לפרמטרים.

תכונות:

- יורשת DBHandler, conn, cursor, table_name, _lock מ -

פעולות:

• new:

- טענת כניסה: המחלקה עצמה (cls), אובייקט חיבור למסד נתונים, אובייקט סמן למסד הנתונים, שם הטבלה.
- טענת יציאה: מופע סינגלטון של המחלקה.
- תיאור: מבטיח שתיווצר רק מופע יחיד של המחלקה ומאתחל אותו עם חיבור וסמן קיימים.

• client_setup_db:

- טענת כניסה: id של הלקוח.
- טענת יציאה: אין.
- תיאור: כותב לוגים בסיסיים שצריכים להיות עבור כל לקוח בעת התחברות.

• delete_id_records_DB:

- טענת כניסה: id של הלקוח.
- טענת יציאה: אין.
- תיאור: מוחק את כל הרשומות מהטבלה של id מסוימת.

• check_inactive__:

- טענת כניסה: id של הלקוח.

- טענת יציאה: טאפל המכיל את הזמן האחרון שהלקוח היה פעיל, ואת מספר הדקות שהוא לא פעיל.
- תיאור: בודק אם הלקוח כרגע לא פעיל.
- `:update_last_input__`
 - טענת כניסה: id של הלקוח.
 - טענת יציאה: אין.
 - תיאור: מעדכן את הזמן האחרון שהמשתמש ביצע אירוע קלט.
- `:get_total_active_time__`
 - טענת כניסה: id של הלקוח.
 - טענת יציאה: מספר דקות הפעילות הכוללות.
 - תיאור: מחשב את זמן הפעילות הכולל של המשתמש.
- `:update_cpu_usage__`
 - טענת כניסה: id של הלקוח, נתוני שימוש במעבד.
 - טענת יציאה: אין.
 - תיאור: מעדכן את רשומות ניצול המעבד.
- `:insert_data`
 - טענת כניסה: id של הלקוח, סוג הנתונים, הנתונים עצמם.
 - טענת יציאה: אין.
 - תיאור: מכניס נתונים לטבלת SQL, אם הרשומה כבר קיימת מגדיל את המונה שלה.
- `:get_process_count`
 - טענת כניסה: id של הלקוח.
 - טענת יציאה: רשימת טאפלים של שם התהליך ומספר הפעמים שנפתח.
 - תיאור: מקבל את מספר הפעמים שכל תהליך נפתח עבור לקוח מסוים.
- `:get_inactive_times`
 - טענת כניסה: id של הלקוח.
 - טענת יציאה: רשימת טאפלים של זמן ומשך חוסר פעילות.
 - תיאור: מחשב את זמני חוסר הפעילות של המשתמש.
- `:get_wpm`
 - טענת כניסה: id של הלקוח, רשימת זמני חוסר פעילות, אינדיקציה אם היה חוסר פעילות אחרי האירוע האחרון.
 - טענת יציאה: מספר מילים לדקה ממוצע.

- תיאור: מחשב את ממוצע המילים לדקה שהמשתמש מקליד תוך התעלמות מזמנים לא פעילים.
- `get_cpu_usage`:
 - טענת כניסה: id של הלקוח.
 - טענת יציאה: טאפל של מילון ליבות מעבד וניצולן ורשימת זמני לוגים.
 - תיאור: מקבל את כל הלוגים של ניצול המעבד.
- `get_active_precentage`:
 - טענת כניסה: id של הלקוח.
 - טענת יציאה: אחוז הזמן שהמשתמש היה פעיל.
 - תיאור: מחשב את אחוז הזמן שהמשתמש היה פעיל.
- `get_reached_out_ips`:
 - טענת כניסה: id של הלקוח.
 - טענת יציאה: רשימת כתובות IP שהמשתמש פנה אליהן.
 - תיאור: מקבל את כל כתובות ה-IP שלקוח מסוים פנה אליהן.

Class: UserId

תפקיד: מחלקה זו גם מרחיבה את DBHandler ומשתמשת בתבנית Singleton. היא אחראית על ניהול זיהוי משתמשים במערכת. היא מאחסנת ומאחזרת מידע משתמשים על סמך כתובות MAC ושמות מארחים.

תכונות:

- יורשת `DBHandler`, `conn`, `cursor`, `table_name`, `_lock` – מ

פעולות:

- `new`:
 - טענת כניסה: המחלקה עצמה (cls), אובייקט חיבור למסד נתונים, אובייקט סמן למסד הנתונים, שם הטבלה.
 - טענת יציאה: מופע סינגלטון של המחלקה.
 - תיאור: מבטיח שתיווצר רק מופע יחיד של המחלקה ומאתחל אותו עם חיבור וסמן קיימים.
- `delete_user`:
 - טענת כניסה: id של לקוח.
 - טענת יציאה: אין.
 - תיאור: מוחק לקוח מהטבלה.
- `insert_data`:

- טענת כניסה: כתובת MAC של משתמש, שם המשתמש ו id (אופציונלי).
- טענת יציאה: ערך בוליאני המציין אם המשתמש כבר מחובר.
- תיאור: מכניס נתונים לטבלת SQL, בודק אם ה-MAC והשם כבר בשימוש, אם כן מוסיף מספרים לשם.

• update_name:

- טענת כניסה: שם קודם, שם חדש.
- טענת יציאה: אין.
- תיאור: מנהל משנה שם ללקוח.

• check_user_existence:

- טענת כניסה: שם של הלקוח.
- טענת יציאה: ערך בוליאני (או מספר שלם המציין קיום).
- תיאור: בודק אם לקוח מסוים כבר מחובר.

• get_clients:

- טענת כניסה: אין.
- טענת יציאה: רשימת טאפלים של id ושם לקוח.
- תיאור: מקבל את כל הנתונים על לקוחות - id ושם לקוח.

• get_mac_by_id:

- טענת כניסה: id של לקוח.
- טענת יציאה: כתובת MAC.
- תיאור: מקבל את כתובת ה-MAC המתאימה של מחשב לפי id.

• get_id_by_hostname:

- טענת כניסה: השם של הלקוח.
- טענת יציאה: id של לקוח.
- תיאור: מקבל id של לקוח לפי השם.

process limit.py

Class: ProcessDebouncer

תפקיד: מחלקה זו מטפלת בסינון תהליכים בעת ביצוע log לתוך המסד נתונים. המחלקה מספקת למשתמש את היכולת לדעת האם תהליך מסוים נרשם במערכת ברגעים האחרונים בכדי לא לבצע לו log חוזר (מערכות הפעלה פותחות מספר רב של תהליכים שקשורים לתהליך מסוים ברגע שהוא נפתח, ברגע שאין את הסינון שמסופק על ידי מחלקה זו, על ידי פתיחה פשוטה של תהליך אצל הלקוח במסד נתונים ישמר עשרות אם לא מאות log'ים על אותו תהליך).

תכונות:

- `time_limit (int)` : זמן מקסימלי של שהייה בין `log` של תהליך מסוים.
- `max_process (int)`: מספר מקסימלי של תהליכים בהם המערכת יודעת לטפל בו-זמנית.
- `cache (OrderedDict)`: הזיכרון בו נשמרים התהליכים – מילון עם אפשרות של סדר בין המפתחות.

פעולות:

- `__init__` :
 - טענת כניסה:
 - `time_limit (int)`: מספר שלהגבלת זמן של שהייה בין כל `log` של תהליך.
 - `max_processes (int)`: מספר מקסימלי של תהליכים.
 - טענת יציאה: אין
 - תיאור: מאתחל את התכונות של המחלקה עם ערכי הפרמטרים.
- `should_log` :
 - טענת כניסה:
 - `process_name (str)` : שם של תהליך
 - טענת יציאה: בוליאני שמציין האם כדאי לבצע `log` לתהליך זה.
 - תיאור: בודק אם כבר בוצע `log` לאותו תהליך בזמן האחרון, אם לא כדאי לבצע `log` לתהליך זה.

encryption.py

Class:DiffieHellman

תפקיד: מחלקה זו מטפלת בחילופי מפתחות. Diffie-Hellman היא מאפשרת לשני צדדים להסכים על סוד משותף מעל תווך לא מאובטח .

תכונות:

- `prime (int)`: מספר ראשוני המשמש לחילופי המפתחות.
- `base (int)`: מספר הבסיס המשמש לחילופי המפתחות.
- `private_key (int)`: מפתח פרטי שנוצר באופן אקראי עבור צד זה.

פעולות:

- `__init__` :
 - טענת כניסה:
 - `prime (int)`: מספר ראשוני לחילופי המפתחות.
 - `base (int)`: מספר בסיס לחילופי המפתחות.

- טענת יציאה: אין
- תיאור: מאתחל את חילופי Diffie-Hellman עם מספר ראשוני ובסיס. אם prime או base הם 0, הוא מייצר פרמטרים חדשים.

• generate_private_key:

- טענת כניסה: אין
- טענת יציאה: מפתח פרטי (int)
- תיאור: מייצר מפתח פרטי אקראי.

• get_public_key:

- טענת כניסה: אין
- טענת יציאה: מפתח ציבורי (int)
- תיאור: מחשב ומחזיר את המפתח הציבורי.

• get_shared_secret:

- טענת כניסה:
- other_public_key (int): המפתח הציבורי של הצד השני.
- טענת יציאה: סוד משותף (int)
- תיאור: מחשב את הסוד המשותף באמצעות המפתח הציבורי של הצד השני.

Class: AESHandler

תפקיד: מחלקה זו מטפלת בהצפנה ופענוח של AES במצב CBC. היא מספקת שיטות להצפנת ופענוח נתונים באמצעות מפתח AES.

תכונות:

- key (bytes, אופציונלי): מפתח ה-AES המשמש להצפנה ופענוח. אם לא סופק, נוצר מפתח אקראי.
- cipher (AES): אובייקט הצפנה של AES.

פעולות:

• __init__:

- טענת כניסה:
- key (bytes, אופציונלי): מפתח AES.
- טענת יציאה: אין
- תיאור: מאתחל את AESHandler עם מפתח. אם לא ניתן מפתח, נוצר מפתח אקראי.

• encrypt:

- טענת כניסה:
- data (str או bytes): הנתונים להצפנה.

