

**שם התלמיד: נועם אזולאי**

**ת"ז: 206824534**

**מנחים: מיכאל צ'רנובילסקי וגד רוזנטל**

**בית הספר: אורט גוטמן נתניה**

**כיתה: י"ב 7**

**אפריל 2016**

**תוכן עניינים**

[תקציר כולל ורציונל הפרויקט 3](#_Toc418019495)

[מבוא ורקע כללי 4](#_Toc418019496)

[תיאור המוצר המוגמר 6](#_Toc418019497)

[שפת התכנות וסביבת העבודה 7](#_Toc418019498)

[ניסוח וניתוח הבעיה האלגוריתמית 8](#_Toc418019499)

[תיאור אלגוריתמים קיימים 9](#_Toc418019500)

[הפתרון הנבחר 11](#_Toc418019501)

[פיתוח הפתרון בשכלול הקוד עם שפת התכנות 15](#_Toc418019502)

[תיאור המודולים המרכיבים את המוצר 23](#_Toc418019503)

[פירוט מבני הנתונים 26](#_Toc418019504)

[פירוט מאגרי המידע 27](#_Toc418019505)

[השוואת העבודה עם פתרונות ויישומים קיימים 29](#_Toc418019506)

[מדריך התקנה למשתמש 31](#_Toc418019507)

[תיאור הממשק למשתמש 32](#_Toc418019508)

[מבט אישי על העבודה ותהליך הפיתוח 35](#_Toc418019509)

[ביבליוגרפיה 37](#_Toc418019510)

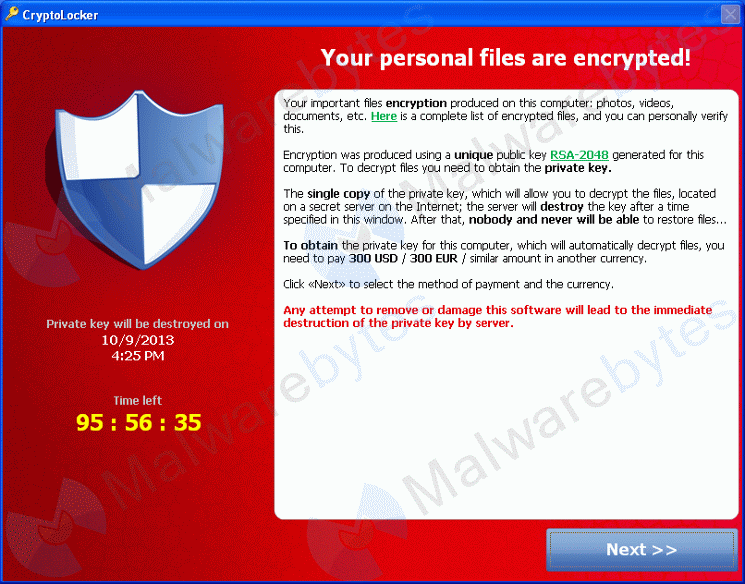
[קוד התוכנה 38](#_Toc418019511)

# תקציר כולל ורציונל הפרויקט

וירוס הRansomware הוא וירוס שמטרתו היא רווחית, הווירוס פועל ע"י הצפנת המידע במחשב, ולאחר מכן הצגת הודעה בעלת התראה שהמחשב הוצפן, וכדי להחזירו לקדמותו יש לשלם סכום מסוים (כופר) ומכאן שמו.  
הווירוס יצפין קבצים עם סיומות מסוימות (לדוגמא txt, בדרך כלל קבצים טקסטואליים כי אלו הם אשר מכילים מידע חשוב, ובדרך כלל עם הצפנת RSA) ולאחר מכן יבקש דמי גבייה על שירות הDecryption, ויקציב זמן מסוים עד מחיקה מלאה של המידע.

המוטיבציה לפיתוח תוכנה שתתמודד עם הבעיה שתוארה לעיל מגיעה מהמקום של הרצון ליצור פיתרון שונה משאר הפתרונות של המניעה לוירוס הזה, כאשר כיום תוכנות האנטי וירוס בעצם משוות את הHash של התהליכים הרצים לHashים ממאגר מידע של האנטי וירוס אשר מתעדכן לאחר גילויים שונים. ולכן רציתי לפתח כלי שבעצם לא יהיה תלוי במאגר אלה כלי שיכול לזהות בעצמו וירוס שכזה על פי ההתנהגות שלו.

התוכנה אמורה כביכול לבדוק כל תהליך אשר ברצונה לקרוא מידע מקובץ, כאשר יוצבו במקומות בעלי חשיבות (לדוגמא כונן C בתיקייה ששמה a יוצבו כמות מסוימת קבצי טקסט שמטרתם תהיה מלכודת לווירוס מסוג שכזה, ובמידה שהקבצים ישונו בצורה דרסטית ובמהירות לא אנושית, יהיה חשד לווירוס, ובמצב שכזה תהליך הווירוס יופסק, ותוצג הודעה על המסך שבה ייכתב שיש חשד לווירוס, והמשתמש יושאל האם ברצונו להמשיך בתוכנית או לא, משום שמצב כזה יכול להיות גם אצל מתכנים.



# מבוא ורקע כללי

הפרוייקט שלי בעצם מביא הגנה ברמה מסויימת **מוירוס הגבייה (Ransomware)**. כאשר

הפרויקט שלי מבוסס על **שליטה מרחוק** (**Remote Control**). בין המחשבים ברשת נוצר חיבור עם השרת המרכזי, וחיבור זה מאפשר שליטה מרחוק עליהם (שליטה מוגבלת, בהתאם לאפשרויות שהתוכנה מספקת). השליטה מרחוק מאפשרת איסוף מידע ממחשבי הקצה והרצת פקודות עליהם (כגון סגירת תהליכים), וזהו הבסיס לפרויקט כולו.

הנושא המרכזי של הפרויקט הוא **ניתוח משאבי מערכת**. למערכת מחשב יש סוגים שונים של משאבים, כגון: זמן מעבד, זיכרון פיזי, דיסק קשיח, תעבורת רשת ועוד. את המשאבים חולקים ביניהם כל התהליכים הרצים במערכת.

הפרויקט שלי עוסק בניתוח של נתוני השימוש במשאבי המערכת. הן של נתוני השימוש הכולל במשאבי המערכת, והן של נתוני שימוש של תהליכים נפרדים במשאבי המערכת. מטרת ניתוח הנתונים היא לזהות תהליכים שגוזלים יותר מדי משאבי מערכת מיתר התהליכים ובכך מאטים את המחשב, ולפעול בהתאם למניעת מצבים כאלו.

לפרויקט יש מספר מרכיבים עיקריים ולהלן סקירת רקע לגביהם:

* מרכיב עיקרי בפרויקט שלי הוא **תקשורת**. תקשורת בעולם המחשבים מבוססת על Sockets, שהם נקודות קצה עבור זרם נתונים בתקשורת בין תהליכים על גבי רשת מחשבים. רוב הסוקטים כיום מבוססים על ה- Internet Protocol (IP).

בפרויקט שלי התקשורת בין השרת ללקוחות מתבצעת באמצעות Internet Sockets, כאשר בשכבת התעבורה יש שימוש בפרוטוקול ה- TCP, שייחודו בכך שהוא מוודא שכל הנתונים שנשלחו התקבלו ובסדר הנכון, ואחרת שולח אותם שוב. בפרויקט שלי יש חשיבות לכך שלא יהיה איבוד מידע ופרוטוקול ה- TCP/IPמתאים בדיוק למטרה זו.

* מרכיב חשוב נוסף בפרויקט הוא **קבלת מידע לגבי תהליכים**. המודול האחראי על קבלת המידע לגבי התהליכים הרצים במחשב משתמש ב- WMI. הוא פונה אליו באמצעות שאילתות SQL ומקבל מידע מפורט לגבי כל התהליכים שרצים במחשב, וגם מידע לגבי השימוש הכולל בזיכרון, בזמן מעבד ופעולות I/O.

כדי להבין איך זה עובד עלינו להבין ראשית מהו ה- WMI. ה- WMI הוא המימוש של Microsoft ל- Web-Based Enterprise Management, שזו יוזמה של התעשייה לפתח תקן של גישה למידע ניהולי בסביבה ארגונית. כלומר ה- WMI הוא ממשק של מערכת ההפעלה שמאפשר לקבל מידע לגבי רכיביה. בעזרתו ניתן לקבל מידע לגבי רכיבי המערכת, החומרה והתוכנה (התהליכים שרצים).

* בנוסף, הפרויקט משלב עבודה עם **מסד נתונים**. הנתונים שהתוכנה אוספת לגבי כל אחד ממחשבי הרשת נשמרים במסד הנתונים, שהעבודה איתו היא בטכנולוגיית ה- SQLite, שמממשת את רוב הסטנדרט של SQL (שהיא השפה המקובלת לטיפול ועיבוד מידע בבסיסי נתונים). להבדיל מרוב המערכות לניהול בסיסי נתונים, SQLite אינה תהליך עצמאי נפרד, אלא ספריה בתוך התהליך באופן אינטגרלי. ‏ זוהי בחירה פופולרית כאשר מפתחים בסיס נתונים מוטמע עבור צד לקוח כשירות ניהול מידע מקומי. מסד הנתונים שאותו מנהלת התוכנה שלי שומר מידע לשימוש מקומי בלבד, ולכן בחרתי להשתמש ב- SQLite שלא דורש הרצה של SQL Server.
* כמו כן, בפרויקט ישנה עבודה עם **קובץ הגדרות**. התוכנה מאפשרת למנהל הרשת לקבוע הגדרות שיכתיבו את אופן פעולתה בתרחישים השונים, כאשר ההגדרות נשמרות בקובץ XML.

ה- XML הוא תקן לייצוג טקסטואלי של נתונים במחשבים. ב- XML טבעי מאוד לסדר מידע באופן היררכי (עץ). הייצוג עצמו מתבצע תוך שימוש בתגיות (Tags) שמגדירות רכיבי מידע (Elements). בתוך רכיבי המידע ניתן לאכסן את המידע עצמו או רכיבים נוספים. קובץ ה- XML בפרויקט שלי מכיל את ההגדרות שהן אוסף כללים המכילים תתי-כללים במבנה היררכי.

# תיאור המוצר המוגמר

שם המוצר: Anti-Ransomware.

המוצר אמור לבצע ניטור ובקרה על תהליכים אשר מתנהגים חשוד, ולטפל בהם בהתאם להוראות המשתמש (הוספה לBlakclist/Whitelist).

המטרה המרכזית של המוצר היא בעצם לנסות ולגלות פעילות חריגה אשר יכולה להיות חלק מההשפעות של הRansomware, המוצר בעצם יידע את המשתמש מה ברצונו לעשות עם התהליך במידה והוא יהיה חשוד כRansomware.  
האם להוסיף את התהליך לBlacklist/Whitelist כאשר תהליך יתווסף לWhitelist לא יישאל לגביו יותר, וכאשר הוא יהיה בBlacklist הוא יימחק מן המחשב וכאשר יורץ ייסגר באופן מיידי.

השאלות שהמערכת עונה עליהן:

1. **כיצד להגן על המידע הרגיש מפני גורמים לא מורשים?** התוכנה מצפינה את כל התקשורת בין מחשבי הרשת לשרת, כך שגורם לא מורשה לא יוכל להפיק שום תועלת מהאזנה למידע הנשלח, הוא לא יוכל לגלות אילו תהליכים רצים על מחשבי החברה וכמה משאבים הם צורכים.
2. **כיצד לעבוד בו-זמנית עם כל המחשבים ברשת?** כאשר מחשבי הרשת מתחברים לשרת, הוא פותח Thread נפרד לכל אחד ומסוגל לתקשר במקביל עם כולם, לקבל מהם מידע ולשלוח להם פקודות, ולכל מחשב ברשת יש בצד השרת מסד נתונים נפרד שבו מאוחסן כל המידע שנאסף עליו.
3. **כיצד להחליט באילו פעולות יש לנקוט בתרחישים הבעייתיים השונים?** התוכנה מאפשרת למנהל הרשת לקבוע את כל ההגדרות שקובעות את אופן פעולתה, ובמסגרת ההגדרות ניתן למשל לקבוע whitelist של תהליכים שהתוכנה לא תתערב בפעולתם, תאפשר להם לצרוך כמה משאבים שהם צריכים ולא תשלח התראות לגביהם.
4. **מה לעשות במקרה שמחשב מסוים מפסיק להגיב לחלוטין?** כאשר מחשב כלשהו נתקע ומפסיק להגיב ולתקשר עם השרת, המערכת מנתקת את התקשורת איתו ושולחת באופן מיידי התראה על הבעיה למנהל הרשת ול- IT כדי שיוכלו להתייחס אליה (לגשת באופן פיזי למחשב ולטפל בו).

# שפת התכנות וסביבת העבודה

הפרויקט נכתב ב- Python וב- C#. כל הליבה של הפרויקט נכתבה ב- Python בשל הפשטות והנוחות של השפה, וה- GUI של הפרויקט נכתב ב- C# כי היה נוח מאוד לכתוב GUI ב- C# בעזרת ה- Visual Studio ואז לחבר אותו לליבה שנכתבה ב- Python בעזרת pipes.

סביבת העבודה עם Python הייתה בתחילה IDLE אך מהר מאוד עברתי ל- PyCharm שהיא תוכנה הרבה יותר מקצועית ונוחה לעבודה על פרויקטים מרובי מודולים וקבצים. היא שימשה גם לפיתוח הקוד ועבודה במקביל עם כל המודולים השונים, וגם להרצתו בשביל בדיקות.

סביבת העבודה עם C# הייתה Microsoft Visual Studio 2010 ששימשה לכתיבת ה- GUI באמצעות גרירת פקדים מה- toolbox, וה- debugger המובנה שימש לבדיקות. הבחירה בסביבת עבודה זו הייתה משום שה- Visual Designer המובנה היה פתרון מאוד נוח לכתיבת ה- GUI של הפרויקט, ובאופן כללי ה- Visual Studio הוא IDE טוב מאוד לעבודה ב- C#.

סביבת העבודה עם מסד הנתונים הייתה SQLite, שהיא ספרייה מוטמעת (כחלק בלתי נפרד) בתוך התהליך של התוכנה, שמאפשרת את ניהול מסדי הנתונים המקומיים בצד השרת.



# ניסוח וניתוח הבעיה האלגוריתמית

במהלך פיתוח המוצר נתקלתי בבעיות אלגוריתמיות רבות:

* **מימוש שרת מולטי-קליינט:**

הבעיה הראשון שעמדה לפניי בפיתוח המוצר היא יצירת שרת מולטי-קליינט. כלומר, הייתי צריך לממש שרת שמתחברים אליו מספר לקוחות בו-זמנית והוא מסוגל לדבר עם כולם. בתוך בעיה אלגוריתמית זו ישנן מספר תתי-בעיות:

* **פיתוח פרוטוקול משלי:** השרת צריך לדבר עם הלקוחות בפרוטוקול מסוים, שמגדיר את אופן התנהלות התקשורת ביניהם. הייתי צריך לפתח פרוטוקול משלי שבו ידבר השרת עם הלקוחות שמתחברים אליו.
* **מימוש אבטחת מידע:** כדי למנוע מהמידע לעבור בין השרת ללקוחות באופן חשוף הייתי צריך לממש אבטחת מידע, כלומר להוסיף לפרוטוקול שלי הצפנה. הבעיה האלגוריתמית היא שליחה של מידע מוצפן בפרוטוקול ופענוח שלו בצד השני של הקו.
* **חיבור עם ה- GUI :** השרת צריך להתחבר ל- GUI לשם הצגת מידע והתראות למנהל באופן גרפי ובשביל לאפשר למנהל לצפות בהגדרות התוכנה ולשנות אותן.
* **השגת מידע על כל התהליכים במחשב:**

בעיה מרכזית בפיתוח הפרויקט הייתה להבין כיצד להשיג את כל המידע הנחוץ לי על התהליכים שרצים במחשב. הבעיה שניצבה בפניי היא כיצד אני בכלל מקבל רשימה של כל התהליכים שרצים במחשב ברגע נתון וכיצד אני מקבל מידע מגוון לגביהם, כגון: מספר מזהה (PID), שימוש במעבד, שימוש בזיכרון, מספר ה- Threads וכו'.

* **הריגת תהליכים וכיבוי המחשב:**

בעיה נוספת הייתה כיצד להרוג תהליכים שרצים במחשב וכיצד לכבות את המחשב (כתגובה לפעילות חריגה שהתוכנה זיהתה). הייתי צריך לפתח קוד שהורג תהליך (או עץ תהליך) לפי המספר המזהה (PID) שלו. בנוסף, הייתי צריך לפתח קוד שמכבה את המחשב (Shutdown).

* **שמירת המידע על התהליכים:**

את המידע שאני משיג על כל התהליכים במחשב אני רוצה לשמור, וזו בעיה אלגוריתמית נוספת שעמדה לפניי – כיצד לשמור את המידע הרב המתקבל על כל הנתונים של כל התהליכים שרצים בכל המחשבים שמחוברים לשרת?

* **אפשרות לקביעת הגדרות לתוכנה:**

רציתי לאפשר למנהל הרשת לקבוע בעצמו את הגדרות התוכנה שקובעות באילו תרחישים היא נכנסת לפעולה ובאיזו פעולה היא נוקטת, ועל כן ניצבתי בפני עוד בעיה אלגוריתמית – כיצד ליצור הגדרות ניתנות לשינוי עבור אופן הפעולה של התוכנה?

# תיאור אלגוריתמים קיימים

* **מימוש שרת מולטי-קליינט:**

המימוש המקובל של תקשורת בין מחשבים הוא באמצעות Sockets, ואולם ישנם מספר סוגים:

* Raw Socket – אין שכבת תעבורה (Transport), זהו Socket בסיסי שפועל בשכבת ה- Internet ויש גישה ישירה ל- packet headers.
* Datagram Socket – מכונה גם connectionless, משתמש בפרוטוקול UDP בשכבת התעבורה ומשמש להעברת מידע במהירות רבה תוך ויתור על הבטחת data integrity.
* Stream Socket – מכונה גם connection-oriented, משתמש בפרוטוקול TCP בשכבת התעבורה ומשמש להעברת מידע תוך וידוא שכל הפקטות מגיעות ליעדן ומגיעות בסדר הנכון.

כמו כן, למימוש השרת כמולטי-קליינט שתומך בתקשורת עם מספר לקוחות בו-זמנית יש שתי שיטות המוכרות לי:

* באמצעות ספריית select – ב- Python יש ספרייה בשם select שבתוכה יש פונקציה עם אותו שם, והיא יכולה לקבל רשימה של Sockets שמהם נרצה לקבל מידע ורשימה של Sockets שנרצה לשלוח דרכם מידע, והיא מחזירה רשימה של Sockets שמוכנים לקריאת מידע מתוכם ורשימה של Sockets שמוכנים לשליחת מידע דרכם לצד השני.
* באמצעות multi-threading – יהיה Socket מאזין שיאזין ללקוחות חדשים שמתחברים ב- Thread הראשי (בלולאה אינסופית), ובכל פעם שמתחבר לקוח חדש ייפתח עבורו Thread חדש ובו תתנהל התקשורת עם לקוח זה. בצורה כזו השרת ידבר עם כל לקוח ב- Thread נפרד וכך יוכל לדבר עם כולם בו-זמנית. ספריית threading של Python מאפשרת לעשות זאת.

לשם אבטחת מידע קיימים אלגוריתמי הצפנה רבים, למשל בהצפנה אסימטרית יש את El-Gamal ואת RSA, ובהצפנה סימטרית יש DES ו- AES (וכמובן עוד רבים אחרים).

* **השגת מידע על כל התהליכים במחשב:**

ישנם מספר ממשקים המאפשרים לקבל מידע על רכיבי החומרה והתוכנה (ועל תהליכים) במחשב (במערכות מבוססות Windows):

* אחד מהם למשל הוא מרחב השמות System.Diagnostics ב- .NET Framework שמכיל אובייקטים כגון Process ו- Performance Counter שמאפשרים לקבל מידע עם תהליכים ומידע כללי על Performance ושימוש במשאבי המחשב.
* ממשק נוסף הוא ה- WMI (Windows Management Instrumentation) שאליו ניתן להתחבר ולהריץ שאילתות SQL כדי לקבל את המידע שרוצים לגבי רכיבי החומרה או לגבי התהליכים הרצים במחשב, כפי שכבר הוסבר בפרק המבוא.
* **שמירת המידע על התהליכים:**
* ראשית, ניתן לשמור את המידע בתוכנה עצמה (כלומר ב- RAM), אולם אז המידע לא יישמר בדיסק ולא יהיה זמין ברגע שהתוכנה תיכבה, ובנוסף כמות הזיכרון ב- RAM מוגבלת.
* שנית, ניתן לכתוב את המידע לתוך קובץ טקסט רגיל. אולם אז צריך לממש באופן עצמאי את כל הפעולות – הכתיבה לקובץ, אחזור המידע מן הקובץ, מחיקה וכו'. וכנראה שהפעולות הללו לא יהיו יעילות הן מבחינת הזמן שהן לוקחות והן מבחינת נפח האחסון שהקובץ יתפוס בדיסק.
* כמו כן, ניתן לשמור את המידע במסד נתונים, שמתאים לכמויות גדולות של מידע ואשר הגישה אליו (הכנסה, אחזור, מחיקה) מתבצעת בקלות באמצעות שפת SQL.

אך גם כאן ישנן מספר אפשרויות, יש כל מיני Database Servers שמאפשרים לעבוד עם מסדי נתונים, ביניהם: MySQL, Microsoft SQL Server, Oracle ועוד.

בנוסף, קיימת טכנולוגיית SQLite שאינה מהווה Database Server של ממש אלא ספרייה מוטמעת בקוד שמאפשרת לעבוד בקלות עם מסדי נתונים מקומיים.

* **אפשרות לקביעת הגדרות לתוכנה:**

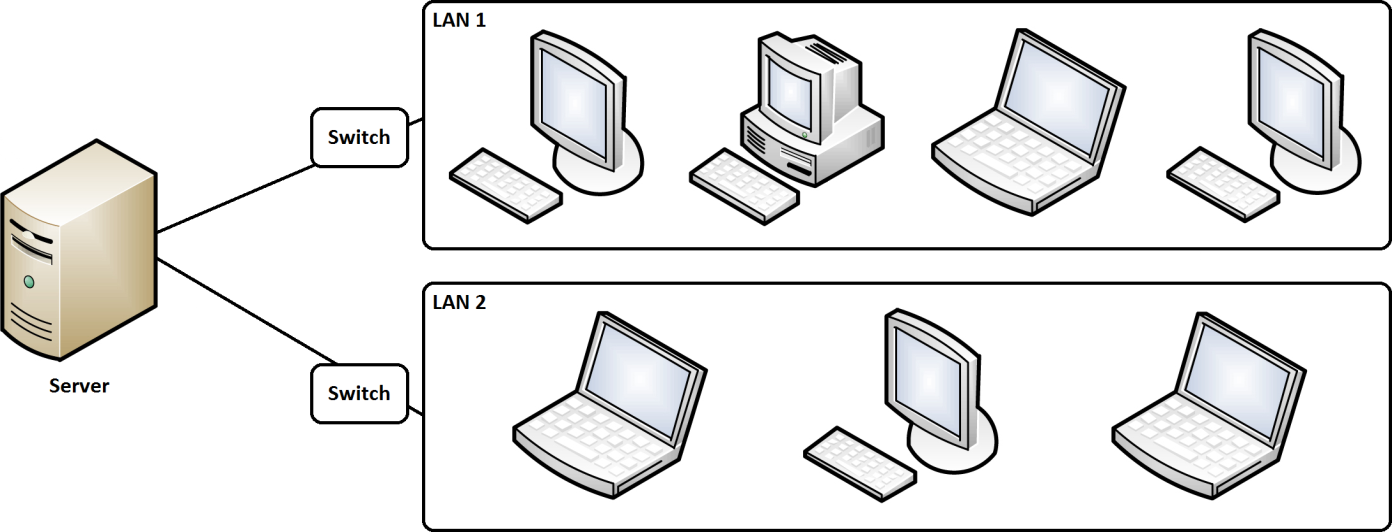
גם כאן ישנן מספר אפשרויות לגבי אופן שמירת ההגדרות:

* ראשית, ניתן לשמור את ההגדרות בתוכנה עצמה בלבד. אולם אז ההגדרות יתאפסו ויחזרו לברירת המחדל בכל פעם שנפעיל מחדש את התוכנה.
* שנית, ניתן לכתוב את המידע לתוך קובץ טקסט שיהיה מאוחסן בדיסק. יש רק פורמט אחד שמוכר לי שמתאים במיוחד למטרה זו, והוא פורמט ה- XML שנמצא בשימוש בכל מקום. פורמט ה- XML מורכב מתגיות שניתן לתת להן שמות כמעט שרירותיים, שיש להן תכונות (attributes) וערכים (values) ובתוכן יכולות להיות תגיות נוספות, כאשר בסופו של דבר מדובר בקובץ טקסט "רגיל" וקריא לחלוטין, שקל ונוח מאוד לעבוד איתו.

# הפתרון הנבחר

* **מימוש שרת מולטי-קליינט:**

את השרת מולטי-קליינט מימשתי באמצעות Stream Socket (בפרוטוקול TCP) שמאזין ב- Thread הראשי, ובכל פעם שמתחבר לקוח, נפתח ללקוח הזה Thread נפרד משלו ונפתח Socket שדרכו מתבצעת כל התקשורת עם לקוח זה. בצורה כזאת השרת מסוגל לדבר בו-זמנית עם מספר לקוחות בת'רדים נפרדים וגם להמשיך להאזין לקבלת חיבורים חדשים. התרשים הבא מסכם את המימוש של השרת המרכזי ואופן הטמעתו ברשת הפנימית:

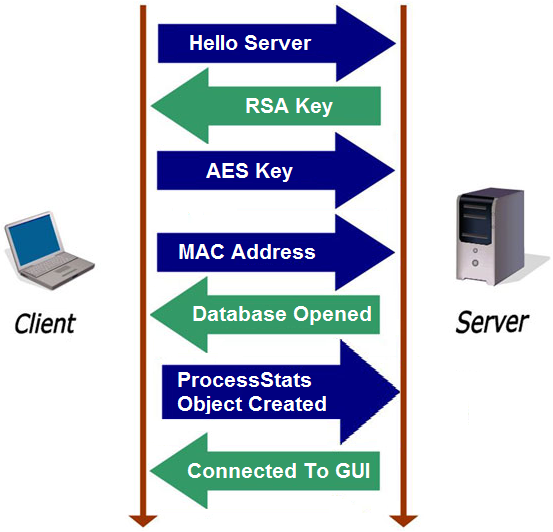


* **פיתוח פרוטוקול משלי ומימוש אבטחת מידע:** פיתחתי פרוטוקול משלי בשביל התקשורת בין השרת ללקוח. בפרוטוקול יש שני רצפי תווים מיוחדים: ### מציין סוף הודעה, ו- $$$ משמש להפרדה בין שני חלקים שונים של ההודעה (וכן מפריד בין ההודעה עצמה לבין ה- hash שלה).

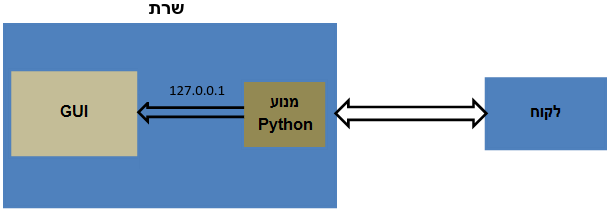
התקשורת בפרוטוקול משתמשת בהצפנה, כאשר תהליך החלפת המפתחות בתחילת התקשורת בין השרת ללקוח מתנהל כך: תחילה השרת שולח ללקוח את מפתח ה- RSA (הפומבי) שלו שמשמש את הלקוח לשם הצפנת מפתח ההצפנה הסימטרית. הלקוח מצפין בעזרת מפתח ה- RSA של השרת את מפתח ה- AES (מפתח הצפנה סימטרית) שלו ושולח אותו לשרת. מאותו רגע ואילך שניהם משתמשים בהצפנה סימטרית (אלגוריתם AES) לתקשורת. בנוסף, לכל הודעה מוצפנת שעוברת בפרוטוקול מתווסף hash שלה שמאפשר בדיקת תקינות, והצד השני מוודא שה- hash אכן תואם את תוכן ההודעה.

כמו כן, בפרוטוקול ישנם שלבים קבועים שמתארים את האופן בו מתנהלת תחילת התקשורת בין השרת ללקוח. אלו הם שלבי-קדם שהשרת והלקוח מבצעים בתור הכנה לפני שמגיעים לחלק העיקרי (שהוא העברת נתוני השימוש של תהליכים במשאבי המערכת מהלקוח לשרת והעברת פקודות לביצוע מהשרת ללקוח), ותפקידם לאפשר תקשורת מוצפנת ולוודא שכל הרכיבים הדרושים לשם העברת הנתונים ושמירתם עובדים.

ניתן לראות את כל השלבים בתרשים הבא:



* **חיבור עם ה- GUI :** החיבור מתבצע באמצעות Socket מקומי כמתואר בתרשים הבא:



* **השגת מידע על כל התהליכים במחשב:**

כדי להשיג את כל המידע הנחוץ לי על התהליכים שרצים במחשב השתמשתי ב- WMI (שהסברתי לגביו כבר קודם לכן). אני מתחבר ל- WMI (באמצעות win32com.client) ולאחר מכן אני מקבל ממנו את המידע שאני צריך באמצעות שאילתות SQL מתאימות.

תחילה אני מריץ שאילתות כלליות כדי לגלות כמה זיכרון RAM פיזי מותקן במחשב בסה"כ, וכמה ליבות (לוגיות) יש למעבד. לאחר מכן, בכל מחזור של קבלת מידע על מצב המערכת (בכל קריאה לפונקציית GetStats()) אני קודם כל מריץ שאילתות כדי לקבל את אחוז השימוש הכולל ב- CPU ואת כמות הזיכרון הפיזי הפנוי בסה"כ, ולאחר מכן אני מריץ שאילתא כדי לקבל את כל הנתונים הרצויים על כל התהליכים. נתוני התהליך שהחלטתי שהם רלוונטיים לי הם:

|  |  |
| --- | --- |
| **שם** | **תפקיד** |
| Name | שם התהליך |
| IDProcess | מספר מזהה (PID) |
| CreatingProcessID | מספר מזהה של התהליך שיצר אותו |
| ElapsedTime | כמות הזמן (בשניות) שהתהליך רץ במחשב |
| PercentProcessorTime | אחוז השימוש במעבד |
| WorkingSetPrivate | שימוש בזיכרון בבייטים |
| IODataBytesPersec | כמות הבייטים לשנייה שעברו בפעולות I/O |
| IODataOperationsPersec | כמות פעולות I/O לשנייה |
| ThreadCount | כמות ה- Threads של התהליך |
| TimeStamp | חותמת זמן שמציינת מתי נאסף המידע |

* **הריגת תהליכים וכיבוי המחשב:**

בעיה זו התגלתה כפשוטה למדי. בשביל להרוג תהליכים נעזרתי במודול win32api. הוא מכיל פונקציה בשם OpenProcess שעל סמך ה- PID של תהליך מחזירה handle אליו. בעזרת ה- handle ניתן לקרוא לפונקציית TerminateProcess שהורגת את התהליך.

לשם כיבוי המחשב נעזרתי במודול המובנה os. יש בתוכו פונקציה בשם system שמאפשרת להריץ פקודות בדיוק באותה צורה כמו ב- cmd. בצורה כזו, כל מה שצריך הוא להריץ פקודת *shutdown /s /t [timeout] /c [comment]* והמחשב יציג למשתמש חלון popup עם ההודעה *comment* (שמסבירה את הסיבה לכיבוי המחשב) וייכבה לאחר *timeout* שניות.

* **שמירת המידע על התהליכים:**

מכיוון שהמידע המצטבר על כל התהליכים שרצים במחשב הוא רב, הגעתי למסקנה שכדאי לשמור אותו במסד נתונים. כדי לאפשר לת'רדים השונים לשמור מידע כל אחד על הלקוח שלו בו-זמנית מבלי שיעכבו אחד את השני, החלטתי לפתוח מסד נתונים נפרד לכל לקוח.

מכיוון שמדובר במסדי נתונים מקומיים בחרתי לעבוד איתם בעזרת sqlite שמוטמע בקוד ומאפשר עבודה פשוטה ונוחה עם מסד הנתונים ולא דורש התקנת SQL Server.

כתבתי מחלקה משלי שפותחת מסד נתונים ומאפשרת לכתוב לתוכו, לקרוא ממנו וכן מוחקת באופן אוטומטי מידע ישן (כדי להגביל את נפח האחסון שמסד הנתונים תופס). כל העבודה של השרת מול מסדי הנתונים מתבצעת באמצעות מחלקה זו.

* **אפשרות לקביעת הגדרות לתוכנה:**

כמקובל בעולם כיום, החלטתי ליצור קובץ הגדרות בפורמט XML, שמתאים בדיוק למטרה זו בשל מבנהו ההיררכי שמאפשר לתאר בקלות הגדרות וכללים. קבעתי מבנה מאוד ספציפי למסמך ה- XML שמתאר את ההגדרות כמורכבות מכללים שמכילים תתי-כללים. בשביל פירוט על מבנה הקובץ, ראה פירוט מאגרי המידע. בנוסף, החלטתי להוסיף קובץ XML נוסף המכיל הגדרות ברירת מחדל שאיתן באה התוכנה ואליהן ניתן לשחזר את ההגדרות בכל עת.

קובץ ההגדרות משמש גם את ה- GUI וגם את מנוע ה- Python:

* ה- GUI קורא מקובץ ההגדרות כדי להציג את ההגדרות הנוכחיות למנהל, וכותב לתוכו כאשר המנהל שומר את השינויים שעשה בהגדרות.
* מנוע ה- Python ניגש לקובץ ההגדרות בכל פעם שהוא מנתח נתונים, כדי לטעון מהקובץ את הכללים ועל פיהם לנתח את הנתונים ולהחליט על הפעולות לביצוע.

# פיתוח הפתרון בשכלול הקוד עם שפת התכנות

* **מימוש שרת מולטי-קליינט:**

הבסיס לשרת המולטי-קליינט הוא:

listening\_socket = socket.socket()  
listening\_socket.bind(ADDR)  
listening\_socket.listen(5) **while** True:  
 clientSock, (ip, port) = listening\_socket.accept()  
 newThread = ClientThread(ip, clientSock)  
 newThread.start()

ה- listening\_socket מאזין ובכל פעם שמתחבר לקוח חדש הוא יוצר בשבילו socket ואני פותח בשבילו Thread חדש בתוך אובייקט ClientThread שמאפשר את המשך העבודה מול הלקוח.