- טענת יציאה: נתונים מוצפנים (bytes)
- תיאור: מצפין נתונים באמצעות AES במצב CBC.
- decrypt:

- טענת כניסה:
- encrypted_data (bytes): הנתונים המוצפנים לפענוח.
- טענת יציאה: נתונים מפוענחים (bytes)
- תיאור: מפענח נתונים מוצפנים באמצעות AES במצב CBC.

Class: EncryptionHandler

תפקיד: מחלקה זו משלבת את חילופי Diffie-Hellman ואת הצפנת AES כדי לספק תקשורת מוצפנת. היא מטפלת בחילופי המפתחות הראשוניים ולאחר מכן משתמשת ב-AES להצפנת ופענוח הודעות.

תכונות:

- dh (DiffieHellman): מופע של המחלקה DiffieHellman לטיפול בחילופי המפתחות.
- aes_handler (AESHHandler, אופציונלי): מופע של המחלקה AESHandler לטיפול בהצפנת AES לאחר השגת סוד משותף.

פעולות:

- __init__:
 - טענת כניסה:
 - prime (int): מספר ראשוני לחילופי Diffie-Hellman.
 - base (int): בסיס לחילופי Diffie-Hellman.
 - טענת יציאה: אין
 - תיאור: מאתחל את EncryptionHandler עם פרמטרים של Diffie-Hellman.
- generate_shared_secret:
 - טענת כניסה:
 - other_public_key (int): המפתח הציבורי של הצד השני.
 - טענת יציאה: אין
 - תיאור: מייצר את הסוד המשותף ומאתחל את AESHandler עם מפתח נגזר.
- encrypt:
 - טענת כניסה:
 - data (str או bytes): הנתונים להצפנה.
 - טענת יציאה: נתונים מוצפנים (bytes)
 - תיאור: מצפין נתונים באמצעות AES.

• decrypt:

○ טענת כניסה:

▪ encrypted_data (bytes): הנתונים המוצפנים לפענוח.

○ טענת יציאה: נתונים מפוענחים (bytes)

○ תיאור: מפענח נתונים מוצפנים באמצעות AES.

protocol.py

המודול נעזר במחלקת EncryptionHandler מ encryption.py

Class: MessageParser

תפקיד: מחלקה זו אחראית על ניתוח ויצירת הודעות בהתאם לכללי הפרוטוקול המוגדרים. היא מגדירה את מבנה ההודעות, סוגי ההודעות ושיטות לקידוד ופענוח שלהן.

תכונות:

○ PROTOCOL_SEPARATOR (bytes): מפריד המשמש להפרדת שדות בהודעות הפרוטוקול.

○ קבועים רבים (str או bytes): המגדירים סוגי הודעות שונים (למשל, CLIENT_MSG_SIG, MANAGER_MSG_EXIT וכו').

פעולות:

• encode_str:

○ טענת כניסה:

▪ msg (str או bytes): ההודעה לקידוד.

○ טענת יציאה: הודעה מקודדת (bytes)

○ תיאור: מקודד מחרוזת ל-bytes.

• protocol_message_construct:

○ טענת כניסה:

▪ msg_type (str): סוג ההודעה.

▪ args*: ארגומנטים נוספים להכללה בהודעה.

○ טענת יציאה: הודעה בנויה (bytes)

○ תיאור: בונה הודעה בהתאם לכללי הפרוטוקול.

• protocol_message_deconstruct:

○ טענת כניסה:

▪ msg (bytes): זרם הבייטים של ההודעה.

▪ part_split (int, אופציונלי): מספר השדות להפרדה מתחילת ההודעה.

○ טענת יציאה: רשימה של שדות הודעה (רשימה של bytes)

- תיאור: מפרק הודעה לשדותיה בהתאם לכללי הפרוטוקול.

Class: TCPsocket

תפקיד: מחלקה זו עוטפת את פונקציונליות ה-socket של TCP, ומספקת שיטות ליצירה, חיבור, שליחה, קבלה וסגירה של socket. היא מפשטת את פעולות ה-socket עבור המחלקות האחרות.

תכונות:

- `MSG_LEN_LEN (int)`: אורך השדה המשמש לציון אורך ההודעה.
- `sock (socket.socket)`: אובייקט ה-socket הבסיסי.
- `ip (str, אופציונלי)`: כתובת ה-IP של ה-socket.

פעולות:

- `__init__`:
 - טענת כניסה:
 - `sock (socket.socket, אופציונלי)`: אובייקט socket קיים.
 - טענת יציאה: אין
 - תיאור: יוצר socket TCP חדש או עוטף socket קיים.
- `set_timeout`:
 - טענת כניסה:
 - `time (float או int)`: זמן timeout בשניות.
 - טענת יציאה: אין
 - תיאור: מגדיר timeout ל-socket.
- `get_ip`:
 - טענת כניסה: אין
 - טענת יציאה: כתובת ה-IP של ה-socket (str)
 - תיאור: מחזיר את כתובת ה-IP של ה-socket.
- `create_server_socket`:
 - טענת כניסה:
 - `bind_ip (str)`: כתובת ה-IP לקשירת השרת.
 - `bind_port (int)`: הפורט לקשירת השרת.
 - `server_listen (int)`: מספר החיבורים המקסימלי להאזנה.
 - טענת יציאה: אין
 - תיאור: מכין socket שרת TCP.

• `:server_socket_recv_client`

- טענת כניסה: אין
- טענת יציאה: אובייקט `socket` של לקוח (`socket.socket`)
- תיאור: השרת מקבל לקוח חדש.

• `:client_socket_connect_server`

- טענת כניסה:
- `dst_ip (str)`: כתובת ה-IP של השרת.
- `dst_port (int)`: הפורט של השרת.
- טענת יציאה: אין
- תיאור: מחבר `socket` לקוח לשרת.

• `:close`

- טענת כניסה: אין
- טענת יציאה: אין
- תיאור: סוגר את ה-`socket`.

• `:send`

- טענת כניסה:
- `data (bytes)`: הנתונים לשליחה.
- טענת יציאה: אין
- תיאור: שולח נתונים דרך ה-`socket`.

• `:recv`

- טענת כניסה:
- `length (int)`: אורך הנתונים לקבלה.
- טענת יציאה: נתונים שהתקבלו (`bytes`)
- תיאור: מקבל נתונים מה-`socket`.

Class: client

תפקיד: מחלקה זו מרחיבה את `TCPsocket` כדי לטפל בתקשורת ספציפית בצד הלקוח. היא מוסיפה פונקציונליות להצפנה ופענוח של הודעות, וכן ניהול של הודעות "לא בטוחות".

תכונות:

- יורשת `__ip__`, `sock`, מ-`TCPsocket`.
- `__encryption` (`EncryptionHandler`, אופציונלי): מופע של `EncryptionHandler` לטיפול בהצפנה.

- `unsafe_msg_cnt (int)___`: מונה של הודעות "לא בטוחות" שהתקבלו.

פעולות:

- `___init___`:
 - טענת כניסה:
 - `manager (bool, אופציונלי)`: דגל המציין אם זהו לקוח מנהל.
 - טענת יציאה: אין
 - תיאור: מאתחל אובייקט לקוח, כולל טיפול בהצפנה אם נדרש.
- `set_address`:
 - טענת כניסה:
 - כתובת MAC (`str`) חדשה ללקוח
 - טענת יציאה: אין
 - תיאור: משנה את התכונה של האובייקט של הכתובת
- `get_address`:
 - טענת כניסה: אין
 - טענת יציאה: מחזיר את הכתובת MAC
 - תיאור: מחזיר את הכתובת MAC של הלקוח/מנהל
- `exchange_keys`:
 - טענת כניסה: אין
 - טענת יציאה: בוליאני המעיד האם ההחלפת מפתחות עברה בהצלחה
 - תיאור: מבצע החלפת מפתחות עם הצד השני המחובר
- `connect`:
 - טענת כניסה:
 - `dst_ip (str)`: כתובת ה-IP של השרת.
 - `dst_port (int)`: הפורט של השרת.
 - טענת יציאה: בוליאני המציין הצלחה או כישלון של החיבור.
 - תיאור: מתחבר לשרת ומטפל בחילופי מפתחות הצפנה.
- `protocol_recv`:
 - טענת כניסה: אין
 - טענת יציאה: רשימה של נתוני הודעה (רשימה של `bytes`)
 - תיאור: מקבל הודעה מהשרת, מפענח אותה אם יש צורך ומחזיר את הנתונים.
- `protocol_send`:
 - טענת כניסה:
 - `msg_type (str)`: סוג ההודעה.
 - `*args`: נתוני ההודעה.
 - `encrypt (bool, אופציונלי)`: דגל המציין אם להצפין את ההודעה.

- טענת יציאה: אין
- תיאור: שולח הודעה לשרת, מצפין אותה אם יש צורך.
- unsafe_msg_cnt_inc:
- טענת כניסה:
- safety (int): סף הבטיחות למספר הודעות לא בטוחות.
- טענת יציאה: בוליאני המציין אם לנתק את הלקוח.
- תיאור: מגדיל את מונה ההודעות הלא בטוחות ומחזיר אם יש לנתק את הלקוח.
- reset_unsafe_msg_cnt:
- טענת כניסה: אין
- טענת יציאה: אין
- תיאור: מאפס את מונה ההודעות הלא בטוחות

Class: Server

תפקיד: מחלקה זו מרחיבה את TCPsocket כדי לטפל בפעולות ספציפיות בצד השרת. היא מגדירה את כתובות ה-IP והיציאות של השרת ומספקת שיטות לקבלת חיבורים מלקוחות.

תכונות:

- יורשת __ip__ sock, מ-TCPsocket.
- SERVER_BIND_IP (str): כתובת ה-IP שעליה השרת יופעל.
- SERVER_BIND_PORT (int): הפורט שעליו השרת יאזין.

פעולות:

- __init__:
- טענת כניסה:
- server_listen (int, אופציונלי): מספר החיבורים המקסימלי להאזנה.
- טענת יציאה: אין
- תיאור: יוצר socket שרת TCP.
- recv_client:
- טענת כניסה: אין
- טענת יציאה: אובייקט client
- תיאור: מקבל לקוח חדש ומחזיר אובייקט client המייצג את החיבור.