* **פיתוח פרוטוקול משלי ומימוש אבטחת מידע:**

ראשית קבעתי את גודל ה- buffer ושני התווים המיוחדים, התו המפריד והתו שמסיים הודעה:

BUFF = 1024END\_OF\_MSG = **'###'**SEPARATOR = **'$$$'**

כתבתי פונקציות לשליחה וקבלה של מידע בהתאם לפרוטוקול שלי. יש פונקציות לשליחה וקבלה של מידע ללא הצפנה:

**def** Send(sock, msg, msg\_complete=True):

*"""  
 Adds padding to the end of the message (if indicated so) and sends it via the socket.* **:param** *sock: the socket which is used to send the data.* **:param** *msg: the message to be sent.* **:param** *msg\_complete: indicates whether the message is complete and should be padded.* **:return***: True if sending procedure was successful, False otherwise.  
 """* **if** msg\_complete:  
 msg += END\_OF\_MSG  
 **try**:  
 sock.sendall(msg)  
 **return** True  
 **except** socket.error:  
 **return** False

**def** Receive(sock, buffer=BUFF):

*"""  
 Receives all the data from the socket by chunks of buffer size at a time.  
 Removes the padding at the end of the message.* **:param** *sock: the socket which is used to receive the data.* **:param** *buffer: the amount of data to receive at a time.* **:return***: the message received, or None if receive was unsuccessful.  
 """* msg = **''  
 try**:  
 data = sock.recv(buffer)  
 **except** socket.error:  
 **return** None  
  
 **while** data:  
 **if** data[-len(END\_OF\_MSG):] == END\_OF\_MSG: *# we reached the end of the message, so remove padding and finish* msg += data  
 **break  
 else**:  
 msg += data  
  
 **try**:  
 data = sock.recv(buffer)  
 **except**:  
 **return** None  
  
 **if** msg[-len(END\_OF\_MSG):] == END\_OF\_MSG:  
 **return** msg[:-len(END\_OF\_MSG)]  
 **return** None

כמו כן, יש פונקציות לשליחה וקבלה של מידע עם הצפנה ופענוח (כולל hash check):

**def** sendEncrypted(sock, key, msg):   
 *"""  
 Encode and encrypt the message and send it along with its hash digest.* **:param** *sock: the socket which is used to send the data.* **:param** *key: the key object to encrypt the message (RSA or AES key).* **:param** *msg: the message to be sent.* **:return***: True if sending procedure was successful, False otherwise.  
 """* msg = b64encode(msg)  
 msghex = SHA256.new(msg).hexdigest()  
  
 **if** isinstance(key, RSA\_Key):

*# if key is an RSA key we need to split msg into blocks of RSA\_BLOCK\_SIZE and encrypt each separately.* **for** i **in** range(0, len(msg), RSA\_BLOCK\_SIZE):  
 **if not** Send(sock, key.encrypt(msg[i:i+RSA\_BLOCK\_SIZE], 0)[0]

+ SEPARATOR, False):  
 **return** False  
  
 **elif** isinstance(key, AESCipher):

*# if key is an AES key, we don't need to split msg.* **if not** Send(sock, key.encrypt(msg) + SEPARATOR, False):  
 **return** False  
  
 **else**:

*# key is of unrecognized type (neither RSA nor AES)* **return** False  
  
 **if not** Send(sock, msghex):  
 **return** False  
  
 **return** True  
  
  
**def** recvDecrypted(sock, key, buffer=BUFF):   
 *"""  
 Decrypt and decode the message and verify it against its hash digest.* **:param** *sock: the socket which is used to receive the data.* **:param** *key: the key object to decrypt the data received (AES key).* **:return***: decrypted message (or error message in case of failure).  
 """* data = Receive(sock, buffer)  
 **if not** data **or** data == **''**:  
 **return** None  
  
 pieces = data.split(SEPARATOR)  
 msghex = pieces.pop()  
 msg = **''  
  
 for** piece **in** pieces:   
 *# decrypt each piece*  
 **try**:  
 msg += key.decrypt(piece)  
 **except**:  
 **return 'Decryption Error!'** *# hash check*

**if** SHA256.new(msg).hexdigest() != msghex:  
 **print 'Hash mismatch:'**, SHA256.new(msg).hexdigest(), msghex  
 **return 'Hash Check Error!'  
  
 return** b64decode(msg)

* **חיבור עם ה- GUI :**

החיבור ל- GUI מתבצע באמצעות Socket רגיל:

g\_GUI\_socket = socket.socket()

g\_GUI\_socket.connect(GUI\_ADDR)

ושליחת המידע ל- GUI מתבצעת באמצעות הפונקציה:

**def** SendToGUI(sock, lock, ip, command, \*params):  
 *"""  
 Adds client ip and padding to the message according to the protocol.* **:param** *sock: the socket to send data to the GUI.* **:param** *lock: socket's lock (to prevent multiple access).* **:param** *ip: client IP address.* **:param** *command: the command to be executed.* **:param** *params: the parameters of the command.* **:return***: True if sending procedure was successful, False otherwise.  
 """* msg = ip + SEPARATOR + command + **''**.join(SEPARATOR + param **for** param

**in** params) + END\_OF\_MSG  
 lock.acquire() *# lock to avoid simultaneous access from multiple threads*

**try**:  
 sock.sendall(msg)  
 **return** True  
 **except** socket.error:  
 **return** False  
 **finally**:  
 lock.release() *# release the lock, no matter what*

* **השגת מידע על כל התהליכים במחשב:**

בשביל העבודה עם WMI להשגת המידע על כל התהליכים יצרתי מחלקה בשם ProcessStats שבה יש פונקציה GetStats() שמחזירה את כל הנתונים לגבי כל התהליכים (בצורה של רשימה של מילונים, ראה פירוט מבני הנתונים). להלן המחלקה ProcessStats:

**class** ProcessStats:   
 *"""  
 ProcessStats provides stats and information about the processes running on the machine.  
 It uses the WMI to receive all data and organizes it in a list of dictionaries.  
 The last dictionary contains data about the TOTAL usage of resources.  
 """* **def** \_\_init\_\_(self):   
 *"""* *Initializes the process attributes list and the WMI service.*  *Defines constants: total amount of RAM and amount of logical cores.*  *"""* self.process\_attrs = [  
 **'Name'**,  
 **'IDProcess'**,  
 **'CreatingProcessID'**,  
 **'ElapsedTime'**,  
 **'PercentProcessorTime'**,  
 **'WorkingSetPrivate'**,  
 **'IODataBytesPersec'**,  
 **'IODataOperationsPersec'**,  
 **'ThreadCount'** ]  
  
 objWMIService = win32com.client.Dispatch(**'WbemScripting.SWbemLocator'**)  
 self.objSWbemServices = objWMIService.ConnectServer(**'.'**, **'root\cimv2'**)  
  
 query\_mem\_total = **'SELECT Capacity FROM Win32\_PhysicalMemory'** installed\_memory = self.objSWbemServices.ExecQuery(query\_mem\_total)  
 self.RAM\_TOTAL = sum(int(memory.Capacity) **for** memory **in** installed\_memory)  
  
 query\_logical\_cores = **'SELECT NumberOfLogicalProcessors FROM Win32\_ComputerSystem'** self.LOGICAL\_CORES = self.objSWbemServices.ExecQuery(query\_logical\_cores)[0].NumberOf

LogicalProcessors  
  
  
 **def** GetStats(self):   
 *"""*   *Show process stats for all processes running on the machine.*  *Returns a list of results (a result list for each process).*  *"""* proc\_results\_list = []  
  
 query\_proc = **'SELECT \* FROM Win32\_PerfFormattedData\_PerfProc\_Process'** Processes = self.objSWbemServices.ExecQuery(query\_proc)  
  
 query\_cpu = **'SELECT \* FROM Win32\_PerfFormattedData\_Counters\_ProcessorInformation WHERE Name="\_Total"'** total\_cpu\_usage = int(self.objSWbemServices.ExecQuery(query\_cpu)[0].PercentProcessorTime)  
  
 query\_mem\_avail = **'SELECT AvailableBytes FROM Win32\_PerfFormattedData\_PerfOS\_Memory'** available\_memory = float(self.objSWbemServices.ExecQuery(query\_mem\_avail)[0].Available

Bytes)  
  
 **for** Process **in** Processes:   
 **if** Process.Name == **'\_Global'**: *# Exclude '\_Global' process if exists* **continue** this\_proc\_dict = {}  
  
 **for** attribute **in** self.process\_attrs:   
 **if** Process.Name == **'\_Total'**:  
 **if** attribute == **'PercentProcessorTime'**: *# Insert total CPU usage percentage* this\_proc\_dict[attribute] = total\_cpu\_usage  
 **elif** attribute == **'WorkingSetPrivate'**: *# Insert physical memory usage percentage* this\_proc\_dict[attribute] = int(100 \* (1 - available\_memory / self.RAM\_TOTAL))  
 **else**:  
 eval\_str = **'Process.'** + attribute  
 this\_proc\_dict[attribute] = eval(eval\_str)  
  
 **else**:  
 **if** attribute == **'PercentProcessorTime'**: *# Divide by the amount of logical cores* this\_proc\_dict[attribute] = int(Process.PercentProcessorTime) / self.LOGICAL\_CORES  
 **else**:  
 eval\_str = **'Process.'** + attribute  
 this\_proc\_dict[attribute] = eval(eval\_str)  
  
 this\_proc\_dict[**'TimeStamp'**] = time.strftime(**'%Y-%m-%d %H:%M:%S'**) *# Add time stamp* proc\_results\_list.append(this\_proc\_dict)  
  
 **return** proc\_results\_list

* **הריגת תהליכים וכיבוי המחשב:**

הריגת תהליך מתבצעת בדיוק כפי שתואר קודם ע"י OpenProcess ו- ProcessTerminate, כאשר ניתן לראות שהוספתי גם בדיקה שמונעת מהתהליך להרוג את עצמו:

PROCESS\_TERMINATE = 1 *# OpenProcess mode that allows process termination*g\_CurrentProcessId = win32api.GetCurrentProcessId() *# PID of the python process it will run on***def** KillProcess(PID):   
 *"""  
 Kills a process by given Process ID.* **:return***: Error description if command failed, 'Success' otherwise.  
 """* **if** PID == g\_CurrentProcessId:  
 **return 'Error: attempted to kill client process.'  
 try**:  
 handle = win32api.OpenProcess(PROCESS\_TERMINATE, False, PID)  
 win32api.TerminateProcess(handle, -1)  
 win32api.CloseHandle(handle)  
 **return 'Success.'  
 except** Exception **as** e:  
 **return 'Exception occurred: '** + str(e)

כיבוי המחשב מתבצע גם כן בדיוק כפי שתואר קודם באמצעות os.system:

**def** Shutdown(timeout, comment):   
 *"""  
 Schedules system shutdown.* **:param** *timeout: the timeout in seconds before the shutdown.* **:param** *comment: a message for the user describing the reason for shutdown.* **:return***: Error description if command failed, 'Success' otherwise.  
 """* **try**:  
 result = os.system(**'shutdown /s /t %s /c "%s"'** % (timeout, comment))  
 **if** result:  
 **return 'Error in shutdown command. Error code %s.'** % result  
 **else**:  
 **return 'Success.'  
 except** Exception **as** e:  
 **return 'Exception occurred: '** + str(e)

* **שמירת המידע על התהליכים:**

כתבתי את המחלקה Database למטרה זו, שנעזרת בספריית sqlite3 של Python:

**import** sqlite3  
**import** datetime

TABLE\_NAME = **'ResourcesData'** *# the name of the (single) table in the database*

**class** Database:   
 *"""  
 The Database object creates or opens a database in the specified file.  
 It allows manipulation of the database, such as:  
 Create or Drop the table, Insert or Retrieve data, Delete old entries.  
 """* **def** \_\_init\_\_(self, db\_name, data\_lifespan):   
  *"""  
 Creates a database (if it doesn't exist already).  
 Creates a cursor object to execute all SQL queries.* **:param** *db\_name: The name of the file that stores the database.* **:param** *data\_lifespan: Determines how many days the entries stay in the database.  
 """* self.data\_lifespan = data\_lifespan  
 self.DB = sqlite3.connect(db\_name)  
 self.CURSOR = self.DB.cursor()  
 self.CURSOR.execute(**'PRAGMA auto\_vacuum = FULL'**)  
  
  
 **def** Save(self, delete=True):

*"""*  *Saves changes to the database.* **:param** *delete: Determines whether or not to delete old entries from the database.*  *"""* **if** delete:  
 DateTimeObj = datetime.datetime.now().date() - datetime.timedelta(days=self.data\_

lifespan)  
 DateThreshold = str(DateTimeObj)  
 self.DeleteOldData(DateThreshold)  
 self.DB.commit()  
  
  
 **def** Close(self):   
 *"""*  *Closes the connection to the database.*  *"""* self.DB.close()  
  
  
 **def** CreateTable(self):   
 *"""*  *Create the ResourceStats table if it doesn't exist already.*  *"""* create\_table\_query = **"CREATE TABLE IF NOT EXISTS "** + TABLE\_NAME + **"(ProcName VARCHAR(255), ProcId INT, CreatingProcId INT, ElapsedTime INT, CpuTimePercent INT, WorkingSetPrivate BIGINT, IOBytesPersec BIGINT, IOOperationsPersec BIGINT, Threads INT,**

**Date DATE, Time TIME)"** self.CURSOR.execute(create\_table\_query)  
 self.Save(False) *# make sure changes are saved!* **def** Insert(self, Name, IDProcess, CreatingProcessID, ElapsedTime, PercentProcessorTime, WorkingSetPrivate, IODataBytesPersec, IODataOperationsPersec, ThreadCount, TimeStamp):   
 *"""* *Insert the given parameters into the table.* *"""* insert\_query = **"INSERT INTO "** + TABLE\_NAME + **" VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"** Date, Time = TimeStamp.split(**' '**)  
 params = [Name, IDProcess, CreatingProcessID, ElapsedTime, PercentProcessorTime, WorkingSetPrivate, IODataBytesPersec, IODataOperationsPersec, ThreadCount, Date, Time]  
 self.CURSOR.execute(insert\_query, params)  
  
  
 **def** Retrieve(self):   
 *"""*  *A generator that retrieves all data from the database.*  *The data is sorted from the most to the least recent.*  *Creates an iterator of the rows in the table.*  *"""* retrieve\_query = **"SELECT \* FROM "** + TABLE\_NAME + **" ORDER BY Date DESC, Time DESC"** self.CURSOR.execute(retrieve\_query)  
 **while** True:  
 row = self.CURSOR.fetchone() *# eventually it will raise StopIteration* proc\_stats = {**'Name'**: row[0],  
 **'IDProcess'**: row[1],  
 **'CreatingProcessID'**: row[2],  
 **'ElapsedTime'**: row[3],  
 **'PercentProcessorTime'**: row[4],  
 **'WorkingSetPrivate'**: row[5],  
 **'IODataBytesPersec'**: row[6],  
 **'IODataOperationsPersec'**: row[7],  
 **'ThreadCount'**: row[8],  
 **'TimeStamp'**: row[9] + **' '** + row[10]}  
 **yield** proc\_stats  
  
  
 **def** DeleteOldData(self, DateThreshold):   
 *"""*  *Deletes all entries associated with a certain computer whose Time Stamp is very old.*  *Threshold should be set to the date from e.g. a day or a week ago.*  *"""* delete\_query = **"DELETE FROM "** + TABLE\_NAME + **" WHERE Date < ?"** params = [DateThreshold]  
 self.CURSOR.execute(delete\_query, params)

* **אפשרות לקביעת הגדרות לתוכנה:**

בקובץ MainForm.cs השייך ל- GUI כתבתי שתי פונקציות לקריאה וכתיבה ל- XML.

הקוד ארוך (בשל ריבוי המאפיינים שמגדירים את הכללים ותתי-הכללים) וניתן לראות אותו בחלק האחרון, קוד התוכנה. בקווים כלליים, הפונקציה שטוענת את ההגדרות מהקובץ קוראת את ה- XML באמצעות אובייקט XmlDocument ויוצרת על פי המידע הנקרא את ה- RuleControls והפונקציה ששומרת את ההגדרות בקובץ כותבת את ה- XML באמצעות אובייקט XmlWriter כאשר המידע שהיא כותבת נלקח מה- RuleControls הקיימים.

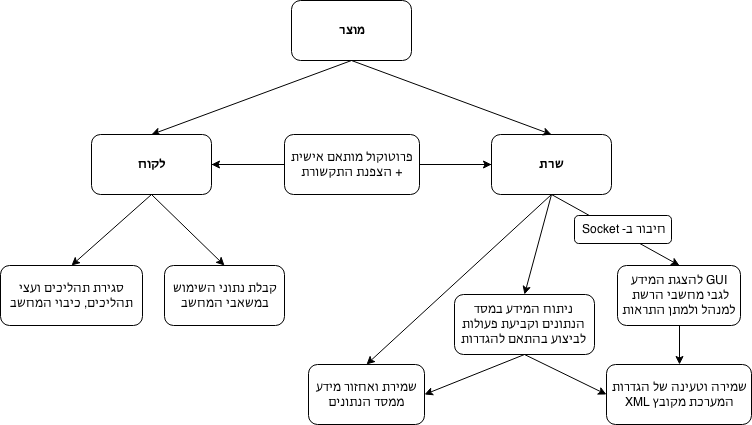
בנוסף, הקריאה מקובץ ה- XML בתוך מנוע ה- Python מתבצעת באמצעות ספריית xml.etree שממנה אני מייבא את ElementTree:

**import** xml.etree.ElementTree **as** ET

**def** LoadRulesFromXML(FilePath):   
 *"""  
 Loads rules from XML settings file.* **:param** *FilePath: the path of the XML file.* **:return***: a list containing all rules loaded.  
 """* Rules = []  
  
 tree = ET.parse(FilePath)  
 root = tree.getroot()  
  
 **for** rule **in** root:  
 SubRules = []  
  
 **for** subrule **in** rule:  
 Whitelist = []  
 **for** process **in** subrule.find(**'whitelist'**):  
 Whitelist.append(process.get(**'name'**))  
  
 Blacklist = []  
 **for** process **in** subrule.find(**'blacklist'**):  
 Blacklist.append(process.get(**'name'**))  
  
 subrule\_dict = subrule.attrib.copy()  
 subrule\_dict.update({**'whitelist'**: Whitelist, **'blacklist'**: Blacklist})  
 SubRules.append(subrule\_dict)  
  
 rule\_dict = rule.attrib.copy()  
 rule\_dict.update({**'subrules'**: SubRules})  
 Rules.append(rule\_dict)  
  
 **return** Rules

# תיאור המודולים המרכיבים את המוצר

עץ המודולים המרכיבים את המוצר:



המודולים המרכזיים הם:

* **Server.py, Client.py** – המודול של השרת והלקוח. השרת מאזין ב- socket בפורט 50,000, והלקוחות מתחברים אליו באמצעות ה- socket. לכל לקוח השרת פותח Thread נפרד ששומר את נתוני אותו לקוח. התקשורת ביניהם היא בפרוטוקול מותאם אישית, כאשר בהתחלה מתבצע תהליך של החלפת מפתחות, הלקוח מזדהה באמצעות הכתובת הפיזית שלו, ואז מתחילה העברת המידע ממש, נתוני השימוש במשאבים של כל התהליכים. כל התקשורת מוצפנת ע"י מודול ההצפנה.
* **AES\_Encryption.py** – המודול האחראי על ההצפנה. מודול זה נמצא גם בצד השרת וגם בצד הלקוח שכן שניהם משתמשים בו. במודול יש מחלקה שיוצרת מפתח הצפנה AES ומאפשרת להצפין ולפענח הודעות באמצעותו.
* **Send\_Receive.py** – המודול האחראי על תקשורת בהתאם לפרוטוקול. המודול מכיל פונקציה לשליחת מידע ל- GUI, ופונקציות לתקשורת עם הלקוח – שליחה וקבלה של מידע מוצפן בהתאם לפרוטוקול, וגם שליחה וקבלה של מידע כמו שהוא ללא הצפנה (השימוש בכך הוא רק בתחילת התקשורת בין השרת ללקוח).
* **Process\_Stats.py** – המודול בצד הלקוח שמקבל את נתוני השימוש במשאבי המחשב.

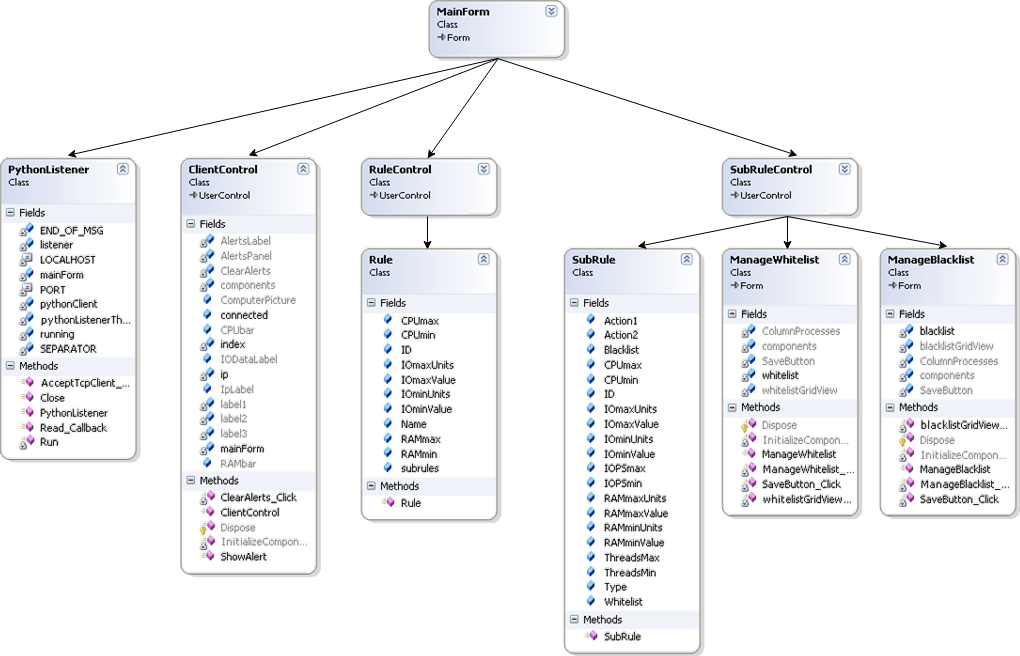
מודול זה משתמש ב- WMI ובאמצעות שאילתות SQL מתאימות מקבל מידע מפורט לגבי כל תהליך שרץ במחשב. באמצעות שאילתות SQL אחרות הוא מקבל מה- WMI מידע לגבי הכמות של הזיכרון הפיזי הכולל והפנוי, ומקבל את אחוז השימוש ב- CPU וכמות הליבות הלוגיות. באמצעות חישובים מתאימים המודול מוצא את אחוז השימוש בזיכרון הפיזי ובזמן המעבד.

* **Process\_Kill.py** – המודול בצד הלקוח האחראי על סגירת תהליכים. מודול פשוט מאוד שמכיל שתי פונקציות – האחת סוגרת תהליך בהינתן מזהה התהליך (ה- PID) והשנייה סוגרת את כל עץ התהליך (את התהליך עצמו, את כל התהליכים שהוא יצר, את כל התהליכים שהם יצרו, וכו'), גם כן בהינתן מזהה התהליך. הפונקציות משתמשות במודול win32api המובנה ב- Python.
* **Sql.py** – המודול בצד השרת שאחראי על העבודה עם מסד הנתונים. מודול זה יוצר מחלקה בשם Database שמאפשרת עבודה עם מסד נתונים. המחלקה מקבלת את שם הקובץ שבו מאוחסן מסד הנתונים (אם הקובץ לא קיים היא יוצרת אותו באופן אוטומטי) ומאפשרת ליצור את טבלת ה- ResourcesData אם היא עוד לא קיימת, להכניס מידע לטבלה, לאחזר מידע מהטבלה ולמחוק מידע ישן. כל העבודה עם מסד הנתונים עושה שימוש במודול sqlite3 המובנה ב- Python.
* **Followed\_Action.py** – המודול בצד השרת שאחראי על ניתוח הנתונים בהתאם להגדרות שקבע מנהל הרשת וביצוע פעולות נדרשות בהתאם. המודול מכיל מחלקה לניתוח נתונים. השרת מקבל מהלקוח את נתוני השימוש הכולל במשאבים שלו ומעביר אותם למודול זה. ניתוח הנתונים מתחיל מטעינה של ההגדרות שקבע מנהל הרשת מקובץ XML. אם זוהתה חריגה בנתונים (בהתאם להגדרות), מתבצעת סריקה של מסד הנתונים ומוציאים ממנו את הנתונים המלאים העדכניים ביותר על כל התהליכים. כעת מתבצע ניתוח של נתונים מלאים אלו בהתאם להגדרות, וכלפי כל תהליך שחורג מההגדרות נוקטים בפעולה הנדרשת (שוב, בהתאם להגדרות).
* **GUI** – המודול של הממשק הגרפי. הוא אחראי על הצגת כל הלקוחות המחוברים לשרת, הצגת נתוני השימוש של כל אחד במשאבי המחשב והצגת התראות לגבי תהליכים שחורגים מהכללים.

בנוסף, המודול מאפשר לצפות בהגדרות התוכנה הנוכחיות ולשנות אותן – להוסיף\לשנות\למחוק כללים ותתי-כללים, וכן ישנה אפשרות להחזיר את ההגדרות לברירת המחדל (ששמורה בקובץ XML נפרד). תפקיד נוסף של המודול הוא הצגת תמצית של המתרחש "מאחורי הקלעים" – התחברות והתנתקות של לקוחות, שגיאות לא צפויות שהתרחשו (Exceptions), נפילת השרת.

מודול ה- GUI מורכב בעצמו ממספר תתי-מודולים, המתוארים בתרשים שבעמוד הבא.

עץ תתי-המודולים של מודול ה- GUI:



תתי-המודולים הם:

* MainForm – הטופס הראשי של ה- GUI שמציג את כל ה- Controls.
* PythonListener – אחראי על התקשורת עם מנוע ה- Python.
* ClientControl – Control שמייצג לקוח שהתחבר ומשמש להצגת נתונים על הלקוח והתראות.
* RuleControl – Control שמייצג כלל בתוך הגדרות התוכנה ומשמש לקביעת תכונות הכלל.
* SubRuleControl – Control שמייצג תת-כלל בתוך כלל ומשמש לקביעת תכונות תת-הכלל.
* Rule – מחלקה שמייצגת כלל.
* SubRule – מחלקה שמייצגת תת-כלל.
* ManageWhitelist – טופס שמאפשר לקבוע את ה- whitelist עבור תת-כלל מסוים.
* ManageBlacklist – טופס שמאפשר לקבוע את ה- blacklist עבור תת-כלל מסוים.

# פירוט מבני הנתונים

|  |  |  |  |
| --- | --- | --- | --- |
| **מודול** | **שם המבנה** | **סוג** | **תפקיד** |
| Process\_Stats.py | self.process\_attrs | רשימה של strings | מגדיר את התכונות של תהליך שלגביהן תיתן מידע מחלקת ProcessStats |
| this\_proc\_dict | מילון string-int/string | מילון שהמפתחות שלו הם שמות התכונות של התהליך, והערכים הם הערכים של תכונות אלה בתהליך הספציפי |
| proc\_results\_list | רשימה של מילונים | רשימת כל המילונים this\_proc\_dict עבור כל התהליכים במחשב |
| Client.py | all\_process\_stats | רשימה של מילונים | רשימה שמכילה את כל הנתונים של כל התהליכים במחשב שהלקוח מקבל לאחר קריאה לפונקציית GetStats() של מחלקת ProcessStats |
| Server.py | all\_process\_stats | רשימה של מילונים | אותה רשימה כמו הנ"ל שהשרת מקבל מהלקוח |
| g\_client\_thread\_list | רשימה של אובייקטים | רשימה של כל האובייקטים ממחלקת ClientThread שיצר השרת (כלומר מעין רשימה של כל הלקוחות) |
| Followed\_Action.py | IT\_EMAIL\_ADDRESSES | רשימה של strings | רשימה של כל כתובות המייל של התמיכה הטכנית (IT) של החברה |
| UNITS\_CONVERSION\_DICT | מילון string-int | מילון המסייע להמיר בין היחידות השונות של מידע (Bytes, KB, MB, GB) |
| RESOURCES\_MATCH | רשימה של tuples של strings | רשימה המכילה זוגות של מחרוזות ומתאימה בין שמות "רגילים" של משאבי המחשב לבין השמות שלהם כתכונות של תהליך |
| SYS\_PROCESSES | רשימה של strings | רשימה המכילה שמות של תהליכי מערכת שמהם יש להתעלם בעת ניתוח הנתונים |

# פירוט מאגרי המידע

* **מסד הנתונים:**

התוכנה יוצרת בצד השרת מסד נתונים אחד לכל לקוח. שם הקובץ של המסד מתקבל מהפעלת hash על הכתובת הפיזית של המחשב (שאליה נוסף prefix). במסד הנתונים יש טבלה בודדת בשם **ResourcesData** שמאחסנת את כל הנתונים הנאספים מאותו מחשב.

מבנה הטבלה הוא:

|  |  |  |
| --- | --- | --- |
| **שם העמודה** | **סוג** | **תפקיד** |
| ProcName | VARCHAR(255) | שם התהליך |
| ProcID | INT | מזהה התהליך (ID) |
| CreatingProcID | INT | מזהה התהליך של התהליך שיצר את התהליך הנוכחי |
| ElapsedTime | INT | הזמן שהתהליך רץ בשניות |
| CpuTimePercent | INT | אחוז השימוש ב- CPU |
| WorkingSetPrivate | BIGINT | השימוש בזיכרון (ב- Bytes) |
| IOBytesPersec | BIGINT | כמות ה- Bytes שנקראו או נכתבו בפעולות I/O לשנייה |
| IOOperationsPersec | BIGINT | כמות פעולות I/O לשנייה |
| Threads | INT | כמות תתי-התהליכים |
| Date | DATE | התאריך שבו נאסף המידע |
| Time | TIME | השעה שבה נאסף המידע |

* **קובץ הגדרות בפורמט XML:**

לשם קביעת אופן הפעולה של התוכנה בתרחישים השונים קיים בצד השרת קובץ הגדרות בשם **settings.xml.** את הקובץ קורא המנוע הכתוב ב- Python בכל פעם שהוא מנתח את משאבי המערכת, וה- GUI כותב לקובץ זה בכל פעם שהמנהל משנה את ההגדרות ושומר אותן.

מבנה הקובץ הוא:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attributes** | **Tag Description** | **Parent Tag** | **Tag Name** |
| - | The root tag. | - | settings |
| ID, Name, CPUmin, CPUmax, RAMmin, RAMmax, IOmin, IOmax | Defines a new rule. | settings | rule |
| ID, CPUmin, CPUmax, RAMmin, RAMmax, IOmin, IOmax, IOPSmin, IOPSmax, ThreadsMin, ThreadsMax, Action, Type | Defines a new sub-rule of a rule. | rule | subrule |
| - | Defines the whitelist of a sub-rule. | subrule | whitelist |
| - | Defines the blacklist of a sub-rule. | subrule | blacklist |
| name | Defines a process in the whitelist or blacklist. | whitelist  **OR**  blacklist | process |

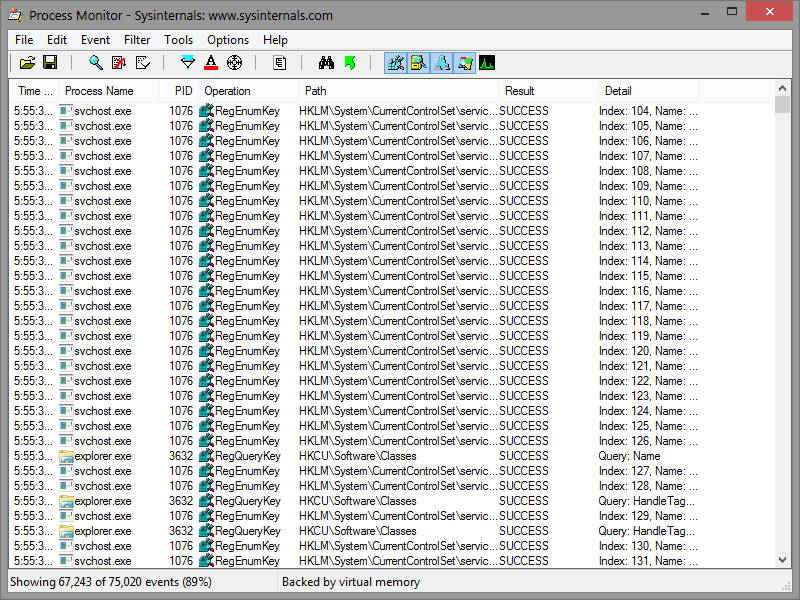
# השוואת העבודה עם פתרונות ויישומים קיימים

בעולם קיימות תוכנות רבות העוסקות במעקב אחר הפעילות המתרחשת במחשב ואחר התהליכים הרצים, ומציגות את המאפיינים שלהם (שימוש במעבד, שימוש בזיכרון, כתיבה וקריאה מהדיסק, חיבור לאינטרנט, תעבורת רשת וכו').

מספר תוכנות מסוג זה שעושות דברים הדומים בחלקם למה שעושה המוצר שלי:

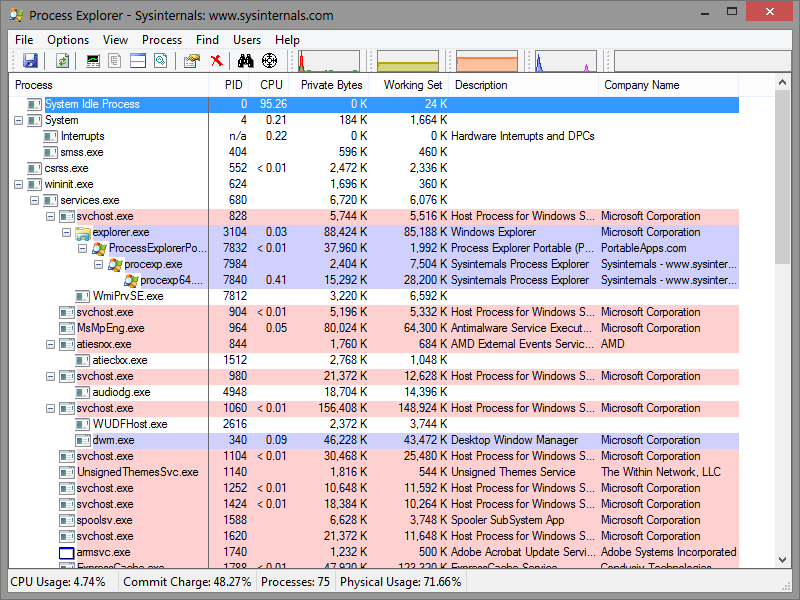
* **Process Monitor:**

זו תוכנה שנוצרה ע"י Sysinternals, שאותה רכשה Microsoft. היא מציגה בזמן אמת את פעילות מערכת המחשב – הפקודות (שמריצים תהליכים שונים) המתרחשות ברגע זה.

מעקב אחר תהליך יכול לשמש לזיהוי ניסיונות כושלים לקרוא ולכתוב מפתחות רישום. זה גם מאפשר סינון על מפתחות ספציפיים, תהליכים, מזהי תהליך וערכים. בנוסף, התוכנה מראה כיצד להשתמש בקבצי EXE וקבצי DLL, מזהה טעויות קריטיות בקבצי מערכת ועוד.

* **Process Explorer:**

זהו מנהל משימות וצג מערכת עבור Windows. גם כלי זה נוצר על ידי Sysinternals. הוא מספק את הפונקציונליות של מנהל המשימות של Windows יחד עם מערך עשיר של תכונות לאיסוף מידע על תהליכים הפועלים במערכת של המשתמש . הדבר יכול לשמש כצעד הראשון בניפוי בעיות תוכנה או מערכת.

Process Explorer יכול לשמש לאיתור בעיות. לדוגמה, הוא מספק אמצעי לחיפוש משאבים שמוחזקים על ידי תהליך. זה יכול לשמש לאיתור של מה שמחזיק את הקובץ פתוח ומונע את השימוש בו על ידי תכנית אחרת. דוגמה נוספת היא היכולת להראות את קווי הפקודה המשמשים להפעלת תכנית, המאפשרת לתהליכים אחרים זהים להיות מכובדים.

**ההבדל העיקרי** בין התוכנות הנ"ל לתוכנה שלי הוא שהתוכנה שלי מאפשרת שליטה מרחוק, ומספקת למנהל הרשת בקרה על המתרחש בכל המחשבים ברשת. בנוסף, קיים ממשק שמאפשר למנהל הרשת להגדיר כללים מדויקים מאוד שקובעים את אופן פעולת התוכנה, ועל פיהם היא מתערבת בנעשה במחשבי הרשת באופן **אוטומטי** כדי למנוע עומס וקריסות וכדי לשמור על נתוני שימוש תקינים.