אתקן את התיאורים כך שתכונות יצינו את המשתנים בקבצים:

cpu_stats.c

תפקיד: קובץ זה אחראי על חישובי ניצולת מעבד. הוא מספק פונקציות לקבלת זמן פעילות ליבות מעבד ותכונות הקשורות לחישוב העומס.

תכונות:

- `REAL_TIME_LENGTH` (קבוע): גודל מחרוזת הזמן.
- `TIME_ZONE_DIFF` (קבוע): הפרש אזור זמן לחישוב הזמן המקומי.

פעולות:

- `get_cpu_idle`:
 - טענת כניסה:
 - `core (int)`: מזהה הליבה שעבורה יש לקבל את זמן הסרק.
 - טענת יציאה: זמן הסרק (`unsigned long`)
 - תיאור: מחזיר את הזמן הכולל שליבת המעבד הייתה במצב סרק.
- `get_cpu_active`:
 - טענת כניסה:
 - `core (int)`: מזהה הליבה שעבורה יש לקבל את זמן הפעילות.
 - טענת יציאה: זמן פעילות כולל (`unsigned long`)
 - תיאור: מחזיר את הזמן הכולל שליבת המעבד הייתה פעילה בכל המצבים.
- `get_real_time`:
 - טענת כניסה:
 - `time_buf (char*)`: מצביע למחרוזת שבה יאוחסן הזמן.
 - טענת יציאה: אין
 - תיאור: ממלא את המחרוזת המסופקת עם תאריך ושעה נוכחיים בפורמט "YYYY-MM-DD HH:MM:SS".

kClientHook.c

המודול נעזר ב `cpu_stats.c` `hide_module.c` `hide_tcp_sock.c` `protocol.c` `tcp_socket.c` `transmission.c` `workqueue.c`

תפקיד: מודול קרנל לניטור פעילות מערכת. מודול זה מחבר עצמו לפונקציות קרנל מרכזיות כדי לאסוף מידע על יצירת תהליכים, אירועי קלט, ניצולת מעבד ותקשורת רשת.

תכונות:

- `kps (struct kprobe[PROBES_SIZE])`: מערך של מבני `kprobe` להתחברות לפונקציות קרנל.
- `unhide_seq_index (int)`: אינדקס מעקב אחרי רצף מקשים לביטול הסתרת המודול.
- `hide_seq_index (int)`: אינדקס מעקב אחרי רצף מקשים להסתרת המודול.

- `cpu_idle_time (unsigned long[NR_CPUS])`: מערך לשמירת זמני סרק קודמים של כל ליבת מעבד.
- `cpu_actv_time (unsigned long[NR_CPUS])`: מערך לשמירת זמני פעילות קודמים של כל ליבת מעבד.
- `first_run (bool)`: דגל המציין אם זהו הריצה הראשונה של חישוב עומס מעבד.
- `unreg_kprobes (atomic_t)`: משתנה אטומי למניעת הסרה כפולה של ה-kprobes.
- `PROBES_SIZE` (קבוע): גודל מערך ה-kprobes.
- `CPU_USAGE_DELAY` (קבוע): ההשהיה בין חישובי עומס מעבד.
- `BUFFER_SIZE` (קבוע): גודל מאגר לאחסון הודעות.

פעולות:

- `handler_pre_do_fork`
 - טענת כניסה :
 - `kp (struct kprobe*)`: מצביע ל-kprobe.
 - `regs (struct pt_regs*)`: מצביע לרגיסטרים.
 - טענת יציאה: קוד שגיאה(int)
 - תיאור: מתעד תהליכים חדשים שנוצרים במערכת.
- `handler_pre_calc_global_load`
 - טענת כניסה :
 - `kp (struct kprobe*)`: מצביע ל-kprobe.
 - `regs (struct pt_regs*)`: מצביע לרגיסטרים.
 - טענת יציאה: קוד שגיאה(int)
 - תיאור: מחשב ושולח מידע על ניצולת המעבד.
- `handler_pre_input_event`
 - טענת כניסה :
 - `kp (struct kprobe*)`: מצביע ל kprobe
 - `regs (struct pt_regs*)`: מצביע לרגיסטרים.
 - טענת יציאה: קוד שגיאה(int)
 - תיאור: מנטר אירועי קלט ובודק רצף מקשים להסתרה או חשיפה של המודול.
- `handler_pre_inet_sendmsg`
 - טענת כניסה :
 - `kp (struct kprobe*)`: מצביע ל kprobe

▪ `regs (struct pt_regs*)` מצביע לרגיסטרים.

- טענת יציאה: קוד שגיאה (int)
- תיאור: מנטר תקשורת רשת יוצאת ומקטלג אותה לפי סוגים.

• `register_probes`:

- טענת כניסה: אין
- טענת יציאה: קוד שגיאה (int)
- תיאור: רושם את כל ה `kprobes`-במערכת.

• `unregister_probes`:

- טענת כניסה :
- `max_probes (int)` מספר ה `probes`-המקסימלי להסרה.

- טענת יציאה: אין
- תיאור: מסיר את כל ה `kprobes`-שנרשמו.

• `hook_init`:

- טענת כניסה: אין
- טענת יציאה: קוד שגיאה (int)
- תיאור: פונקציית אתחול המודול. מאתחלת את כל רכיבי המודול ורושמת את ה-
`kprobes`.

• `hook_exit`:

- טענת כניסה: אין
- טענת יציאה: אין
- תיאור: פונקציית יציאה של המודול. משחררת את כל המשאבים ומסירה את ה-
`kprobes`.

mac find.c

תפקיד: קובץ זה אחראי על להחזיר כתובת IP קבועה לכל מחשב (הכוונה לא יחזיר כתובת של כרטיס רשת אחר במחשב)

תכונות: אין

פעולות:

- `get_mac_address`:
- `mac_buf (char *)` מצביע למחרוזת היעד
- טענת יציאה: כתובת MAC של כרטיס רשת במחשב.
- תיאור: עובר על כל כתובות ה MAC במחשב ומחזיר את הכתובת בעלת הערכים הנמוכים ביותר בכדי להבטיח כתובת קבועה כל פעם.

protocol.c

המודול נעזר ב `workqueue.c` `transmission.c`

תפקיד: קובץ זה אחראי על פורמט הודעות לפי פרוטוקול התקשורת. הוא מספק פונקציות ליצירת הודעות בפורמט הדרוש ושליחתן.

תכונות:

- `dAddress (char*)` כתובת IP של היעד לשליחת ההודעות.
- `dPort (uint16_t)` פורט היעד לשליחת ההודעות.
- `BUFFER_SIZE` (קבוע): גודל מאגר להודעות.
- `SIZE_OF_SIZE` (קבוע): גודל שדה אורך ההודעה בפרוטוקול.

פעולות:

- `protocol_format`
 - טענת כניסה :
 - `dst (char*)` מצביע למחרוזת היעד.
 - `format (const char*)` פורמט המחרוזת.
 - ... (משתנים): פרמטרים משתנים לפורמט.
 - טענת יציאה: אורך ההודעה המפורמטת או קוד שגיאה (`int`)
 - תיאור: מפרמט הודעה לפי פרוטוקול, מוסיף את אורך ההודעה בתחילתה.
- `protocol_send_message`
 - טענת כניסה :
 - `format (const char*)` פורמט המחרוזת.
 - ... (משתנים): פרמטרים משתנים לפורמט.
 - טענת יציאה: קוד שגיאה (`int`)
 - תיאור: מפרמט הודעה ושולח אותה דרך תור העבודה.

tcp_socket.c

תפקיד: קובץ זה מטפל בתקשורת TCP במערכת. הוא מספק פונקציות ליצירת סוקטי TCP, התחברות, שליחת נתונים וסגירת חיבורים.

תכונות:

- `SOCK_TIMEO` (קבוע): זמן פסק (timeout) בננו-שניות לפעולות סוקט.
- `MODULE_MARK` (קבוע): סימן מיוחד עבור סוקטים שנוצרו על ידי המודול.

פעולות:

- `tcp_sock_create`

- טענת כניסה: אין
- טענת יציאה: מצביע לסוקט (struct socket*) TCP או קוד שגיאה
- תיאור: יוצר סוקט TCP חדש עם הגדרות מתאימות.
- tcp_sock_connect:
 - טענת כניסה :
 - sock (struct socket*) מצביע לסוקט.
 - dst_ip (const char*) כתובת IP יעד.
 - port (uint16_t) פורט יעד.
 - טענת יציאה: קוד שגיאה(int)
 - תיאור: מתחבר לכתובת IP ופורט יעד מוגדרים.
- tcp_send_msg:
 - טענת כניסה :
 - sock (struct socket*) מצביע לסוקט.
 - msg (const char*) הודעה לשליחה.
 - length (size_t) אורך ההודעה.
 - טענת יציאה: קוד שגיאה(int)
 - תיאור: שולח הודעה דרך TCP.
- check_valid_connection:
 - טענת כניסה:
 - sock (struct socket *) מצביע לסוקט
 - טענת יציאה: מספר שמייצג האם הסוקט מחובר לשרת.
 - תיאור: בודק אם הסוקט מחובר אל השרת.
- tcp_sock_close:
 - טענת כניסה :
 - sock (struct socket*) מצביע לסוקט לסגירה.
 - טענת יציאה: אין
 - תיאור: סוגר סוקט TCP.
- check_sock_mark:
 - טענת כניסה :
 - sock (struct sock*) מצביע למבנה sock.

- mark (__u32) סימן לבדיקה.
- טענת יציאה: אמת אם הסימן תואם, שקר אחרת (bool)
- תיאור: בודק אם לסוקט יש סימן מסוים.

transmission.c

המודול נעזר ב workqueue.c tcp_socket.c protocol.c mac_find.c file_storage.c

תפקיד: קובץ זה אחראי על העברת נתונים דרך רשת. הוא מנהל חיבור TCP, שולח נתונים ומטפל במצבי ניתוק.

תכונות:

- sock (struct socket*): מצביע לסוקט TCP המשמש לתקשורת.
- connected (bool): מציין אם המודול מחובר כרגע.
- trns_mutex (struct mutex): מנעול להבטחת גישה בטוחה לסוקט מריבוי חוטים.
- cred (char[BUFFER_SIZE]): מחרוזת המשמשת לאחסון פרטי אימות.
- BUFFER_SIZE (קבוע): גודל מאגר להודעות.
- MAC_SIZE (קבוע): גודל כתובת MAC.