# מדריך התקנה למשתמש

אילוצים ודרישות מערכת

* התוכנה פותחה עבור מערכת ההפעלה Windows. היא נבדקה ונוסתה בגרסת **Windows 7** בלבד, ועל כן השימוש בה בגרסה אחרת של Windows הוא באחריות המשתמש בלבד.
* בנוסף, יש לוודא שמותקן בכל מחשב קצה ובשרת **Python 2.7 (32-bit)**, ושמותקנות ספריות **PyWin32** ו- **PyCrypto** עבור גרסה זו של Python. בביליוגרפיה נמצאים לינקים להורדת Python ולהתקנת הספריות.
* כמו כן, צריכה להיות מותקנת במחשב השרת **.NET Framework 4.0** או גרסה עדכנית יותר (התוכנה נבדקה בגרסאות **4.0** ו- **4.5**).
* לבסוף, יש לוודא שקיים חיבור בין מחשבי הקצה למחשב עליו יותקן השרת (ובפרט יש לסדר את הגדרות ה- Firewalls שקיימים ברשת הפנימית כך שיאפשרו לתוכנה ליצור חיבורים).

התקנת והרצת התוכנה

* צד שרת:

1. יש להוריד את תיקיית **CRM Server** למחשב השרת.
2. בקובץ config.txt ניתן לקבוע את מספר ה- Port שהשרת יפתח כדי לקבל חיבורים מלקוחות (המספר בשורה הראשונה בקובץ) ואת מספר ה- Port שהשרת יפתח בשביל לתקשר עם ה- GUI (המספר בשורה השנייה בקובץ). ערכי ברירת המחדל הם 50,000 ו- 30,000.
3. כדי להריץ את השרת, יש להריץ את הקובץ **Computer Resources Monitoring.exe** שבתוך תיקיית **CRM Server**. קרוב לוודאי שבהרצה הראשונה תתקבל הודעת אזהרה מ- Windows Firewall ויש לאשר אותה.

* צד לקוח:

1. יש להוריד את תיקיית **CRM Client** לכל אחד ממחשבי הקצה שרוצים להריץ עליהם את התוכנה.
2. בקובץ emails.txt יש להכניס את כל כתובות המייל של התמיכה הטכנית (IT) של החברה. לכתובות מייל אלה התוכנה תשלח מיילים כאשר נבחרת אפשרות "Send Email to IT". כל כתובת מייל צריכה להופיע בשורה חדשה, ללא הוספת תווים נוספים להפרדה (כגון פסיקים).
3. כדי להריץ את התוכנה על מחשבי הקצה, יש ללחוץ לחיצה כפולה על הקובץ **Client.py** שבתוך תיקיית **CRM Client** במחשב זה. גם כאן קרוב לוודאי שבהרצה הראשונה תתקבל הודעת אזהרה מ- Windows Firewall ויש לאשר אותה.

# תיאור הממשק למשתמש

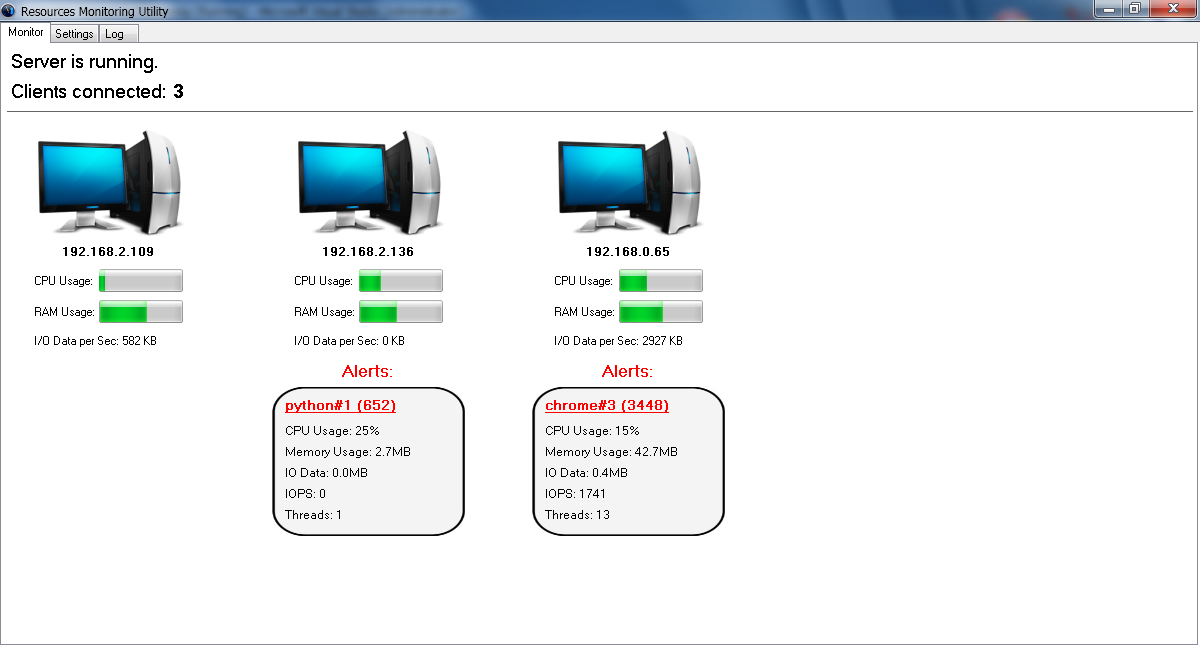
הממשק הגרפי אותו הינך (מנהל הרשת) רואה לאחר הרצת התוכנה מתחלק ל-3 חלקים:

1. ה- **Monitor**.

בחלק זה למעלה יש פס טקסט המתאר את מצבו של ה- Server (אם הוא עלה, רץ או נסגר). שאר המסך מציג את המחשבים שהתחברו לשרת. צבע מסך כחול מעיד על חיבור פעיל ואילו צבע אפור מעיד על מחשב שהתנתק מהשרת.

לכל מחשב מחובר יש כיתוב מתחתיו עם כתובת ה- IP שלו, ומתחתיה מוצגים נתוני השימוש שלו במשאבי מערכת (במעבד, בזיכרון הפיזי ופעולות I/O), אשר מתעדכנים כל פרק זמן קצוב.

מתחת לנתונים אלה מוצגות התראות אשר התוכנה שולחת כאשר היא מזהה פעילות חריגה במחשב הקצה המדובר. ההתראות מכילות את שם התהליך, מזהה תהליך ופירוט של השימוש במשאבים של התהליך. ההתראות העדכניות ביותר תמיד יהיו העליונות ביותר, וכדי לראות התראות ישנות יותר (כאשר מצטברות מספר התראות) יש לגלול עם העכבר למטה.



1. ה- **Settings**.

חלק זה מאפשר לך לצפות בהגדרות התוכנה הנוכחיות ולשנות אותן (לקבוע הגדרות משלך).

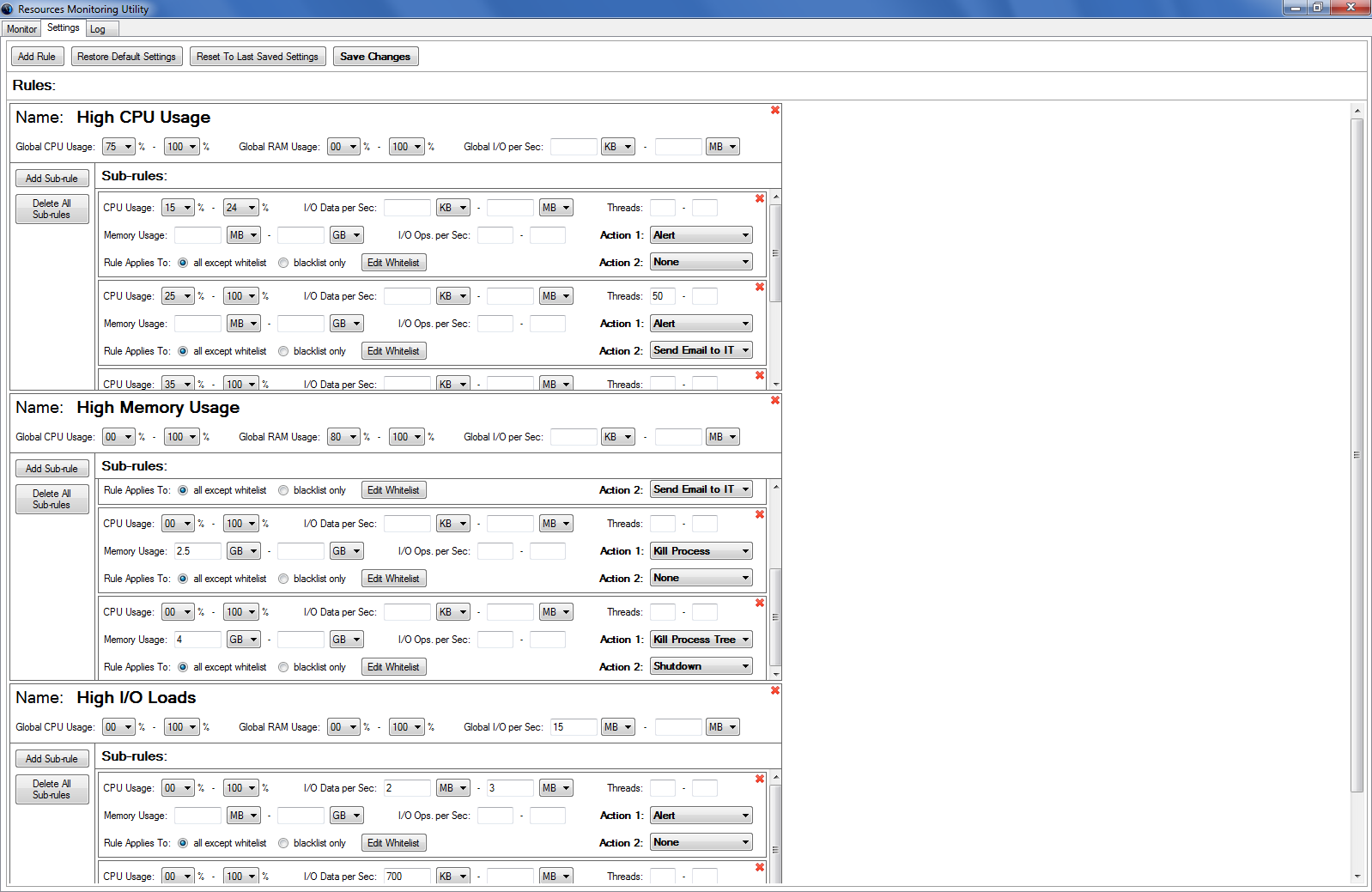
ההגדרות מורכבות מכללים. הכללים קובעים את טווחי השימוש הכולל במשאבי המערכת שלגביהם הם תקפים. למשל, כאשר יש שימוש ביותר מ-25% מזמן המעבד וביותר מ- 3GB של RAM, הכלל נכנס לתוקף.

לכללים יש תתי-כללים, שקובעים את טווחי השימוש במשאבי המערכת של תהליכים אינדיבידואליים שלגביהם הם תקפים. למשל, כאשר תהליך מסוים צורך יותר מ- 10% מזמן המעבד, יותר מ- 300 MB של RAM, ויש לו מעל 50 ת'רדים, אז תת-הכלל נכנס לתוקף (אם הכלל שמכיל אותו בתוקף), ונוקט כלפי תהליך זה בפעולה שנבחרה באופן אוטומטי.

הכפתורים בחלקו העליון של מסך זה מאפשרים להוסיף כלל חדש, להחזיר את ההגדרות לברירת מחדל, להחזיר את ההגדרות למצב האחרון שנשמר, ולשמור את השינויים שבוצעו. יש להקפיד ללחוץ על כפתור שמירת השינויים כדי שכל השינויים בהגדרות יישמרו.

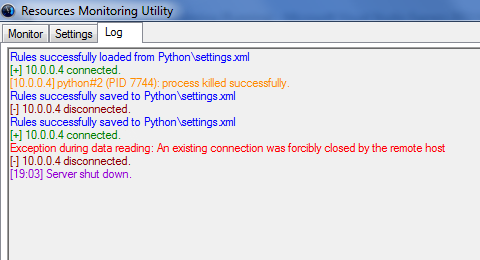
לכל כלל אפשר לתת שם, שבעזרתו יהיה קל יותר לזהות את הכלל ולהבין מה מטרתו הכללית. לכל כלל ניתן לקבוע לגבי אילו טווחים של שימוש במעבד, בזיכרון וב- I/O הוא תקף. הכפתורים בצד שמאל מאפשרים להוסיף תתי-כללים חדשים ולמחוק את כל תתי-הכללים של הכלל. האיקס האדום בפינה הימנית-עליונה מאפשר למחוק את הכלל.

לכל תת-כלל ניתן לקבוע את טווח השימוש במשאבים שלגביו הוא תקף (שימוש במעבד, בזיכרון, כמות I/O Data לשנייה, כמות פעולות I/O לשנייה ומספר Threads). בנוסף, ניתן לקבוע את הפעולות שהתוכנה תבצע כאשר תהליך מקיים את תת-הכלל. כמו כן, ניתן לבחור האם תת-הכלל תקף לכל התהליכים פרט לרשימה מסוימת (whitelist), או שהוא תקף רק לרשימה מסוימת של תהליכים (blacklist), ורשימות אלה כמובן ניתנות לעריכה. גם כאן האיקס האדום בפינה הימנית-עליונה מאפשר למחוק את תת-הכלל.



1. ה- **Log**.

חלק זה מציג תמצית של המתרחש "מאחורי הקלעים". הוא מודיע על התחברות והתנתקות של לקוחות, על שגיאות לא צפויות שהתרחשו (Exceptions), על נפילה של השרת וכו'. זה מאפשר צפייה רטרואקטיבית בפעילות של התוכנה, בכל השלבים והפעולות שננקטו. כמו כן, אם וכאשר מתרחשים דברים לא צפויים, זה עשוי לעזור לאתר את מקור הבעיה.



# מבט אישי על העבודה ותהליך הפיתוח

במהלך העבודה על הפרויקט למדתי לראשונה על ה- WMI ומאוד נהניתי לגלות שקיים ממשק כה נוח לקבלת מידע לגבי רכיבי המערכת. נהניתי ללמוד על האפשרויות המגוונות שהוא מספק ולעבוד איתן. כמו כן, נהניתי לגלות את מודול ה- sqlite3 ב- Python שהיה מאוד נוח ובעזרתו כתבתי בקלות רבה מחלקה לעבודה עם מסד נתונים. גם במהלך העבודה עם קבצי XML הופתעתי לטובה מקלות השימוש בהם, הן מבחינה לוגית והן מבחינה תכנותית.

פיתחתי הרגלי תכנות טובים במהלך העבודה על הפרויקט, הכוללים חלוקה למודולים ותכנון לוגי מראש, תיעוד מלא, שימוש בקבועים, במחלקות ובפונקציות, נתינת שמות משמעותיים למשתנים.

האתגרים והקשיים שנתקלתי בהם במהלך פיתוח הפרויקט הם:

* כאשר הוספתי הצפנה של המידע העובר בין השרת והלקוח, בתחילה בחרתי באלגוריתם האסימטרי El-Gamal, אך גיליתי שייצור של מפתח גדול מספיק הנחשב לבטיחותי (2048 ביטים) לוקח זמן רב מדי. כדי לפתור את הבעיה עברתי לאלגוריתם הנפוץ יותר RSA, שבו זמן ייצור המפתח הוא קצר בהרבה.
* לאחר מכן הבנתי שהצפנה באמצעות מפתח RSA בלבד היא לא ישימה, כי אין דרך בטוחה לתאם בין המפתחות של השרת והלקוח (הדרך היחידה היא להשתמש באותו מפתח קבוע תמיד וזה לא בטיחותי). כדי לפתור את הבעיה מימשתי בפרוטוקול שלי החלפת מפתחות מאובטחת שבמסגרתה השרת שולח ללקוח את מפתח ה- RSA הפומבי שלו והלקוח מצפין בעזרתו את מפתח ה- AES שלו ושולח לשרת. מאותו רגע שני הצדדים משתמשים במפתח ה- AES להצפנה ופיענוח.
* כשעבדתי על המודול שמקבל את נתוני השימוש במשאבים של כל התהליכים, הבדיקות שערכתי הראו שהשימוש הכולל ב- CPU שהתקבל היה מאוד לא מדויק. ערכתי חיפושים באינטרנט ומצאתי מספר דרכים שונות לקבל את הנתון הזה, וניסיתי את כולן כדי לבדוק איזו מהן מניבה את התוצאות הכי מדויקות, ואותה בחרתי בשביל המוצר המוגמר.
* קושי נוסף שהתעורר היה כיצד להחליט איזה נתון של שימוש בזיכרון לקחת בחשבון. ישנם מספר נתוני שימוש בזיכרון של תהליך (Private Bytes, Virtual Bytes, Working Set ועוד) ורציתי לבחור אחד (לשם פשטות) שישקף בצורה המיטבית את השימוש בזיכרון. לאחר מחקר וקריאה של חומר רקע באינטרנט הגעתי למסקנה שהנתון הכי מתאים לצרכים שלי הוא ה- Private Working Set ובו השתמשתי במוצר המוגמר.
* במהלך הבדיקות גיליתי שכאשר מספר מחשבים מתחברים לשרת והשרת מנסה בו-זמנית להכניס לתוך מסד הנתונים את המידע ששולח כל אחד מהם, מתעוררת בעיה, כי לא ניתן לכתוב לאותו קובץ מכמה Threads בו-זמנית. כדי לפתור את הבעיה שיניתי את הקוד כך שלכל מחשב יהיה מסד נתונים משלו בהתאם לכתובת הפיזית שלו.

הערכת הפתרון לעומת התכנון:

* התכנון המקורי היה שהמוצר יוכל לא רק לסגור תהליכים במקרה של זיהוי פעילות חריגה, אלא גם יוכל להגביל את המשאבים העומדים לרשותו של התהליך הבעייתי כצעד מקדים לפני סגירתו. במהלך המחקר גיליתי שהדבר ניתן למימוש רק ע"י עבודה ב- Kernel Mode והדבר כרוך בקשיים רבים. הגעתי למסקנה שכדי לעמוד בזמנים אין לי ברירה אלא לזנוח רעיון זה ולהשאיר רק את האפשרות של סגירת תהליך.
* בתכנון המקורי המוצר היה אמור ללמוד לבד את הרגלי השימוש במשאבי המחשב מהנתונים שהוא אוסף לאורך זמן, ועל סמך זה לזהות חריגויות ולפעול בהתאם. גם כאן בשל מורכבות הדבר (יש לפתח אלגוריתם של לימוד עצמי, מעין בינה מלאכותית) החלטתי לוותר על רעיון זה ולהסתפק במתן אפשרות למנהל הרשת לתת הגדרות מפורטות שעל פיהן יפעל המוצר בכל מקרה ומקרה.
* מלבד הנקודות שהוזכרו לעיל, המוצר המוגמר תואם בדיוק את התכנון המקורי.