פעולות:

- disconnect:
 - טענת כניסה :
 - msg (char*): הודעה שנכשלה בשליחה.
 - len (size_t): אורך ההודעה.
 - טענת יציאה: אין
 - תיאור: מנתק את החיבור הנוכחי ומגבה את ההודעה שלא נשלחה.
- transmit_data:
 - טענת כניסה :
 - work (struct work_struct*): מבנה העבודה המכיל את ההודעה לשליחה.
 - טענת יציאה: אין
 - תיאור: מתחבר לשרת אם צריך, שולח את ההודעה הנוכחית ואת כל ההודעות שבגיבוי.
- handle_credentials:
 - טענת כניסה: אין
 - טענת יציאה: אין

○ תיאור: מכין את פרטי האימות עבור החיבור.

• :data_transmission_init

○ טענת כניסה: אין

○ טענת יציאה: אין

○ תיאור: מאתחל את כל האובייקטים הדרושים להעברת נתונים.

• :data_transmission_release

○ טענת כניסה: אין

○ טענת יציאה: אין

○ תיאור: משחרר את כל המשאבים שנוצרו על ידי מודול העברת הנתונים.

workqueue.c

תפקיד: קובץ זה מנהל תור עבודה עבור פעולות אסינכרוניות. הוא מאפשר שליחת הודעות ברקע בלי לחסום את התהליך הראשי.

תכונות:

• workqueue (struct workqueue_struct*): תור עבודה גלובלי להעברת נתונים.

• BUFFER_SIZE(קבוע): גודל מאגר להודעות.

פעולות:

• :init_singlethread_workqueue

○ טענת כניסה :

▪ workqueue_name (const char*) שם תור העבודה.

○ טענת יציאה: קוד שגיאה (int)

○ תיאור: יוצר תור עבודה חד-חוט.

• :release_singlethread_workqueue

○ טענת כניסה: אין

○ טענת יציאה: אין

○ תיאור: משחרר את תור העבודה ומחכה לסיום כל העבודות התלויות.

• :workqueue_message

○ טענת כניסה :

▪ queued_function (function pointer): פונקציה שתקרא כאשר העבודה מתבצעת.

▪ msg (const char*): הודעה לשליחה.

▪ length (size_t): אורך ההודעה.

- טענת יציאה: אין
- תיאור: מכניס הודעה חדשה לתור העבודה לשליחה.

hide module.c

תפקיד: קובץ זה מספק פונקציונליות להסתרה וחשיפה של המודול במערכת. באמצעות מניפולציה של רשימת המודולים, ניתן להסתיר את המודול מהרשימה הגלויה של מודולי קרנל.

תכונות:

- hidden (int) מציין אם המודול מוסתר כרגע.
- prev_module (struct list_head*) מצביע לרשומה הקודמת ברשימת המודולים.

פעולות:

- hide_this_module:
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: מסתיר את המודול הנוכחי על ידי הסרתו מרשימת המודולים של הקרנל.
- unhide_this_module:
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: מחזיר את המודול הנוכחי לרשימת המודולים של הקרנל כדי שיהיה גלוי.

hide tcp sock.c

תפקיד: קובץ זה מספק מנגנון להסתרת חיבורי TCP ספציפיים מכלי ניטור כמו netstat, וכן להסתרת חבילות תקשורת (packets) היוצאות לכתובת ופורט מסוימים. הוא משתמש במנגנון ftrace של הקרנל כדי לשנות (hook) פונקציות קרנל מרכזיות.

תכונות:

- ftrace_hook (struct): מבנה נתונים המכיל מידע על הפונקציה שעליה מבצעים hook
- sock_hidden (int): מציין האם החיבור מוסתר כרגע.
- tcp4_seq_show_address (tcp4_seq_show_t): מצביע לפונקציה המקורית שמציגה חיבורי TCP.
- dev_queue_xmit_nit_addr (dev_queue_xmit_nit_t): שמטפלת בחבילות תקשורת יוצאות. מצביע לפונקציה המקורית

פעולות:

- register_tcp_sock_hook:
 - טענת כניסה: אין
 - טענת יציאה: 0 אם ההסתרה הצליחה, קוד שגיאה אחרת

- תיאור: רושם hooks על פונקציות הקרנל כדי להסתיר חיבור TCP ספציפי.
- unregister_tcp_sock_hook:
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: מסיר את ה hooks-ומפסיק את הסתרת החיבור.
- tcp4_seq_show_hook:
 - טענת כניסה: מצביע לקובץ רצף ומצביע למבנה נתונים
 - טענת יציאה: 0 אם החיבור מוסתר, אחרת התוצאה של הפונקציה המקורית
 - תיאור: בודק אם החיבור הנוכחי מתאים לכתובת והפורט שיש להסתיר.
- dev_queue_xmit_nit_hook:
 - טענת כניסה: מצביע לחבילת תקשורת ומצביע למכשיר רשת
 - טענת יציאה: אין
 - תיאור: בודק אם חבילת התקשורת מיועדת לכתובת והפורט שיש להסתיר.

file_storage.c

תפקיד: קובץ זה מספק מנגנון אחסון קבצים למערכת 'silent_net' המשמש לגיבוי נתונים כאשר השרת אינו זמין. הוא מיישם מנגנון של חוצץ מעגלי (circular buffer) המאפשר שמירת נתונים באופן רציף וקריאתם באופן יעיל

תכונות:

- filename (char*): שם הקובץ בו נשמרים הנתונים. ("/var/tmp/.syscache")
- file (struct file*): מצביע לקובץ הפתוח.
- read_pos/write_pos (loff_t): מיקומי הקריאה והכתיבה בקובץ.
- read_pos_offset/write_pos_offset (loff_t): מיקומי הקריאה והכתיבה בקובץ.

פעולות:

- file_storage_init:
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: מאתחל את מנגנון אחסון הקבצים, פותח את הקובץ ומשחזר את מיקומי הקריאה והכתיבה האחרונים.
- file_storage_release:
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: משחרר את משאבי אחסון הקבצים וסוגר את הקובץ.

- truncate_file:
 - טענת כניסה: אין
 - טענת יציאה: אין
 - תיאור: מקצר את הקובץ כדי לפנות מקום לנתונים חדשים, תוך שמירה על מסרים שלמים.
- write_circular:
 - טענת כניסה: מצביע לנתונים ואורך הנתונים
 - טענת יציאה: אין
 - תיאור: כותב נתונים לחוץ המעגלי, מטפל במקרים של גלישה בחוץ.
- read_circular:
 - טענת כניסה: מצביע לחוץ יעד ואורך לקריאה
 - טענת יציאה: מספר הבתים שנקראו או קוד שגיאה
 - תיאור: קורא נתונים מהחוץ המעגלי, מטפל במקרים של גלישה בחוץ.
- backup_data_log:
 - טענת כניסה: מצביע לנתונים ואורך הנתונים
 - טענת יציאה: אין
 - תיאור: מגבה את הנתונים הנתונים בקובץ, כולל הוספת מפריד הודעות.
- read_backup_data_log:
 - טענת כניסה: מצביע לחוץ יעד
 - טענת יציאה: אורך ההודעה שנקראה או קוד שגיאה
 - תיאור: קורא הודעה מגיבוי הנתונים, כולל פענוח אורך ההודעה.

חשוב לציין!

שלל הקבצים שמהווים חלק מ GUI של הפרויקט לא נכתבו על ידי אלא נכתבו על ידי AI, כיוון שהקוד עוסק רק בהצגה גרפית של המידע ולא מעבר (כל שאר המידע מועבר על ידי פרוטוקול שלי) ביקשתי מ AI שיכתוב לי GUI לפרויקט שיציג את המידע בצורה נוחה לשימוש ויפה לעין, צד שאין לי בו הבנה מספיק מעמיקה. בעת כתיבת צד זה של הפרויקט פעמים רבות הדרכתני את ה AI בצורה מדויקת מה אני רוצה ואיך יראה ולא רק כתבתי פרומפט אחד ולקחתי את מה שהוא כתב, אלא זה היה תהליך ארוך של ניסוי וטעיה עד שהגעתי לתוצאה הרצויה. אך כן חשוב לציין כי לא אני זה שכתב את הקוד בכל הקשור לקבצי ה GUI (כן כתבתי את ההתמשקות נגד ה GUI עם קובץ python שמשתמש ב Flask) ולכן אינני בקיא מספיק בכדי לכתוב תיעוד לפונקציות בתוך הקבצים הללו. (הקבצים שנכתבו על ידי AI הינם קבצי js .css .html). בסופו של דבר אני מרוצה מהעבודה היוזואלית ש AI סיפק לי, הממשק הוא נוח לשימוש וקל להבנה אפילו לאדם שאינו מבין במערכת, ובנוסף על כל זה גם הוא ביצע עבודה יוצאת מן הכלל והממשק הגרפי גם נראה טוב מאוד, דבר שלא פחות חשוב משאר החלקים בפרויקטים כה גדולים. לתוצאה כזו של GUI לא יכולתי להגיע לבד, לכן העובדה ש AI כתב את כל הקשור לחלקים הגרפיים בפרויקט ראוייה לציין בתיק פרויקט.

בעיות אלגוריתמיות וקטעי קוד מיוחדים

בפרק הקודם בחלק בעיות אלגוריתם מרכזיות בפרויקט צוין לגבי אחסון המידע בצד הלקוח (בזמן שאין תקשורת בין השרת ללקוח) כאשר קיימת חשיבות לא להעמיס את המערכת במידע רב מדי. להלן שתי הפעולות המרכזיות – קריאה וכתובה בשיטה מעגלית (מתוך הקובץ file_storage.c). שתי הפונקציות הללו הן הלב של האלגוריתם, אשר הן מיישמות את שיטת הבאפר המעגלי עם קובץ.

```
// Writing to file in circular style
void write_circular(const char *data, size_t len) {
    ssize_t ret;
    loff_t original_write_pos = write_pos;

    if (!data || len == 0 || len > MAX_FILE_SIZE) {
        printk(KERN_ERR "Invalid parameters for write_circular\n");
        return;
    }

    write_pos %= MAX_FILE_SIZE; // Ensure write_pos is within bounds

    // Check space (with truncation if needed)
    size_t space_remaining;
    if (write_pos >= read_pos)
        space_remaining = MAX_FILE_SIZE - (write_pos - read_pos);
    else
        space_remaining = read_pos - write_pos;

    if (len >= space_remaining) {
        truncate_file(); // Free space by discarding old messages
    }

    // Handle wrap-around: Write in two parts if needed
    if (write_pos + len > MAX_FILE_SIZE) {
        size_t first_part = MAX_FILE_SIZE - write_pos;
        ret = safe_file_write(file, data, first_part, &write_pos);
        if (ret != first_part) {
            write_pos = original_write_pos; // Rollback on failure
            printk(KERN_ERR "Failed to write first part (ret=%zd)\n", ret);
            return;
        }

        // Update for second part
        data += first_part;
        len -= first_part;
        write_pos = 0;
    }

    // Write remaining data (or full data if no wrap-around)
    ret = safe_file_write(file, data, len, &write_pos);
    if (ret != len) {
        write_pos = original_write_pos; // Rollback on failure
        printk(KERN_ERR "Failed to write data (ret=%zd)\n", ret);
    }
}
```

```
// Reading from file in circular style
int read_circular(char *buf, size_t len) {
    ssize_t ret;
    loff_t original_read_pos = read_pos;

    read_pos %= MAX_FILE_SIZE; // Ensure read_pos is within bounds

    if (buf == NULL) {
        read_pos = (read_pos + len) % MAX_FILE_SIZE;
        return len;
    }

    if (!buf || len == 0 || len > MAX_FILE_SIZE) {
        printk(KERN_ERR "Invalid parameters for read_circular\n");
        return -EINVAL;
    }

    // Handle wrap-around: Read in two parts if needed
    if (read_pos + len > MAX_FILE_SIZE) {
        size_t first_part = MAX_FILE_SIZE - read_pos;
        ret = safe_file_read(file, buf, first_part, &read_pos);
        if (ret != first_part) {
            read_pos = original_read_pos; // Rollback on failure
            printk(KERN_ERR "Failed to read first part (ret=%zd)\n", ret);
            return ret < 0 ? ret : -EIO;
        }

        // Update for second part
        buf += first_part;
        len -= first_part;
        read_pos = 0;
    }

    // Read remaining data (or full data if no wrap-around)
    ret = safe_file_read(file, buf, len, &read_pos);
    if (ret != len) {
        read_pos = original_read_pos; // Rollback on failure
        printk(KERN_ERR "Failed to read data (ret=%zd)\n", ret);
        return ret < 0 ? ret : -EIO;
    }

    return len; // Total bytes read
}
```

– קטע קוד הבא נלקח מתוך הקובץ DB.py

```
def get_cpu_usage(self, mac : str):
    """
    Gets all logs of cpu usage

    INPUT: mac
    OUTPUT: Tuple of Dictionary of cpu cores and their usages and list of times of logs

    @mac: MAC address of user's computer
    """

    command = f"SELECT data FROM {self.table_name} WHERE mac = ? AND type = ?;"
    cores_logs = self.commit(command, mac, MessageParser.CLIENT_CPU_USAGE)[0][0]
    if isinstance(cores_logs, str):
        cores_logs = cores_logs.encode()
    cores_logs = cores_logs.split(b"|")

    logs = [log for i in cores_logs if len(i) > 1 for log in
i.split(MessageParser.PROTOCOL_SEPARATOR)]
    cpu_usage_logs = []

    core_usage = {}
    for log in logs:
        log = log.decode().split(",")
        core, usage = log[:2]

        if core not in core_usage:
            core_usage[core] = []
        core_usage[core].append(int(usage))

        if len(log) == 3:
            cpu_usage_logs.append(log[2])

    return core_usage, cpu_usage_logs
```

הפונקציה get_cpu_usage אחראית לשלוח ממסד הנתונים את לוגי השימוש במעבד (CPU) של מחשב לפי כתובת MAC. כל לוג מכיל את נתוני השימוש של כל ליבה במעבד, כאשר כל לוג מופרד באמצעות תו מפריד (protocol separator), ובסוף הלוג של הליבה האחרונה מופיע גם תאריך ושעה. הקוד מפענח את הלוגים, מפרק אותם לפי ליבות, ובונה מבנה נתונים שבו ניתן לראות עבור כל ליבה את רמות השימוש לאורך זמן. בנוסף, הוא שומר את רשימת הזמנים שבהם התקבלו הלוגים.

בדיקות ותוצאות

בדיקות משלב האפיון

1. בדיקת העברת נתונים לשרת ממחשב של עובד -

| | |
|--------------|---|
| מטרת בדיקה | לודא שהמידע שנשמר במערכת של העובד הינו המידע שמגיע אל השרת בשלמותו ושאינו איבוד מידע כחלק מהתהליך המסובך. |
| ביצוע בפועל | הדפסתי בצד הלקוח את המידע שהוא שמר מפעולות שונות ולצד זה הדפסתי את ההודעות שמתקבלות מלקוח שהתחבר לצד השרת, ווידאתי שההודעות אכן מכילות את אותו תוכן. בנוסף הסתכלתי ב Wireshark בכדי לראות באמת כל בית והאם זהה למידע שנשלח. |
| תוצאות בדיקה | הבדיקה עברה בהצלחה לאחר כמה תקלות בתחילת הבדיקה, המידע אכן מצליח להישלח בצורה מלאה אל מחשב אחר. |
| בעיות | בתחילת הבדיקה הבחנתי כי צד הלקוח לא מתחבר בכלל אל צד השרת (connect), לאחר כמה זמן ניסיתי לבדוק האם התוכנה מצליחה להתחבר ל socket על מחשב מאותו מחשב (אצל מחשב הלקוח) והדבר עבד. לאחר בדיקות רבות באינטרנט הבנתי כי הבעיה הייתה ה firewall של windows שאחד החוקים המוגדרים אצלו היה חסימת תקשורת בין ה VM למחשב שלי (כלומר שניהם מורצים על אותו מחשב), לאחר ביטול חוק זה התקשורת עבדה באופן מוצלח כמו שתואר בתוצאות הבדיקה. בעיה נוספת שצצה לאחר מכן הייתה אם לוקח למחשב זמן רב מידי לשלוח את ההודעה (מחשב השרת קורס) מחשב הלקוח היה קורס גם כן, זוהי הייתה בעיה בשל ביצוע פעולה blocking בתוך interrupt context דבר מסוכן מאוד בכתיבת קוד בקרנל, בעיה זו נפתרה על ידי שימוש ב workqueue – המידע ישמר תוך כדי interrupt context אבל שליחת המידע עצמה תתבצע על ידי העברת המידע ל singlethreaded queue שלכל הודעה שמתקבלת דוחפים אליו את ההודעה והוא ישלח את המידע, הסיבה שזה עובד זה מכיוון שאותו queue מנוהל על ידי thread, כלומר process context ושליחת ההודעות תתבצע מתי שהמערכת יכולה ולא מתי שהקוד שלי רוצה. |

2. בדיקת אמינות הנתונים –

| | |
|--------------|---|
| מטרת בדיקה | לודא שהמידע שנשמר אצל הלקוח מבטא את פעילות הלקוח בזמן אמת, לדוגמה – הלקוח פתח תהליך בשם game, על מחשב הלקוח לשמור את שם תהליך זה מיד לאחר פתיחת התהליך. כלומר על המידע להתאים לפעילות האמיתית של הלקוח בזמן אמת. |
| ביצוע בפועל | לכל פעולה שביצעתי לה hook בנוסף לכך הוספתי הדפסות המכילות את כל הנתונים שאפשר להשיג מאותה פעולה (מה – hook), לאחר מכן הפעלתי דברים שונים במערכת אשר יגרמו לקריאה לאותן פעולות, למשל: פתיחת תהליך, גלישה באתר אינטרנט, לחיצות מקלדת ועכבר, ועוד... |
| תוצאות בדיקה | בזכות השימוש במודול לביצוע hook הבנוי לתוך מערכת הפעלה Linux ושמו Kprobes, הבדיקות עברו באופן מוצלח בפעם הראשונה, כל המידע שהיה צריך להיות הופיע בזמן אמת. |
| בעיות | אין |

3. בדיקת השפעה על ביצועי מערכת –

| | |
|--------------|---|
| מטרת בדיקה | לוודא שהמערכת שנבנתה איננה מכבידה על המחשב עליו מורצת. אם המערכת לא מתפקדת בצורה מלאה בחלק מהמחשבים בשל יכולות נמוכות של המחשב זוהי בעיה שיש להתחשב בה, אחת ממטרות הפרויקט היא שהפרויקט ירוץ על כל המחשבים השונים עליו מורץ |
| ביצוע בפועל | המערכת אצל הלקוח הורצה על Virtual Box בו אפשר להגדיר את משאבי המערכת. הגדרתי נתונים מינימליים ביותר למערכת הפעלה ובדקתי אם המערכת מתפקדת כראוי וכל המידע נשלח כמו שצריך, בנוסף בזכות אחד הנתונים שנשלח אל המנהל (אחוזי פעולה של כל ליבת CPU) ניתן היה לבדוק בצד המנהל אם המחשב מתאמץ יותר מבדרך כלל. בנוסף בדקתי על כמה מחשבים שונים עם משאבים שונים. |
| תוצאות בדיקה | הבדיקה עברה כמצופה, הפרויקט לא מכביד כלל על המערכת בה הוא מורץ. כל מה שהמערכת מבצעת מאחורי הקלעים זה התעסקות עם סטרינגים ושליחת הודעה על ידי Kernel thread, דבר שלא אמור להכביד כלל על המערכת. אחוזי ה CPU היו זהים גם למתי שהתוכנית לא הייתה מותקנת על המחשב. |
| בעיות | אין |