ההמלצות לשיפורו הן אם כן:

* להוסיף אפשרות להגביל את השימוש של התהליכים במשאבים.
* להוסיף אלגוריתם לימוד עצמי, שילמד את הרגלי השימוש בכל אחד ממחשבי הרשת באמצעות המידע שנאגר לאורך זמן במסד הנתונים, ויזהה פעילות חריגה בהתאם לכך.
* בנוסף, אין ספק שיש מקום לשיפור המוצר מבחינת אבטחת מידע ומימוש פרוטוקול יותר מקצועי, יעיל ומאובטח.

# ביבליוגרפיה

1. אתר הדרכה לסייבר שיוסד על ידי המנחה שלי מיכאל:  
   [https://sites.google.com/site/coursecyber](https://sites.google.com/site/coursecyber/)
2. הפורום הגדול ביותר לשאלות ותשובות בנושאי תכנות:  
   [http://stackoverflow.com](http://stackoverflow.com/)
3. MSDN, הרשת הרשמית של Microsoft למפתחים:  
   <http://msdn.microsoft.com/library>
4. התיעוד הרשמי של Python:  
   [https://docs.python.org/2.7](https://docs.python.org/2.7/)
5. הוויקי הרשמית של Python:  
   [https://wiki.python.org/moin](https://wiki.python.org/moin/)
6. אתר ללימוד Python:  
   [http://www.tutorialspoint.com/python](http://www.tutorialspoint.com/python/)
7. להורדת Python 2.7 (32-bit):

<https://www.python.org/ftp/python/2.7/python-2.7.msi>

1. להתקנת ספריית PyWin32 בשביל Python 2.7 (32-bit):

[http://sourceforge.net/projects/pywin32/files/pywin32/Build 219/pywin32-219.win32-py2.7.exe/download](http://sourceforge.net/projects/pywin32/files/pywin32/Build%20219/pywin32-219.win32-py2.7.exe/download)

1. להתקנת ספריית PyCrypto בשביל Python 2.7 (32-bit):

<http://www.voidspace.org.uk/downloads/pycrypto26/pycrypto-2.6.win32-py2.7.exe>

**קוד התוכנה**

**Send\_Receive.py:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*# region IMPORTS***import** sys  
**import** socket  
**from** base64 **import** b64encode, b64decode  
  
**try**:  
 **from** Crypto.PublicKey.RSA **import** \_RSAobj **as** RSA\_Key  
 **from** Crypto.Hash **import** SHA256  
**except** ImportError:  
 **print 'PyCrypto module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
  
**try**:  
 **from** AES\_Encryption **import** AESCipher  
**except** ImportError:  
 **print 'AES\_Encryption module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
*# endregion  
  
  
# region CONSTANTS*BUFF = 1024 *# the BUFFER for the socket send/receive*END\_OF\_MSG = **'###'** *# end of message string in the protocol*SEPARATOR = **'$$$'** *# separator string in the protocol*RSA\_BLOCK\_SIZE = 128 *# max block size allowed to encrypt at once by RSA  
# endregion***def** SendToGUI(sock, lock, ip, command, \*params):  
 *"""  
 Adds client ip and padding to the message according to the protocol.* **:param** *sock: the socket to send data to the GUI.* **:param** *lock: socket's lock (to prevent multiple access).* **:param** *ip: client IP address.* **:param** *command: the command to be executed.* **:param** *params: the parameters of the command.* **:return***: True if sending procedure was successful, False otherwise.  
 """* msg = ip + SEPARATOR + command + **''**.join(SEPARATOR + param **for** param **in** params) + END\_OF\_MSG  
 lock.acquire() *# lock to avoid simultaneous access from multiple threads* **try**:  
 sock.sendall(msg)  
 **return** True  
 **except** socket.error:  
 **return** False  
 **finally**:  
 lock.release() *# release the lock, no matter what***def** Send(sock, msg, msg\_complete=True):  
 *"""  
 Adds padding to the end of the message (if indicated so) and sends it via the socket.* **:param** *sock: the socket which is used to send the data.* **:param** *msg: the message to be sent.* **:param** *msg\_complete: indicates whether the message is complete and should be padded.* **:return***: True if sending procedure was successful, False otherwise.  
 """* **if** msg\_complete:  
 msg += END\_OF\_MSG  
 **try**:  
 sock.sendall(msg)  
 **return** True  
 **except** socket.error:  
 **return** False  
  
  
**def** Receive(sock, buffer=BUFF):  
 *"""  
 Receives all the data from the socket by chunks of buffer size at a time.  
 Removes the padding at the end of the message.* **:param** *sock: the socket which is used to receive the data.* **:param** *buffer: the amount of data to receive at a time.* **:return***: the message received (unpadded), or None if receive was unsuccessful.  
 """* msg = **''  
 try**:  
 data = sock.recv(buffer)  
 **except** socket.error:  
 **return** None  
  
 **while** data:  
 **if** data[-len(END\_OF\_MSG):] == END\_OF\_MSG: *# we reached the end of the message, so remove padding and finish* msg += data  
 **break  
 else**:  
 msg += data  
  
 **try**:  
 data = sock.recv(buffer)  
 **except** socket.error:  
 **return** None  
  
 **if** msg[-len(END\_OF\_MSG):] == END\_OF\_MSG:  
 **return** msg[:-len(END\_OF\_MSG)]  
 **return** None  
  
  
**def** sendEncrypted(sock, key, msg):  
 *"""  
 Encode and encrypt the message and send it along with its hash digest.* **:param** *sock: the socket which is used to send the data.* **:param** *key: the key object to encrypt the message (RSA or AES key).* **:param** *msg: the message to be sent.* **:return***: True if sending procedure was successful, False otherwise.  
 """* msg = b64encode(msg)  
 msghex = SHA256.new(msg).hexdigest()  
  
 **if** isinstance(key, RSA\_Key):  
 *# if key is an RSA key we need to split msg into blocks of RSA\_BLOCK\_SIZE and encrypt each separately.* **for** i **in** range(0, len(msg), RSA\_BLOCK\_SIZE):  
 **if not** Send(sock, key.encrypt(msg[i:i+RSA\_BLOCK\_SIZE], 0)[0] + SEPARATOR, False):  
 **return** False  
  
 **elif** isinstance(key, AESCipher):  
 *# if key is an AES key, we don't need to split msg.* **if not** Send(sock, key.encrypt(msg) + SEPARATOR, False):  
 **return** False  
  
 **else**:  
 *# key is of unrecognized type (neither RSA nor AES)* **return** False  
  
 **if not** Send(sock, msghex):  
 **return** False  
  
 **return** True  
  
  
**def** recvDecrypted(sock, key, buffer=BUFF):  
 *"""  
 Decrypt and decode the message and verify it against its hash digest.* **:param** *sock: the socket which is used to receive the data.* **:param** *key: the key object to decrypt the data received (AES key).* **:return***: decrypted message (or error message in case of failure).  
 """* data = Receive(sock, buffer)  
 **if not** data **or** data == **''**:  
 **return** None  
  
 pieces = data.split(SEPARATOR)  
 msghex = pieces.pop()  
 msg = **''  
  
 for** piece **in** pieces:  
 *# decrypt each piece* **try**:  
 msg += key.decrypt(piece)  
 **except**:  
 **return 'Decryption Error!'** *# hash check* **if** SHA256.new(msg).hexdigest() != msghex:  
 **print 'Hash mismatch:'**, SHA256.new(msg).hexdigest(), msghex  
 **return 'Hash Check Error!'  
  
 return** b64decode(msg)

**AES\_Encryption.py:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*# region IMPORTS***import** sys  
**from** base64 **import** b64encode, b64decode  
  
**try**:  
 **from** Crypto **import** Random  
 **from** Crypto.Cipher **import** AES  
 **from** Crypto.Hash **import** SHA256  
**except** ImportError:  
 **print 'PyCrypto module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
*# endregion  
  
  
# region CONSTANTS*AES\_BLOCK\_SIZE = 32 *# the block size for the AES cipher object  
# endregion***class** AESCipher:  
 *"""  
 Creates an object which can be used to encrypt and decrypt messages using the AES algorithm.  
 Receives a string upon initialization which is used to create a random AES key.  
 """* **def** \_\_init\_\_(self, keystring):  
 *"""  
 Initializes the cipher object.  
 Sets the cipher block size and creates the key by hashing the given keystring string.  
 """* self.bs = AES\_BLOCK\_SIZE  
 self.key = SHA256.new(keystring).digest()  
  
 **def** encrypt(self, raw):  
 *"""  
 Receives raw message.  
 Adds padding and creates an AES cipher object to encrypt it.  
 Returns the encrypted and encoded message.  
 """* raw = self.pad(raw)  
 iv = Random.new().read(AES.block\_size)  
 cipher = AES.new(self.key, AES.MODE\_CBC, iv)  
 **return** b64encode(iv + cipher.encrypt(raw))  
  
 **def** decrypt(self, enc):  
 *"""  
 Receives encrypted and encoded message.  
 Decodes it and creates an AES cipher object to decrypt it.  
 Returns the decrypted raw message.  
 """* enc = b64decode(enc)  
 iv = enc[:AES.block\_size]  
 cipher = AES.new(self.key, AES.MODE\_CBC, iv)  
 **return** self.unpad(cipher.decrypt(enc[AES.block\_size:]))  
  
 **def** pad(self, s):  
 *"""  
 Receives input string s.  
 Adds padding in order to make string length divisible by the block size, as required in AES.  
 Returns the padded string.  
 """* **return** s + (self.bs - len(s) % self.bs) \* chr(self.bs - len(s) % self.bs)  
  
 @staticmethod  
 **def** unpad(s):  
 *"""  
 Receives input string s.  
 Removes the padding that was added to it during the encryption process.  
 Returns the unpadded string.  
 """* **return** s[:-ord(s[len(s)-1:])]

**Client.py:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*# region IMPORTS***import** sys  
**import** socket  
**import** pickle  
**import** time  
**from** uuid **import** getnode  
  
**try**:  
 **from** Crypto **import** Random  
 **from** Crypto.PublicKey **import** RSA  
**except** ImportError:  
 **print 'PyCrypto module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
  
**try**:  
 **from** AES\_Encryption **import** AESCipher  
**except** ImportError:  
 **print 'AES\_Encryption module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
  
**try**:  
 **from** Send\_Receive **import** Send, Receive, sendEncrypted, recvDecrypted  
**except** ImportError:  
 **print 'Send\_Receive module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
  
**try**:  
 **from** Process\_Stats **import** ProcessStats  
**except** ImportError:  
 **print 'Process\_Stats module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
  
**try**:  
 **from** Process\_Kill **import** KillProcess, KillProcessTree, Shutdown  
**except** ImportError:  
 **print 'Process\_Kill module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
*# endregion  
  
  
# region CONFIG***def** LoadConfig():  
 *""" Loads the configuration from the file """* config\_file = None  
 **try**:  
 config\_file = open(**'config.txt'**)  
 host = config\_file.readline().strip(**'\n'**) *# read host name from file* port = int(config\_file.readline().strip(**'\n'**)) *# read port number from file* **return** host, port  
 **except**:  
 Log(LOG\_FILE\_PATH, **'Bad configuration file! Exiting...'**)  
 sys.exit(1)  
 **finally**:  
 **if** config\_file:  
 config\_file.close() *# close the file when finished*config = LoadConfig()  
*# endregion  
  
  
# region CONSTANTS*HOST = config[0] *# the host of the server*PORT = config[1] *# the port of the server*ADDR = (HOST, PORT) *# the full address of the server*AES\_KEY\_SIZE = 32 *# the size of the key generating the AES cipher object*PAUSE\_INTERVAL = 5 *# interval (in seconds) to pause between cycles of process stats sending*ERR\_SERVER\_DOWN = 100 *# error code: server is down*ERR\_DATABASE\_PROBLEM = 200 *# error code: database didn't open*ERR\_OBJ\_NOT\_CREATED = 300 *# error code: ProcessStats object wasn't created*COMMAND\_SEPARATOR = **' && '** *# separator between command and params*KILL\_PROC\_COMMAND = **'Kill Process'** *# Kill Process command*KILL\_PROC\_TREE\_COMMAND = **'Kill Process Tree'** *# Kill Process Tree command*SHUTDOWN\_COMMAND = **'Shutdown'** *# Shutdown command*LOG\_FILE\_PATH = **'log.txt'** *# the path of the log file  
# endregion  
  
  
# region GLOBALS*g\_AES\_Key = AESCipher(Random.new().read(AES\_KEY\_SIZE)) *# AES key*g\_MAC\_Address = (**'{}{}:'**\*6)[:-1].format(\*list(**'{:012x}'**.format(getnode()).upper())) *# MAC address  
# endregion***def** Log(FilePath, string, add\_caller=False):  
 *"""  
 Writes to the log file provided the file path, and also prints the string.  
 Adds time stamp and caller function name (if add\_caller is set to True).  
 """* caller\_str = **'In '** + sys.\_getframe().f\_back.f\_code.co\_name + **' - '  
 if** add\_caller:  
 string = caller\_str + string  
 time\_stamp = **'['** + time.strftime(**'%d/%m/%Y %H:%M:%S'**) + **'] '** f = open(FilePath, **'a'**)  
 f.write(time\_stamp + string + **'\n'**)  
 f.close()  
 **print** string  
  
  
**def** StartConnection():  
 *""" Starts the connection with the server """* **global** sock  
 sock = socket.socket()  
 **while** True:  
 **try**:  
 sock.connect(ADDR)  
 **break  
 except** socket.error:  
 Log(LOG\_FILE\_PATH, **"Server isn't responding..."**)  
 time.sleep(PAUSE\_INTERVAL)  
  
 Send(sock, **'Hello Server.'**)  
  
  
**def** GetRSAKey():  
 *""" Obtains the RSA key from the server. """* **global** server\_RSA\_Key  
 server\_RSA\_Key = None  
  
 **while not** server\_RSA\_Key:  
 data = Receive(sock)  
 **if not** data:  
 **return** ERR\_SERVER\_DOWN  
 **try**:  
 server\_RSA\_Key = pickle.loads(data) *# load key from data using pickle* **assert** isinstance(server\_RSA\_Key, RSA.\_RSAobj) *# verify that key is RSA key object* Log(LOG\_FILE\_PATH, **'RSA key obtained.'**)  
 Send(sock, **'Key Accepted.'**)  
 **except**:  
 *# if data was not a legal pickle dump OR key object wasn't an RSA key* Log(LOG\_FILE\_PATH, **'Error! Bad RSA key. Requesting again...'**)  
 server\_RSA\_Key = None  
 Send(sock, **'Bad Key.'**)  
  
 data = Receive(sock)  
 **if not** data:  
 **return** ERR\_SERVER\_DOWN  
 **if** data != **'Send AES Key.'**:  
 Log(LOG\_FILE\_PATH, **'Unexpected data received:'** + data, True)  
  
  
**def** SendAESKey(server\_RSA\_Key):  
 *""" Sends the AES key to the server. """* **while** True:  
 sendEncrypted(sock, server\_RSA\_Key, pickle.dumps(g\_AES\_Key))  
 data = Receive(sock)  
  
 **if not** data:  
 **return** ERR\_SERVER\_DOWN  
  
 **elif** data == **'Decryption Error.'**:  
 Log(LOG\_FILE\_PATH, **'Decryption error on server side. Retrying...'**, True)  
  
 **elif** data == **'Hash Check Failed.'**:  
 Log(LOG\_FILE\_PATH, **'Hash check failed on server side. Retrying...'**, True)  
  
 **elif** data == **'Bad key.'**:  
 Log(LOG\_FILE\_PATH, **'Key unaccepted by server. Retrying...'**, True)  
  
 **elif** data == **'Key Accepted. Send MAC Address.'**:  
 Log(LOG\_FILE\_PATH, **'AES key accepted.'**)  
 **break** time.sleep(PAUSE\_INTERVAL)  
  
  
**def** SendMACAddress():  
 *""" Sends the MAC address to the server. """* **while** True:  
 sendEncrypted(sock, g\_AES\_Key, g\_MAC\_Address)  
 data = recvDecrypted(sock, g\_AES\_Key)  
  
 **if not** data:  
 **return** ERR\_SERVER\_DOWN  
  
 **elif** data == **'Decryption Error.'**:  
 Log(LOG\_FILE\_PATH, **'Decryption error on server side. Retrying...'**, True)  
  
 **elif** data == **'Hash Check Failed.'**:  
 Log(LOG\_FILE\_PATH, **'Hash check failed on server side. Retrying...'**, True)  
  
 **elif** data == **'Invalid MAC.'**:  
 Log(LOG\_FILE\_PATH, **'MAC unaccepted. Retrying...'**, True)  
  
 **elif** data == **'MAC Accepted.'**:  
 Log(LOG\_FILE\_PATH, **'MAC accepted.'**)  
 **break** time.sleep(PAUSE\_INTERVAL)  
  
  
**def** OpenDatabase():  
 *""" Tells the server to open the database and verifies it was opened. """* **while** True:  
 sendEncrypted(sock, g\_AES\_Key, **'Open Database.'**)  
 data = recvDecrypted(sock, g\_AES\_Key)  
  
 **if not** data:  
 **return** ERR\_SERVER\_DOWN  
  
 **elif** data == **'Decryption Error.'**:  
 Log(LOG\_FILE\_PATH, **'Decryption error on server side. Retrying...'**, True)  
  
 **elif** data == **'Hash Check Failed.'**:  
 Log(LOG\_FILE\_PATH, **'Hash check failed on server side. Retrying...'**, True)  
  
 **elif** data == **'Database Error.'**:  
 **return** ERR\_DATABASE\_PROBLEM  
  
 **elif** data == **'Database Opened.'**:  
 Log(LOG\_FILE\_PATH, **'Database opened on server side.'**)  
 **break** time.sleep(PAUSE\_INTERVAL)  
  