4. בדיקת אבטחת נתונים

| | |
|--------------|---|
| מטרת בדיקה | לוודא שהנתונים המועברים בין המנהל לשרת מאובטחים לפי ההצפנות ואין דרך למאזין להבין את התקשורת. |
| ביצוע בפועל | לאחר שהמידע הוצפן הדפסתי אותו בכדי להסתכל ולראות האם באמת השתנה, בנוסף אם הצד השני הצליח לפענח את המידע לפי ההצפנות DH ו AES זאת אומרת שהמידע הוצפן כמו שצריך, לכן בדקתי אם הצד השני מצליח לפענח את המידע כמו שצריך למידע המקורי. |
| תוצאות בדיקה | המידע מוצפן כמו שצריך, שני הצדדים מצליחים להצפין ולפענח את המידע ביניהם. לא ניתן לפענח את המידע על ידי האזנה פשוטה. |
| בעיות | אין |

5. בדיקת החבאת תוכנית הלקוח

| | |
|--------------|---|
| מטרת בדיקה | לוודא שהתוכנית שמורצת אצל הלקוח מוחבאת והמשתמש הרגיל לא יוכל למצוא אותה ולהפריע בעבודתה. |
| ביצוע בפועל | לאחר הרצת התוכנית בדקתי באמצעות כלים שונים כגון netstat ו wireshark אם רואים את ההודעות הנשלחות אל השרת או את החיבור בין המחשב לשרת. בנוסף השתמשתי בפקודה lsmod בכדי לבדוק אם המערכת מזהה את התוכנה (כיוון שהיא מודול, כל שמות המודולים מודפסים בפקודה זו). |
| תוצאות בדיקה | הבדיקה עברה כמצופה, לא ניתן למצוא עקבות ברורות מצד העובד של התוכנה. |
| בעיות | לראשונה לאחר הסתרת המודול עצמו, לא חשבתי על האפשרות כאשר המנהל בעצמו רוצה להסיר את התוכנה ממחשב המשתמש. דבר זה לא יתאפשר אשר המודול כבוי לכן המחשב לא יכול למצוא את המודול בכדי להסיר אותו. בעיה זו נפתרה על ידי הגדרה סט של מקשים שצריך ללחוץ על המקלדת אחד אחרי השני ברצף בכדי להסיר/להציג את התוכנית. אותו סט מקשים רק המנהל אמור לדעת עליו וגם הוא סט מקשים שככל הנראה המשתמש לעולם לא ילחץ עליהם בטעות (אלא אם כן הוא מודע לאותו סט מקשים, וגם אם ילחץ על אותו סט מקשים בטעות כנראה לא יהיה מודע לכך). |

בדיקות נוספות למערכת

1. בדיקת גיבוי מידע אצל הלקוח

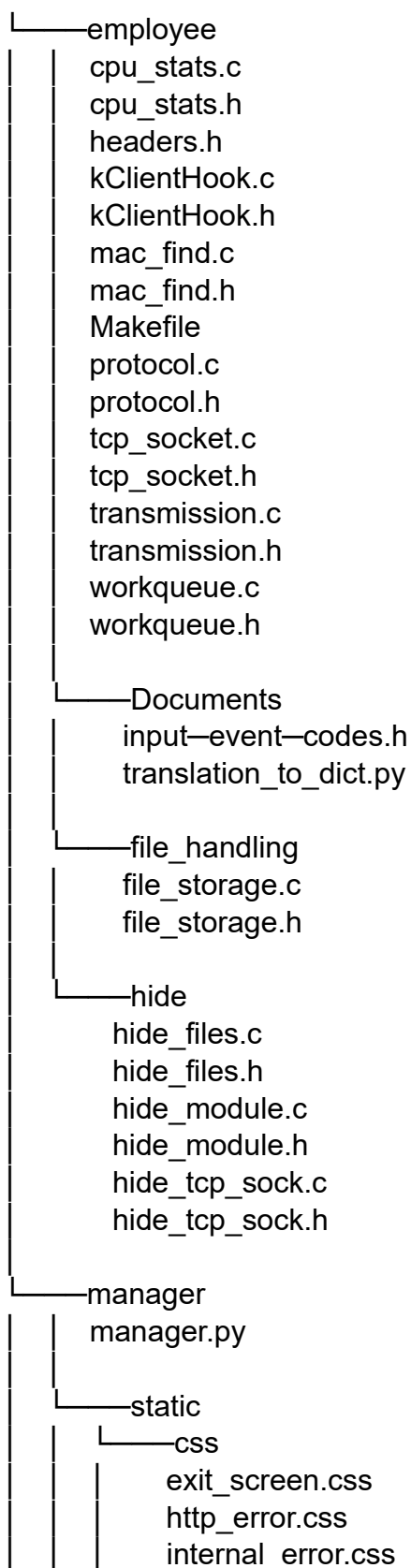
| | |
|--------------|--|
| מטרת בדיקה | לודא שכאשר השרת לא פועל הלקוח מסוגל לשמור מידע אצלו (עד לזמן מוגבל מוגדר) והמידע לא נאבד ברגעים בהם השרת לא מתפקד. |
| ביצוע בפועל | תוך כדי הרצה מלאה של התוכנית סגרתי את השרת בסגירה ברוטלית, ולאחר מכן הפעלתי תהליכים שונים ועוד שלל דברים אשר ידליקו פעולות של שליחת מידע. לאחר כמה דקות הדלקתי מחדש את המערכת ובדקתי האם המידע שהיה צריך להישלח נשלח אל השרת, אם המידע נשלח משמע הגיבוי הצליח. |
| תוצאות בדיקה | גיבוי המידע עובר בשלום, כל המידע שצריך להישמר נשמר ומצליח להישלח לשרת. |
| בעיות | במהלך הבדיקה שמתי לב לבעיה שמכיוון שטבלת אחוזי הליבות של המעבד שמה את הנתונים לפי הסדר שהם נשלחים, הטבלה נראית מוזרה מכיוון שהנקודות עצמן מונחות במקום הנכון אבל הקו בין הנתונים עובר לפי הסדר בהם הגיעו. בשביל לתקן את הבעיה בתוך הקוד שמציג את הטבלה כעת המידע עובר מיון לפי שעה לפני שמוצג מה שמתקן את הבעיה. |

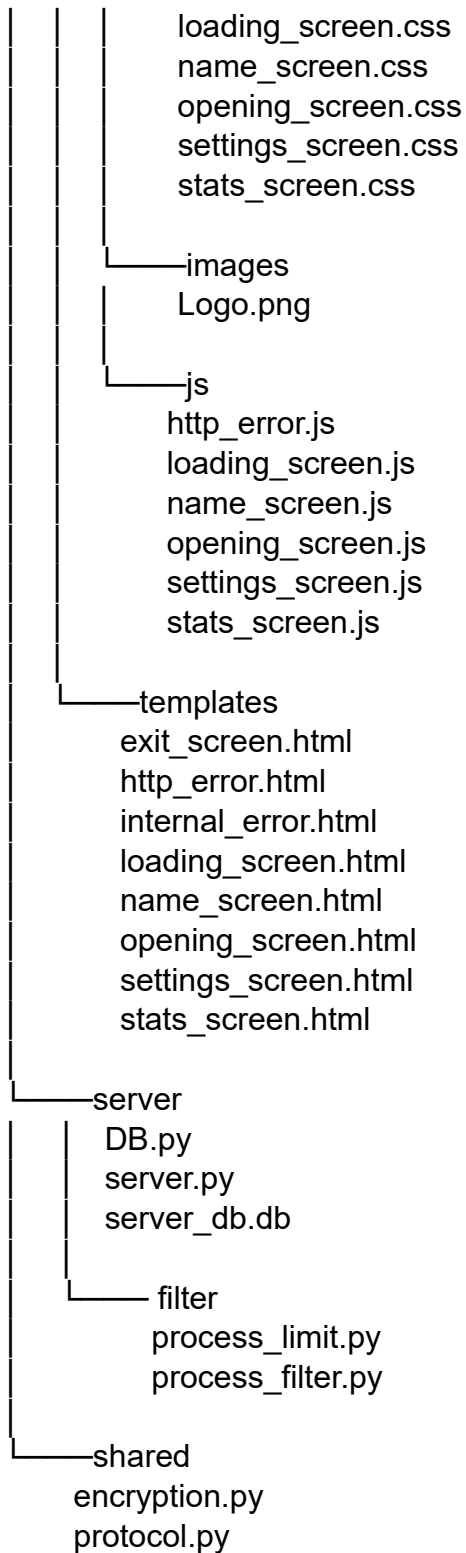
2. בדיקת רמת האבטחה והגבלת משתמשים

| | |
|--------------|--|
| מטרת בדיקה | לודא שהגבלות המנהל מיושמות על ידי השרת. |
| ביצוע בפועל | הגבלת כמות הלקוחות למספר קטן ממספר המחשבים הנבדקים, לדוגמה נגביל את המספר הלקוחות לאחד, ומכיוון שהמנהל הוא כבר מחובר אז שום לקוח לא אמור להיות מסוגל להתחבר. לאחר כמה זמן העליתי את ההגבלה לכך שמספר הלקוחות הרצויים כן יצליחו להתחבר וצריך לראות כי הם באמת מתחברים ללא שום עזרה חיצונית. בנוסף יצרתי לקוח ששולח הודעות סתמיות, כלומר הודעות שלא עובדות לפי הפרוטוקול, ובדקתי שבאמת לאחר מספר הודעות מסוים השרת מנתק אותו. |
| תוצאות בדיקה | הבדיקה עברה כמו שצריך, מספר הלקוחות לא עובר את ההגבלה ואכן השרת מנתק לקוחות שלא עובדים לפי הפרוטוקול. |
| בעיות | במהלך הבדיקה שמתי לב שכאשר אני מגביל את הלקוחות למספר מסוים, ותוך כדי מחבר מעל מספר זה של לקוחות ולאחר מכן מעלה את ההגבלה, הלקוחות לא מתחברים בחזרה אל השרת. לאחר בדיקה קצרה הבחנתי כי ה event שעליו מחכה ה main thread בשרת שאחראי על קבלת לקוחות לא מתעדכן כאשר מנהל משנה את ההגדרות להגדרות חדשות. בשביל לפתור זאת כל מה שעשיתי היה להוסיף בדיקה לאחר שינוי ההגדרות של האם ההגדרות החדשות מאפשרות נעילה/שחרור של ה event. |

מדריך למשתמש

עץ קבצים





התקנת מערכת

בכדי להריץ את הפרויקט צריך להתייחס לשלושת החלקים השונים של הפרויקט.