  
**def** CreateProcStatsObj():  
 *""" Creates the ProcessStats object and notifies the server about it. """* **global** proc\_stats\_obj  
  
 **try**:  
 proc\_stats\_obj = ProcessStats()  
 Log(LOG\_FILE\_PATH, **'ProcessStats object created.'**)  
  
 sendEncrypted(sock, g\_AES\_Key, **'ProcessStats Object Created.'**)  
 data = recvDecrypted(sock, g\_AES\_Key)  
 **if not** data:  
 **return** ERR\_SERVER\_DOWN  
 **elif** data != **'Start Transmitting Data.'**:  
 Log(LOG\_FILE\_PATH, **'Unexpected reply received:'** + data, True)  
  
 **except** Exception **as** e:  
 Log(LOG\_FILE\_PATH, type(e).\_\_name\_\_ + **':'** + str(e), True)  
 sendEncrypted(sock, g\_AES\_Key, **'ProcessStats Object Creation Failed.'**)  
 **return** ERR\_OBJ\_NOT\_CREATED  
  
  
**def** MainCommunicationCycle(proc\_stats\_obj):  
 *"""  
 The main communication, which includes both:  
 Sending process stats data to the server,  
 Receiving commands from the server and executing them.  
 """* Log(LOG\_FILE\_PATH, **'Starting main communication.'**)  
 **while** True:  
 *# Get process stats and send them to the server* all\_process\_stats = proc\_stats\_obj.GetStats()  
  
 **while** True:  
 sendEncrypted(sock, g\_AES\_Key, pickle.dumps(all\_process\_stats))  
 data = recvDecrypted(sock, g\_AES\_Key)  
 **if not** data:  
 **return** ERR\_SERVER\_DOWN  
  
 **elif** data == **'Decryption Error.'**:  
 Log(LOG\_FILE\_PATH, **'Decryption error on server side. Retrying...'**, True)  
  
 **elif** data == **'Hash Check Failed.'**:  
 Log(LOG\_FILE\_PATH, **'Hash check failed on server side. Retrying...'**, True)  
  
 **elif** data == **'Data Obtained.'**:  
 Log(LOG\_FILE\_PATH, **'Data was successfully obtained by server.'**)  
 sendEncrypted(sock, g\_AES\_Key, **'Waiting for further instructions.'**)  
 **break  
  
 else**:  
 Log(LOG\_FILE\_PATH, **'Unexpected data received:'** + data, True)   
   
 time.sleep(PAUSE\_INTERVAL)  
  
*# Receive commands (Alert, Kill Process, Kill Process Tree, Shutdown)***while** True:  
 data = recvDecrypted(sock, g\_AES\_Key)  
 **if not** data:  
 **return** ERR\_SERVER\_DOWN  
  
 **elif** data == **'Decryption Error.'**:  
 Log(LOG\_FILE\_PATH, **'Decryption error on server side. Retrying...'**, True)  
  
 **elif** data == **'Hash Check Failed.'**:  
 Log(LOG\_FILE\_PATH, **'Hash check failed on server side. Retrying...'**, True)  
  
 **elif** COMMAND\_SEPARATOR **in** data:  
 args = data.split(COMMAND\_SEPARATOR)  
 command = args[0]  
  
 **if** command == KILL\_PROC\_COMMAND:  
 pid = int(args[1])  
 Log(LOG\_FILE\_PATH, **'Killing Process '** + str(pid))  
 result = KillProcess(pid)  
 Log(LOG\_FILE\_PATH, result)  
 sendEncrypted(sock, g\_AES\_Key, result)  
  
 **elif** command == KILL\_PROC\_TREE\_COMMAND:  
 pid = int(args[1])  
 Log(LOG\_FILE\_PATH, **'Killing Process Tree '** + str(pid))  
 result = KillProcessTree(pid)  
 Log(LOG\_FILE\_PATH, result)  
 sendEncrypted(sock, g\_AES\_Key, result)  
  
 **elif** command == SHUTDOWN\_COMMAND:  
 timeout, command = args[1], args[2]  
 Log(LOG\_FILE\_PATH, **'Shutting down in %s seconds.'** % timeout)  
 result = Shutdown(timeout, command)  
 Log(LOG\_FILE\_PATH, result)  
 sendEncrypted(sock, g\_AES\_Key, result)  
  
 **elif** data == **'Proceed.'**:  
 *# Exit the command receiving loop  
 # to send data to the server again* Log(LOG\_FILE\_PATH, **'Proceeding.'**)  
 **break  
  
 else**:  
 Log(LOG\_FILE\_PATH, **'Unexpected data received:'** + data, True)  
   
 time.sleep(PAUSE\_INTERVAL)  
   
time.sleep(PAUSE\_INTERVAL)  
  
  
**def** CloseConnection(errcode):  
 *""" Prints an error message and closes the connection with the server. """* **if** errcode == ERR\_SERVER\_DOWN:  
 Log(LOG\_FILE\_PATH, **'Error! Server is Down. Restarting...'**)  
 **elif** errcode == ERR\_DATABASE\_PROBLEM:  
 Log(LOG\_FILE\_PATH, **'Error opening the database. Restarting...'**)  
 **elif** errcode == ERR\_OBJ\_NOT\_CREATED:  
 Log(LOG\_FILE\_PATH, **'Could not create ProcessStats object. Restarting...'**)  
 **else**:  
 Log(LOG\_FILE\_PATH, **'Unknown Error: '** + str(errcode) + **'. Restarting...'**)  
 sock.close()  
  
  
**def** DoAllCommunication():  
 *"""  
 Runs sequentially all the functions used for communication with the server.  
 If an error occurs, it restarts (closes the connection and re-runs the loop).  
 """* **while** True:  
 StartConnection()  
  
 err = GetRSAKey()  
 **if** err:  
 CloseConnection(err)  
 **continue** err = SendAESKey(server\_RSA\_Key)  
 **if** err:  
 CloseConnection(err)  
 **continue** err = SendMACAddress()  
 **if** err:  
 CloseConnection(err)  
 **continue** err = OpenDatabase()  
 **if** err:  
 CloseConnection(err)  
 **continue** err = CreateProcStatsObj()  
 **if** err:  
 CloseConnection(err)  
 **continue** err = MainCommunicationCycle(proc\_stats\_obj)  
 **if** err:  
 CloseConnection(err)  
 **continue  
  
  
if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 DoAllCommunication()

**Server.py:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*# region IMPORTS***import** sys  
**import** os  
**import** time  
**import** socket  
**import** threading  
**import** pickle  
**from** string **import** hexdigits  
  
**try**:  
 **from** Crypto **import** Random  
 **from** Crypto.PublicKey **import** RSA  
 **from** Crypto.Hash **import** SHA256  
**except** ImportError:  
 **print 'PyCrypto module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
  
**try**:  
 **from** AES\_Encryption **import** AESCipher  
**except** ImportError:  
 **print 'AES\_Encryption module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
  
**try**:  
 **from** Send\_Receive **import** SendToGUI, Send, Receive, sendEncrypted, recvDecrypted  
**except** ImportError:  
 **print 'Send\_Receive module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
  
**try**:  
 **from** Sql **import** Database  
**except** ImportError:  
 **print 'Sql module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
  
**try**:  
 **from** Followed\_Action **import** LOG\_FILE\_PATH, Log, StatsAnalyzer  
**except** ImportError:  
 **print 'Followed\_Action module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
*# endregion  
  
  
# region CONFIG***def** LoadConfig():  
 *""" Loads the configuration from the file """* config\_file = None  
 **try**:  
 config\_file = open(**'config.txt'**)  
 server\_port = int(config\_file.readline().strip(**'\n'**)) *# read Server port from file* gui\_port = int(config\_file.readline().strip(**'\n'**)) *# read GUI port from file* **return** server\_port, gui\_port  
 **except**:  
 Log(LOG\_FILE\_PATH, **'Bad configuration file! Exiting...'**)  
 raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
 **finally**:  
 **if** config\_file:  
 config\_file.close() *# close the file when finished*config = LoadConfig()  
*# endregion  
# region CONSTANTS*HOST = **''** *# the current server host*PORT = config[0] *# the port to be opened by the server*ADDR = (HOST, PORT) *# the full address the server binds itself to*GUI\_HOST = **'127.0.0.1'** *# the HOST of the GUI (localhost)*GUI\_PORT = config[1] *# the PORT of the GUI*GUI\_ADDR = (GUI\_HOST, GUI\_PORT) *# the full ADDRESS of the GUI*RSA\_KEY\_SIZE = 2048 *# RSA key size*DATABASE\_DATA\_LIFESPAN = 3 *# the life span (in days) of data in the databases  
# endregion  
  
  
# region GLOBALS*g\_RSA\_Key = RSA.generate(RSA\_KEY\_SIZE, Random.new().read) *# Server RSA key*g\_RSA\_PublicKey = g\_RSA\_Key.publickey() *# RSA public key only*g\_GUI\_socket = socket.socket() *# the socket to connect with the GUI*g\_GUI\_sock\_lock = threading.Lock() *# lock to control access of threads to the GUI socket*g\_client\_thread\_list = [] *# a list of all client threads  
# endregion***class** ClientThread(threading.Thread):  
 *"""  
 Each client connecting to the server receives its own ClientThread instance.  
 Each such instance runs in a separate thread and stores relevant data about the client.  
 It is responsible for the communication with the client, receiving monitoring data and sending commands.  
 """* **def** \_\_init\_\_(self, ip, sock):  
 *"""  
 Initialize the thread and the client parameters (ip, source port & socket).  
 """* threading.Thread.\_\_init\_\_(self)  
  
 self.ip = ip *# client IP address* self.socket = sock *# socket to communicate with the client* self.key = None *# AES key for safe communication* self.MAC = None *# Client MAC address* self.DB = None *# Database object to access client's database* self.StatsAnalyzer = None *# StatsAnalyzer object to analyze process stats* Log(LOG\_FILE\_PATH, **'[+] '** + ip + **' connected.'**)  
  
  
 **def** run(self):  
 *"""  
 When the object's start() function is called, it starts the thread and calls this function.  
 It's the main function, which is responsible for the communication with the client.  
 It receives the process stats and stores them in the database.  
 """* **if not** self.BeginCommunication():  
 **return** -1  
  
 *# Create the StatsAnalyzer object* self.StatsAnalyzer = StatsAnalyzer(self.ip, g\_GUI\_socket, g\_GUI\_sock\_lock, self.socket, self.key, self.DB)  
  
 *# Start the actual communication* **return** self.MainCommunicationCycle()  
  
  
 **def** BeginCommunication(self):  
 *"""  
 Responsible for the beginning of the communication with the client.  
 Performs all the one-time preparatory actions before the actual data exchange.  
 Returns False if something went wrong (and connection with client was closed), and True otherwise.  
 """* **if not** self.StartConnection():  
 **return** False  
  
 *# Send server RSA key* **if not** self.SendRSAKey():  
 **return** False  
  
 *# Get client's AES key* **if not** self.GetAESKey():  
 **return** False  
  
 *# Get client's MAC address* **if not** self.GetMACAddress():  
 **return** False  
  
 *# Connect to the database* **if not** self.ConnectToDatabase():  
 **return** False  
  
 *# Verify ProcessStats object creation* **if not** self.VerifyProcessStatsObj():  
 **return** False  
  
 *# Connect to GUI* **if not** self.ConnectToGUI():  
 **return** False  
  
 **return** True  
  
  
 **def** MainCommunicationCycle(self):   
 *"""*  *The main communication with the client.*  *Receives process stats data and stores it in the database.*  *Analyzes the stats and sends commands to the client.*  *"""*  
 **while** True:  
 *# Receive process stats data* data = recvDecrypted(self.socket, self.key)  
 **if not** data:  
 sendEncrypted(self.socket, self.key, **'No Data. Goodbye.'**)  
 self.CloseConnection()  
 **return** -1  
  
 **elif** data == **'Decryption Error!'**:  
 sendEncrypted(self.socket, self.key, **'Decryption Error.'**)  
  
 **elif** data == **'Hash Check Error!'**:  
 sendEncrypted(self.socket, self.key, **'Hash Check Failed.'**)  
  
 **else**:  
 **try**:  
 *# Load the stats into a list using pickle* all\_process\_stats = pickle.loads(data)  
 sendEncrypted(self.socket, self.key, **'Data Obtained.'**)  
  
 *# Save the stats in the database* **for** proc\_stats **in** all\_process\_stats:  
 self.DB.Insert(\*\*proc\_stats)  
 self.DB.Save()  
  
 **except** Exception **as** e:  
 Log(LOG\_FILE\_PATH, type(e).\_\_name\_\_ + **':'** + str(e))  
 **continue** recvDecrypted(self.socket, self.key)  
  
 *# Find the \_Total process stats* **for** proc\_stats **in** all\_process\_stats:  
 **if** proc\_stats[**'Name'**] == **'\_Total'**:  
 total\_proc\_stats = proc\_stats  
 **break** *# Send the stats to GUI* CPU = str(total\_proc\_stats[**'PercentProcessorTime'**])  
 RAM = str(total\_proc\_stats[**'WorkingSetPrivate'**])  
 IO = str(total\_proc\_stats[**'IODataBytesPersec'**])  
 **if not** SendToGUI(g\_GUI\_socket, g\_GUI\_sock\_lock, self.ip, **'UsageData'**, CPU, RAM, IO):  
 *# If GUI is down close all active connections and exit* **for** client\_thread **in** g\_client\_thread\_list:  
 client\_thread.CloseConnection()  
 os.\_exit(1)  
  
 *# Analyze the stats using the StatsAnalyzer object* self.StatsAnalyzer.Analyze(total\_proc\_stats)  
  
 *# Tell client to proceed* sendEncrypted(self.socket, self.key, **'Proceed.'**)  
  
  
 **def** StartConnection(self):  
 *"""  
 Starts the connection with the client by receiving 'Hello Server.'  
 Returns False if didn't receive 'Hello Server.' from client,  
 Returns True otherwise.  
 """* data = Receive(self.socket)  
  
 **if** data != **'Hello Server.'**:  
 *# Communication must start with the message 'Hello Server.'* Send(self.socket, **'Goodbye.'**)  
 self.CloseConnection()  
 **return** False  
  
 **return** True  
  
  
 **def** SendRSAKey(self):  
 *"""  
 Sends the RSA key to the client until he accepts it.  
 Returns False if client disconnected,  
 Returns True if everything was successful.  
 """* **while** True:  
 Send(self.socket, pickle.dumps(g\_RSA\_PublicKey))  
 data = Receive(self.socket)  
  
 **if not** data:  
 *# No data was received (or client is down), so close the connection* Send(self.socket, **'No Data. Goodbye.'**)  
 self.CloseConnection()  
 **return** False  
  
 **elif** data == **'Bad Key.'**:  
 Log(LOG\_FILE\_PATH, **'RSA key unaccepted by client. Retrying...'**)  
  
 **elif** data == **'Key Accepted.'**:  
 **break  
  
 else**:  
 Log(LOG\_FILE\_PATH, **'Unexpected data received: '** + data, True)  
  
 Log(LOG\_FILE\_PATH, **'RSA key accepted.'**)  
 **return** True  
  
  
 **def** GetAESKey(self):  
 *"""  
 Gets the AES key from the client and verifies it.  
 Returns False if client disconnected,  
 Returns True if everything was successful.  
 """* Send(self.socket, **'Send AES Key.'**)  
  
 **while not** self.key:  
 data = recvDecrypted(self.socket, g\_RSA\_Key)  
 **if not** data:  
 Send(self.socket, **'No Data. Goodbye.'**)  
 self.CloseConnection()  
 **return** False  
  
 **elif** data == **'Decryption Error!'**:  
 Send(self.socket, **'Decryption Error.'**)  
  
 **elif** data == **'Hash Check Error!'**:  
 Send(self.socket, **'Hash Check Failed.'**)  
  
 **else**:  
 **try**:  
 self.key = pickle.loads(data) *# load key from data using pickle* **assert** isinstance(self.key, AESCipher) *# verify key is AESCipher object* Send(self.socket, **'Key Accepted. Send MAC Address.'**) *# No encryption because client uses Receive, since he doesn't know if key was accepted.* **except**:  
 *# If data was not a legal pickle dump OR key object wasn't an AES key* self.key = None  
 Send(self.socket, **'Bad key.'**)  
  
 Log(LOG\_FILE\_PATH, **'AES key obtained.'**)  
 **return** True  
  
  
 **def** GetMACAddress(self):  
 *"""  
 Gets the MAC address from the client and verifies it.  
 Returns False if client disconnected,  
 Returns True if everything was successful.  
 """* **while not** self.MAC:  
 data = recvDecrypted(self.socket, self.key)  
 **if not** data:  
 sendEncrypted(self.socket, self.key, **'No Data. Goodbye.'**)  
 self.CloseConnection()  
 **return** False  
  
 **elif** data == **'Decryption Error!'**:  
 sendEncrypted(self.socket, self.key, **'Decryption Error.'**)  
 **continue  
  
 elif** data == **'Hash Check Error!'**:  
 sendEncrypted(self.socket, self.key, **'Hash Check Failed.'**)  
 **continue** mac = data.upper()  
  
 *# Verify mac is a legal MAC Address string* **if** len(mac) != 17:  
 Log(LOG\_FILE\_PATH, **'Invalid MAC address!'**)  
 sendEncrypted(self.socket, self.key, **'Invalid MAC.'**)  
 **continue** valid = True  
 **for** i **in** range(17):  
 **if** (i % 3 == 2 **and** mac[i] != **':'**) **or** (i % 3 != 2 **and** mac[i] **not in** hexdigits):  
 Log(LOG\_FILE\_PATH, **'Invalid MAC address!'**)  
 sendEncrypted(self.socket, self.key, **'Invalid MAC.'**)  
 valid = False  
 **break  
 if not** valid:  
 **continue** *# After verification, tell client the MAC was accepted and store it for future use.* sendEncrypted(self.socket, self.key, **'MAC Accepted.'**)  
 self.MAC = mac  
 Log(LOG\_FILE\_PATH, **'MAC obtained.'**)  
 **return** True  
  