מנהל:

- סביבה: המערכת דורשת Python עם ספריית Flask. בנוסף קובץ encryption.py דורש את הספריות pycryptodome ו-cryptography. (דורש פייתון 3.6 ומעלה).
- כלים: נדרשים כלים מהספרייה המשותפת, כולל protocol.py ו-encryption.py, שמיובאים מהתיקייה './shared'.
- מיקום קבצים: התוכנית מצפה שקבצי פרוטוקול והצפנה יהיו בתיקיית shared מעל התיקייה הנוכחית. תבניות HTML נשמרות בתיקיית templates.
- נתונים התחלתיים: המערכת דורשת כתובת IPv4 של השרת ב argv, בנוסף דורשת סיסמת מנהל להתחברות כחלק מהרצת התוכנית (על המנהל לדעת את הסיסמה).
- רשת: המערכת מתחברת לשרת עם סוקט TCP בכתובת הנתונה בפורט שמוגדר ב-SERVER_BIND_PORT. הממשק עצמו נפתח בדפדפן ב-IP מקומי עם פורט שנבחר אוטומטית. בכדי שהמנהל יתקשר עם השרת יצטרך להיות תחת אותו LAN איתו אלא אם כן השרת מוגדר ככתובת WAN.
- ארכיטקטורה מינימלית: דרישות ארכיטקטורה למנהל הן מינימליות ביותר, כל מה שהמנהל צריך להריץ זה דפדפן וקובץ פייתון, ולכן יש לציין שרוב המחשבים המודרניים יצליחו להריץ את חלק המנהל ללא בעיה כלל.

שרת:

- סביבה: המערכת דורשת Python עם ספריית keyboard. בנוסף קובץ encryption.py דורש את הספריות pycryptodome ו-cryptography. (דורש פייתון 3.6 ומעלה).
- כלים: נדרשים כלים מהספרייה המשותפת, כולל protocol.py, encryption.py ו-DB.py, שמיובאים מהתיקייה './shared'.
- מיקום קבצים: התוכנית מצפה שקבצי פרוטוקול, הצפנה ומסד נתונים יהיו בתיקיית shared מעל התיקייה הנוכחית. מסדי הנתונים נשמרים בתיקייה הנוכחית.
- נתונים התחלתיים: המערכת משתמשת בסיסמת ברירת מחדל "itzik" למנהל, אך ניתן לשנות זאת בהפעלה על ידי פרמטרים argv. קיימים גם פרמטרים ברירת מחדל למספר לקוחות מקסימלי (5) ורמת אבטחה (5).
- רשת: השרת מאזין לחיבורים נכנסים עם סוקט TCP ומגדיר timeout של שנייה אחת בין בדיקות. הוא מתקשר עם הלקוחות והמנהלים באמצעות פרוטוקול ייעודי. השרת יוכל להיות מוגדר באיזה רשת שצריך, אין מגבלה עליו כל עוד שאר חלקי הפרויקט מודעים לכתובת שלו. השרת מאזין על פורט קבוע 6734.
- ארכיטקטורה מינימלית: המערכת מבוססת על multi-threading, עם thread נפרד לכל לקוח. גם כן פה המערכת דרושה לארכיטקטורה מינימלית, השרת לא מטפל בהרבה עבודה ולכן רוב המחשבים המודרניים יריצו את השרת ללא בעיה, יש לציין כי קיימת הגבלה של עד 40 לקוחות לא משנה מה אחרת השרת אכן יהיה מוצף עם הודעות ויהיה לו קשה לתפקד.

לקוח:

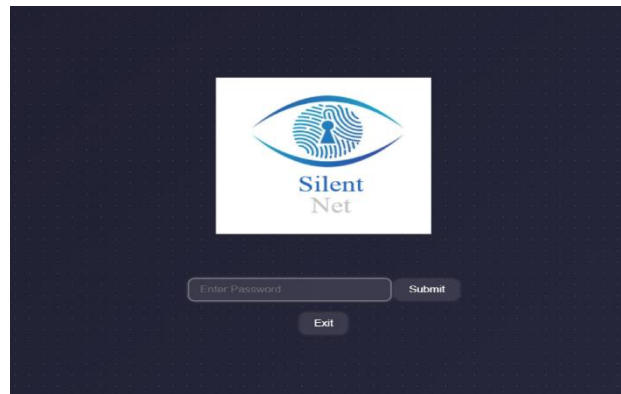
- סביבה: VM ומערכת הפעלה Linux מגרסה 6.11.0-19-generic, בנוסף בשביל הרצה והסרה צריך הרשאות מנהל.
- כלים: לא צריך כלים מיוחדים חוץ מהכלים הבנויים כבר לתוך מערכת ההפעלה, כל הכלים בצד הלקוח הם כלים built-in של מערכת ההפעלה.
- מיקום קבצים: לאחר שהתוכנית עברה קימפול לתוך קובץ 'ko'. אין חשיבות למיקום הקבצים. בשביל לקמפל את התוכנית צריך שהקבצים יהיו לפי עץ הקבצים בתחילת הפרק (רק הקבצים תחת התיקייה employee).
- נתונים התחלתיים: התוכנית משתמשת בנתונים התחלתיים של כתובת IP של השרת ופורט ("10.100.102.103" ו-6734), אם מי שמריץ את התוכנית רוצה לשנות כך יוכל להגדיר זאת בעת הרצת התוכנית ב argv.
- רשת: התוכנית מתחברת לשרת בפרוטוקול TCP ומתחבר אליו עם הנתונים ההתחלתיים, על הלקוח להיות באותו LAN עם השרת אלא אם כן השרת מוגדר ככתובת WAN.
- ארכיטקטורה מינימלית: כיוון שמתעסקים פה עם VM, יש לציין את הארכיטקטורה שצריך להגדיר ל VM. ההגדרות המינימליות שעליהם הפרויקט עובד כמצופה הן 2 ליבות מעבד וכ-2048 MB RAM.

משתמשי המערכת

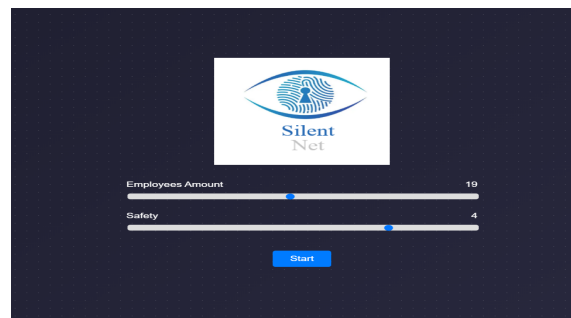
מנהל

הפעלה של המערכת (להריץ מתוך תיקיית manager) - `python manager.py <server_ip>`

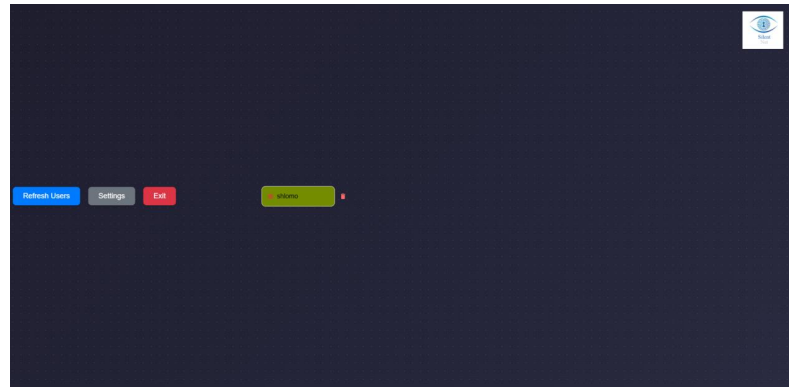
לאחר שהתוכנית תתחיל לרוץ החלק הגרפי יוצג אוטומטית למנהל - מסך ראשוני, המנהל מכניס את הסיסמה הידועה מראש ומתחבר אל השרת, לאחר מכן ילחץ על submit, אם הסיסמה לא נכונה הדף יכתוב על כך, אם ברצונו לסגור את התוכנית יוכל ללחוץ על exit.



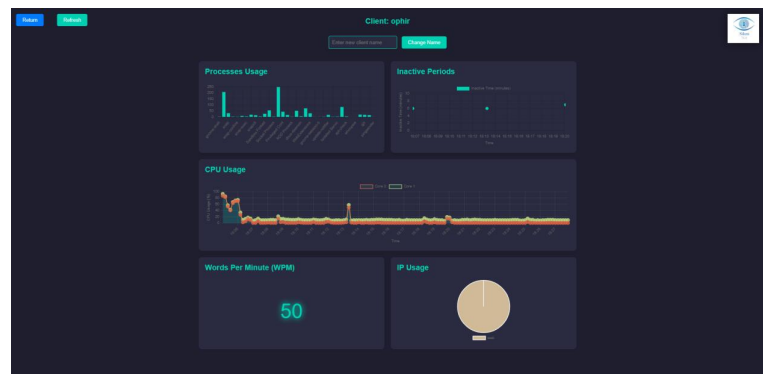
מסך שני, המנהל יכול להגדיר את רמת הבטיחות וכמות הלקוחות המקסימלית על ידי גרירת הנקודה והמספר משתנה בצד ימין. לאחר שסיים לבחור כרצונו ילחץ על start.



מסך שלישי, רשימת העובדים הרשומים במערכת אצל השרת (הצבע על כל עובד זה רמת הפעילות שלו, ככל שהצבע יותר אדום משמע העובד עובד פחות טוב. בנוסף הנקודה על כל שם מייצגת אם מחשב זה כרגע מחובר לשרת, אם לא הנקודה היא אדומה, אם מחובר הנקודה אפורה). הכפתורים משמאל מאפשרים לעדכן את העמוד הנוכחי, לחזור לעמוד ההגדרות ולשנות הגדרות או לסגור את התוכנית לגמרי ולהתנתק. השם של כל לקוח מוגבל במסך זה ל 18 תווים, אם השם של לקוח גדול מאורך זה, השם יופיע בחלקו, ניתן לראות את השם המלא של לקוח על ידי מעבר עם העכבר מעל הכפתור של אותו לקוח, והשם שלו יופיע בחלק העליון של המסך.



מסך רביעי, הנתונים שנאספו על העובד בצורה מסודרת ויפה לעין, בנוסף המנהל יכול לשנות את השם בו שמור העובד לצורכי נוחות ולחזור לדף הקודם או לעדכן את הדף הנוכחי בכפתורים למעלה.



(שאר המסכים הינם מסכי error/התחברות, אין צורך להסביר אותם אשר המידע מוסבר בהם).

שרת

אופן הפעלת המערכת (להריץ מתוך תיקיית server) –

`python server.py <max_clients> <safety> <password>`

(ב args צריך לתת את מספר הלקוחות המקסימליים הדיפולטיביים של השרת, רמת הבטיחות והסיסה שהמנהלים צריכים להכניס).

```
C:\Users\omerk\Desktop\git\SilentNet\src\server>py server.py
Using default configuration values
Usage: python server.py <max_clients:int> <safety:int> <password:str>

Server running with configuration:
Max clients: 5
Safety: 5
Password: itzik

Press 'q' to quit server
Press 'e' to erase all logs
```

הכתוב זה מה שמופיע כל פעם שמריצים את השרת, כמו שניתן לראות בנוסף להוראות הקודמות אם רוצים לסגור את השרת מכל סיבה שהיא ניתן ללחוץ על המקש q ואם רוצים למחוק את כל הנתונים השמורים על לקוחות ניתן ללחוץ על e (לא מוחק את הלקוחות עצמם, אלא רק את הפרטים שלהם, בשביל מחיקה של הלקוח המנהל יכול בעמוד שלו על ידי לחיצה על אייקון אשפה ליד השם של הלקוח).

לקות

אופן הפעלת המערכת (להריץ מתיקייה בה proj.ko נמצא בה), לשים לב שרק אדם עם הרשאות sudo יכול לבצע פעולה זו –

```
sudo insmod proj.ko dAddress="server_ip" dPort=server_port
```

לאחר מכן המערכת מוחבאת, לכן אם צריך להציג אותה – כלומר להוריד אותה מלהיות מוחבאת, צריך ללחוץ על המקלדת את סדר המקשים הבא אחד אחרי השני

הצגה: x-x

הסתרה: h-h

(בדגש על '-' שלא על הלוח מספרים, כלומר מקלדת מרכזית)

השימוש המרכזי יהיה בסדר המקשים הראשון אשר צריכים אותו בשביל להסיר את התוכנית. כלומר לבצע את סדר המקשים ולאחר מכן לכתוב

```
sudo rmmod proj.ko
```

אין צורך בלבצע את ההצגה וההסתרה סתם ככה, אבל בשביל צרכים אחרים שהם אינם הסרת התוכנית כגון בדיקות כאלו ושונות אלא הם סדרי המקשים.

אם צריך לקמפל את התוכנית, חשוב לשים לב שכל הקבצים תחת ל employee נמצאים בסדר הנכון, ולאחר מכן בתיקייה employee לכתוב את הפקודה make מה שייצור את הקובץ proj.ko, אם ברצון למחוק את כל הקבצים שנוצרו אחרי פקודת make ניתן לכתוב make clean מה שימחק את כל הקבצים שנוצרו על ידי הפקודה (גם את הקובץ proj.ko).

רפלקציה

פרויקט זה לראשונה עלה לי לראש בסוף כיתה י"א, כאשר ידעתי שארצה לבנות פרויקט שמעמיק בנושא מערכות הפעלה. בהתחלה התלבטתי מה לבנות, לבסוף החלטתי ללכת לפרויקט זה כיוון שרציתי להתעמק במערכת ההפעלה Linux ולהבין אותה יותר טוב בכל הקשור ל"מאחורי הקלעים" שלה.

לפני שהתחלתי לעבוד על הפרויקט, קראתי שני ספרים בנושא שהעשירו את הידע שלי לגבי מערכת ההפעלה ועזרו לי להבין מושגים שמשמשים בהם במאמרים שונים. התייעצתי גם עם מורה המגמה לקבלת תמונה שלמה יותר של הפרויקט.

בזכות התכנון המוקדם נתקלתי בפחות מכשולים לאורך הדרך (בניגוד לפרויקטים עבר בהם חוסר התכנון המוקדם יצר קושי רב בעת כתיבת הפרויקט), ויכולתי לממש כמעט את כל מה שתכננתי במסמך האפיון. מכאן למדתי שלפני כל פרויקט גדול כדאי לתכנן היטב כל פריט בפרויקט.

למרות זאת, נתקלתי במספר אתגרים משמעותיים:

1. מימוש באפר מעגלי – בשל חוסר תכנון מספק בנושא זה נתקלתי בבעיות שלא הצלחתי להבין את מקורן. לאחר תקופה ארוכה שהייתי תקוע על הנושא, החלטתי שאני נעזר בAI אך גם הוא לא היה ידע את מקור הבעיה. לבסוף החלטתי לתכנן חלק זה מחדש ולכתוב את כולו מאפס עוד פעם, מה שפתר את הבעיה.

2. חסימת המערכת בשל timeout ארוך מדי (socket timeout) - בעיה ייחודית לקוד קרנלי שגרמה לתקיעה מלאה של המערכת. מצאתי פתרון באמצעות מבנה נתונים מובנה בקרנל בשם workqueue, את מבנה נתונים זה הצלחתי למצוא באחד הספרים אותם קראתי, שם מזכירים אותו, לאחר מכן חקרתי עליו באינטרנט והבנתי שהוא מתאים כמו כפפה לבעיה שלי ולכן החלטתי להתאים את הפרויקט מחדש בכדי שאוכל להשתמש במבנה נתונים זה.

נעזרתי בעמיתים לכיתה בעיקר בפיתוח מסד הנתונים. לאחר שראיתי שהקובץ שמחזיק את המידע גדל משמעותית אחרי כל הרצה גם אם זה הרצה קצרה של כמה דקות, התייעצתי עם חבר והחלטנו למספר כל נתון כדי לחסוך זיכרון וזמן חישוב – מצד אחד כמות הזיכרון הנשמרת קטנה בצורה דרסטית ומצד השני הזמן שלוקח לחפש כל נתון גם כן לוקח הרבה פחות כאשר הכל נמצא במקום אחד מאשר לעבור על כל ההודעות שנשלחו אי פעם ממחשב ספציפי.

בראייה לאחור, הייתי משלב שני תהליכים: אחד בקרנל להשגת מידע שדורש הרשאות גבוהות, והשני ב user mode - לניהול התקשורת עם השרת. כך גם הייתי יכול להוסיף הצפנה בין הלקוח לשרת ואף עוד פיצ'רים נוספים בהם השרת מתקשר בחזרה עם הלקוח (דבר שהיה מוסיף סיבוך רב אם הייתי כותב בקוד קרנל, ולכן בחרתי לא לבצע זאת). זוהי הדרך הנכונה יותר לכתוב את הפרויקט שלי, במקום לנסות לכתוב הכול בקרנל כמו שאני עשיתי.

בהינתן יותר זמן, הייתי מרחיב את הפרויקט כך שהשרת יוכל לשלוח מידע חזרה לפי בקשת המנהל, לשינוי קונפיגורציות מרחוק במחשבי העובדים ובכך למנהל תהיה בקרה אולטימטיבית על מחשבי עובדי החברה. בנוסף, הייתי מוסיף התאמה למגוון גרסאות לינוקס ולא רק לגרסה עליה אני כתבתי.

מהפרויקט הרווחתי ידע ייחודי בתחום מערכות ההפעלה. למרות שהיום ניתן לשאול AI על כמעט כל נושא, הלמידה העצמאית באמצעות ספרים ומאמרים והפיתוח העצמי תרמו להבנה עמוקה יותר של התחום. חקירת קוד של מערכת הפעלה הקפיצה את ההבנה שלי במספר מדרגות. אני מודה לכל מי שסייע לי בדרך: חברים לכיתה איתם התייעצתי, המורה אופיר שביט שעזר בהבנה כללית של פיתוח הפרויקט, ואחי שמבין בתחום וייעץ לי לאורך הדרך.

ביבליוגרפיה

קורבט, ג'., רוביני, א., וקרואה-הארטמן, ג. (2005). Linux Device Drivers (מהדורה שלישית). O'Reilly Media. <https://bootlin.com/doc/books/ldd3.pdf>.

לוב, ר. (2010). Linux Kernel Development (מהדורה שלישית). Pearson Education. <https://www.doc-developpement-durable.org/file/Projets-informatiques/cours-&-manuels-informatiques/Linux/Linux%20Kernel%20Development,%203rd%20Edition.pdf>

קניסטון, ג'. פנצ'מוקי, פ' ס', והיראמצו, מ'. (2007). Linux kernel. Kernel Probes (Kprobes). docs. <https://docs.kernel.org/trace/kprobes.html>.

איתמר, מ. (2021). בניית rootkit KLM בלינוקס (חלק ב'). Digital Whisper. <https://www.digitalwhisper.co.il/files/Zines/0x7D/DW125-2-LinuxRootkit-Part2.pdf>

הערות

לגבי שני הספרים הראשונים, כמובן שלא כל תוכן הספרים רלוונטי ולא הייתי צריך להיעזר בכולו בעת כתיבת הפרויקט, לכן אציין את הפרקים/נושאים בהם נעזרתי ספציפית בעת כתיבה. המידע שנמצא שם תרם לי רבות בהבנת הנושא (מודולים בקרנל) ולכן חשוב לאזכר את הנושאים בהם נעזרתי.

– Linux Device Drivers

פרקים/נושאים:

פרק שני - Building and Running Modules, הפרק עוסק בהרצה ראשונית של מודולים.

פרק שביעי - Time, Delays, and Deferred Work, הפרק עוסק בכלים שונים שהקרנל מציע בכדי להתמודד עם בעיות שונות, אחת הבעיות בפרויקט שלי שהפרק מדבר עליו הוא בעית תזמון משימות. לכן הנושא הספציפי מתוך הפרק בו נעזרתי בעת כתיבה הוא Workqueues (עמוד 205).

– Linux Kernel Development

פרקים/נושאים:

פרק עשירי - Kernel Synchronization Methods, הפרק עוסק באמצעי סנכרון שונים של הקרנל. מתוך הפרק נעזרתי בנושא Atomic Operations (עמוד 175) ו - Mutexes (עמוד 195).

נספחים

קוד הפרויקט