  
 **def** ConnectToDatabase(self):  
 *"""  
 Opens and connects to the client's database.  
 Returns False if couldn't connect to the database or client disconnected,  
 Returns True if everything was successful.  
 """* **while** True:  
 data = recvDecrypted(self.socket, self.key)  
 **if not** data:  
 sendEncrypted(self.socket, self.key, **'No Data. Goodbye.'**)  
 self.CloseConnection()  
 **return** False  
  
 **elif** data == **'Decryption Error!'**:  
 sendEncrypted(self.socket, self.key, **'Decryption Error.'**)  
  
 **elif** data == **'Hash Check Error!'**:  
 sendEncrypted(self.socket, self.key, **'Hash Check Failed.'**)  
  
 **elif** data == **'Open Database.'**:  
 **try**:  
 db\_name = SHA256.new(**'database\_'** + self.MAC).hexdigest()[:32] + **'.db'** self.DB = Database(db\_name, DATABASE\_DATA\_LIFESPAN)  
 self.DB.CreateTable()  
 Log(LOG\_FILE\_PATH, **'Connected to database.'**)  
 sendEncrypted(self.socket, self.key, **'Database Opened.'**)  
 **return** True  
  
 **except** Exception **as** e:  
 Log(LOG\_FILE\_PATH, type(e).\_\_name\_\_ + **':'** + str(e), True)  
 Log(LOG\_FILE\_PATH, **'Could not open/create the database. Terminating connection...'**)  
 sendEncrypted(self.socket, self.key, **'Database Error.'**)  
 self.CloseConnection()  
 **return** False  
  
 **else**:  
 Log(LOG\_FILE\_PATH, **'Unexpected data received: '** + data, True)  
  
  
 **def** VerifyProcessStatsObj(self):  
 *"""  
 Verifies ProcessStats object was created successfully by client.  
 Returns False if client couldn't create the ProcessStats object or he disconnected,  
 Returns True if everything was successful.  
 """* data = recvDecrypted(self.socket, self.key)  
 **if not** data:  
 sendEncrypted(self.socket, self.key, **'No Data. Goodbye.'**)  
 self.CloseConnection()  
 **return** False  
  
 **elif** data == **'Decryption Error!'**:  
 sendEncrypted(self.socket, self.key, **'Decryption Error.'**)  
 **return** False  
  
 **elif** data == **'Hash Check Error!'**:  
 sendEncrypted(self.socket, self.key, **'Hash Check Failed.'**)  
 **return** False  
  
 **elif** data == **'ProcessStats Object Creation Failed.'**:  
 Log(LOG\_FILE\_PATH, **'Error! Client could not create ProcessStats object. Terminating connection.'**)  
 self.CloseConnection()  
 **return** False  
  
 **elif** data == **'ProcessStats Object Created.'**:  
 sendEncrypted(self.socket, self.key, **'Start Transmitting Data.'**)  
 **return** True  
  
 **else**:  
 Log(LOG\_FILE\_PATH, **'Unexpected data received: '** + data, True)  
 **return** False  
  
  
 **def** ConnectToGUI(self):  
 *"""  
 Connects the client thread to GUI.  
 Returns False if couldn't connect to GUI,  
 Returns True if everything was successful.  
 """* **if not** SendToGUI(g\_GUI\_socket, g\_GUI\_sock\_lock, self.ip, **'ClientConnected'**):  
 self.CloseConnection()  
 **return** False  
  
 Log(LOG\_FILE\_PATH, **'Connected to GUI.'**)  
 **return** True  
  
  
 **def** CloseConnection(self):  
 *"""  
 Closes the connection with client.  
 Closes the database and the socket, notifies GUI and exits the thread.  
 """* **if** self.DB:  
 self.DB.Close()  
 self.socket.close()  
 SendToGUI(g\_GUI\_socket, g\_GUI\_sock\_lock, self.ip, **'ClientDisconnected'**)  
 Log(LOG\_FILE\_PATH, **'[-] '** + self.ip + **' disconnected.'**)  
 sys.exit(1)  
  
  
  
g\_GUI\_socket.connect(GUI\_ADDR)  
  
listening\_socket = socket.socket()  
listening\_socket.bind(ADDR)  
listening\_socket.listen(5)  
  
Log(LOG\_FILE\_PATH, **'Server is ready.'**)  
  
**while** True:  
 clientSock, (ip, port) = listening\_socket.accept()  
 newThread = ClientThread(ip, clientSock)  
 newThread.start()  
  
listening\_socket.close()  
g\_GUI\_socket.close()

**Process\_Stats.py:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*# region IMPORTS***import** sys  
**import** time  
  
**try**:  
 **import** win32com.client  
**except** ImportError:  
 **print 'PyWin32 module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
*# endregion***class** ProcessStats:  
 *"""  
 ProcessStats provides stats and information about the processes running on the machine.  
 It uses the WMI to receive all data and organizes it in a list of dictionaries.  
 The last dictionary contains data about the TOTAL usage of resources.  
 """* **def** \_\_init\_\_(self):  
 *"""  
 Initializes the process attributes list and the WMI service.  
 Defines constants: total amount of RAM and amount of logical cores.  
 """* self.process\_attrs = [  
 **'Name'**,  
 **'IDProcess'**,  
 **'CreatingProcessID'**,  
 **'ElapsedTime'**,  
 **'PercentProcessorTime'**,  
 **'WorkingSetPrivate'**,  
 **'IODataBytesPersec'**,  
 **'IODataOperationsPersec'**,  
 **'ThreadCount'** ]  
  
 objWMIService = win32com.client.Dispatch(**'WbemScripting.SWbemLocator'**)  
 self.objSWbemServices = objWMIService.ConnectServer(**'.'**, **'root\cimv2'**)  
  
 query\_mem\_total = **'SELECT Capacity FROM Win32\_PhysicalMemory'** installed\_memory = self.objSWbemServices.ExecQuery(query\_mem\_total)  
 self.RAM\_TOTAL = sum(int(memory.Capacity) **for** memory **in** installed\_memory)  
  
 query\_logical\_cores = **'SELECT NumberOfLogicalProcessors FROM Win32\_ComputerSystem'** self.LOGICAL\_CORES = self.objSWbemServices.ExecQuery(query\_logical\_cores)[0].NumberOfLogicalProcessors  
  
  
 **def** GetStats(self):  
 *"""  
 Show process stats for all processes running on the machine.  
 Returns a list of results (a result list for each process).  
 """* proc\_results\_list = []  
  
 query\_proc = **'SELECT \* FROM Win32\_PerfFormattedData\_PerfProc\_Process'** Processes = self.objSWbemServices.ExecQuery(query\_proc)  
  
 query\_cpu = **'SELECT \* FROM Win32\_PerfFormattedData\_Counters\_ProcessorInformation WHERE Name="\_Total"'** total\_cpu\_usage = int(self.objSWbemServices.ExecQuery(query\_cpu)[0].PercentProcessorTime)  
  
 query\_mem\_avail = **'SELECT AvailableBytes FROM Win32\_PerfFormattedData\_PerfOS\_Memory'** available\_memory = float(self.objSWbemServices.ExecQuery(query\_mem\_avail)[0].AvailableBytes)  
  
 **for** Process **in** Processes:  
  
 **if** Process.Name == **'\_Global'**: *# Exclude '\_Global' process if exists* **continue** this\_proc\_dict = {}  
  
 **for** attribute **in** self.process\_attrs:  
  
 **if** Process.Name == **'\_Total'**:  
 **if** attribute == **'PercentProcessorTime'**: *# Insert total CPU usage percentage* this\_proc\_dict[attribute] = total\_cpu\_usage  
 **elif** attribute == **'WorkingSetPrivate'**: *# Insert physical memory usage percentage* this\_proc\_dict[attribute] = int(100 \* (1 - available\_memory / self.RAM\_TOTAL))  
 **else**:  
 eval\_str = **'Process.'** + attribute  
 this\_proc\_dict[attribute] = eval(eval\_str)  
  
 **else**:  
 **if** attribute == **'PercentProcessorTime'**: *# Divide by the amount of logical cores* this\_proc\_dict[attribute] = int(Process.PercentProcessorTime) / self.LOGICAL\_CORES  
 **else**:  
 eval\_str = **'Process.'** + attribute  
 this\_proc\_dict[attribute] = eval(eval\_str)  
  
 this\_proc\_dict[**'TimeStamp'**] = time.strftime(**'%Y-%m-%d %H:%M:%S'**) *# Add time stamp* proc\_results\_list.append(this\_proc\_dict)  
  
 **return** proc\_results\_list

**Process\_Kill.py:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*# region IMPORTS***import** sys  
**import** os  
  
**try**:  
 **import** win32api  
**except** ImportError:  
 **print 'PyWin32 module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
*# endregion  
  
  
# region CONSTANTS*PROCESS\_TERMINATE = 1 *# OpenProcess mode that allows process termination  
# endregion  
  
  
# region GLOBALS*g\_CurrentProcessId = win32api.GetCurrentProcessId() *# PID of the python process it will run on  
# endregion***def** KillProcess(PID):  
 *"""  
 Kills a process by given Process ID.* **:return***: Error description if command failed, 'Success' otherwise.  
 """* **if** PID == g\_CurrentProcessId:  
 **return 'Error: attempted to kill client process.'  
 try**:  
 handle = win32api.OpenProcess(PROCESS\_TERMINATE, False, PID)  
 win32api.TerminateProcess(handle, -1)  
 win32api.CloseHandle(handle)  
 **return 'Success.'  
 except** Exception **as** e:  
 **return 'Exception occurred: '** + str(e)  
  
  
**def** KillProcessTree(PID):  
 *"""  
 Kills a process by given Process ID and kills its entire process tree.  
 (All processes created by this process, processes created by them, etc.)* **:return***: Error description if command failed, 'Success' otherwise.  
 """* **if** PID == g\_CurrentProcessId:  
 **return 'Error: attempted to kill client process tree.'  
 try**:  
 handle = win32api.OpenProcess(PROCESS\_TERMINATE, True, PID)  
 win32api.TerminateProcess(handle, -1)  
 win32api.CloseHandle(handle)  
 **return 'Success.'  
 except** Exception **as** e:  
 **return 'Exception occurred: '** + str(e)  
  
  
**def** Shutdown(timeout, comment):  
 *"""  
 Schedules system shutdown.* **:param** *timeout: the timeout in seconds before the shutdown.* **:param** *comment: a message for the user describing the reason for shutdown.* **:return***: Error description if command failed, 'Success' otherwise.  
 """* **try**:  
 result = os.system(**'shutdown /s /t %s /c "%s"'** % (timeout, comment))  
 **if** result:  
 **return 'Error in shutdown command. Error code %s.'** % result  
 **else**:  
 **return 'Success.'  
 except** Exception **as** e:  
 **return 'Exception occurred: '** + str(e)

**Sql.py:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**import** sqlite3  
**import** datetime  
  
  
*# region CONSTANTS*TABLE\_NAME = **'ResourcesData'** *# the name of the (single) table in the database  
# endregion***class** Database:  
 *"""  
 The Database object creates or opens a database in the specified file.  
 It allows manipulation of the database, such as:  
 Create or Drop the table, Insert or Retrieve data, Delete old entries.  
 """* **def** \_\_init\_\_(self, db\_name, data\_lifespan):  
 *"""  
 Creates a database (if it doesn't exist already).  
 Creates a cursor object to execute all SQL queries.* **:param** *db\_name: The name of the file that stores the database.* **:param** *data\_lifespan: Determines how many days the entries stay in the database.  
 """* self.data\_lifespan = data\_lifespan  
 self.DB = sqlite3.connect(db\_name)  
 self.CURSOR = self.DB.cursor()  
 self.CURSOR.execute(**'PRAGMA auto\_vacuum = FULL'**)  
  
  
 **def** Save(self, delete=True):  
 *"""  
 Saves changes to the database.* **:param** *delete: Determines whether or not to delete old entries from the database.  
 """* **if** delete:  
 DateTimeObj = datetime.datetime.now().date() - datetime.timedelta(days=self.data\_lifespan)  
 DateThreshold = str(DateTimeObj)  
 self.DeleteOldData(DateThreshold)  
 self.DB.commit()  
  
  
 **def** Close(self):  
 *"""  
 Closes the connection to the database.  
 """* self.DB.close()  
  
  
 **def** CreateTable(self):  
 *"""  
 Create the ResourceStats table if it doesn't exist already.  
 """* create\_table\_query = **"CREATE TABLE IF NOT EXISTS "** + TABLE\_NAME + **"(ProcName VARCHAR(255), ProcId INT, \  
 CreatingProcId INT, ElapsedTime INT, CpuTimePercent INT, WorkingSetPrivate BIGINT, \  
 IOBytesPersec BIGINT, IOOperationsPersec BIGINT, Threads INT, Date DATE, Time TIME)"** self.CURSOR.execute(create\_table\_query)  
 self.Save(False) *# make sure changes are saved!* **def** Insert(self, Name, IDProcess, CreatingProcessID, ElapsedTime, PercentProcessorTime, WorkingSetPrivate,  
 IODataBytesPersec, IODataOperationsPersec, ThreadCount, TimeStamp):  
 *"""  
 Insert the given parameters into the table.  
 """* insert\_query = **"INSERT INTO "** + TABLE\_NAME + **" VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"** Date, Time = TimeStamp.split(**' '**)  
 params = [Name, IDProcess, CreatingProcessID, ElapsedTime, PercentProcessorTime, WorkingSetPrivate,  
 IODataBytesPersec, IODataOperationsPersec, ThreadCount, Date, Time]  
 self.CURSOR.execute(insert\_query, params)  
  
  
 **def** Retrieve(self):  
 *"""  
 A generator that retrieves all data from the database.  
 The data is sorted from the most to the least recent.  
 Creates an iterator of the rows in the table.  
 """* retrieve\_query = **"SELECT \* FROM "** + TABLE\_NAME + **" ORDER BY Date DESC, Time DESC"** self.CURSOR.execute(retrieve\_query)  
 **while** True:  
 row = self.CURSOR.fetchone() *# eventually it will raise StopIteration* proc\_stats = {**'Name'**: row[0],  
 **'IDProcess'**: row[1],  
 **'CreatingProcessID'**: row[2],  
 **'ElapsedTime'**: row[3],  
 **'PercentProcessorTime'**: row[4],  
 **'WorkingSetPrivate'**: row[5],  
 **'IODataBytesPersec'**: row[6],  
 **'IODataOperationsPersec'**: row[7],  
 **'ThreadCount'**: row[8],  
 **'TimeStamp'**: row[9] + **' '** + row[10]}  
 **yield** proc\_stats  
  
  
 **def** DeleteOldData(self, DateThreshold):  
 *"""  
 Deletes all entries associated with a certain computer whose Time Stamp is very old.  
 Threshold should be set to the date from e.g. a day or a week ago.  
 """* delete\_query = **"DELETE FROM "** + TABLE\_NAME + **" WHERE Date < ?"** params = [DateThreshold]  
 self.CURSOR.execute(delete\_query, params)

**Followed\_Action.py:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*# region IMPORTS***import** sys  
**import** time  
**import** xml.etree.ElementTree **as** ET  
**import** smtplib  
**from** email.mime.text **import** MIMEText  
  
**try**:  
 **from** AES\_Encryption **import** AESCipher  
**except** ImportError:  
 **print 'AES\_Encryption module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
  
**try**:  
 **from** Send\_Receive **import** SendToGUI, sendEncrypted, recvDecrypted  
**except** ImportError:  
 **print 'Send\_Receive module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
  
**try**:  
 **from** Sql **import** Database  
**except** ImportError:  
 **print 'Sql module not found.'** raw\_input(**'Press Enter to exit...'**)  
 sys.exit(1)  
*# endregion***def** LoadEmails():  
 *""" Loads the IT email addresses from the file """* emails = []  
 emails\_file = None  
 **try**:  
 emails\_file = open(**'emails.txt'**)  
 email = **'email'  
 while** email:  
 email = emails\_file.readline() *# read email address from file* emails.append(email.strip(**'\n'**)) *# add to list* **return** emails  
 **except**:  
 Log(LOG\_FILE\_PATH, **'Bad emails file! Sending email to IT feature will not be available.'**)  
 **return** []  
 **finally**:  
 **if** emails\_file:  
 emails\_file.close() *# close the file when finished  
  
  
# region CONSTANTS*SETTINGS\_FILE\_PATH = **'settings.xml'** *# the path of the settings file*LOG\_FILE\_PATH = **'log.txt'** *# the path of the log file*SMTP\_SERVER = **'smtp.gmail.com:587'** *# SMTP server address (I'm using Gmail)*MY\_EMAIL\_ADDRESS = **'CRMU2015@gmail.com'** *# the email address used to send emails to IT*MY\_USERNAME = **'CRMU2015'** *# the username of the email address*MY\_PASSWORD = **'acfppsss'** *# the password of the email address*IT\_EMAIL\_ADDRESSES = LoadEmails() *# a list of email addresses of company's IT support*COMMA\_SPACE\_SEPARATOR = **', '** *# separator between email addresses*MAX\_BYTES\_VALUE = 2\*\*60 *# maximum value for I/O or Memory Bytes*MAX\_IOPS\_VALUE = 10\*\*7 *# maximum value for I/O Operations Per Sec*MAX\_THREADS\_VALUE = 10\*\*5 *# maximum value for Threads Count*COMMAND\_SEPARATOR = **' && '** *# separator between command and params*KILL\_PROC\_PREFIX = **'Kill Process'** + COMMAND\_SEPARATOR *# prefix for Kill Process command*KILL\_PROC\_TREE\_PREFIX = **'Kill Process Tree'** + COMMAND\_SEPARATOR *# prefix for Kill Process Tree command*SHUTDOWN\_PREFIX = **'Shutdown'** + COMMAND\_SEPARATOR *# prefix for Shutdown command*NEW\_LINE = **'\r\n'** *# new line string (for emails)*UNITS\_CONVERSION\_DICT = {**'KB'**: 1024, **'MB'**: 1024\*\*2, **'GB'**: 1024\*\*3} *# a dictionary for data units conversion*RESOURCES\_MATCH = [ *# a tuple list to match resource names to* (**'CPU Usage'**, **'PercentProcessorTime'**), *# their names in the proc\_stats dictionary* (**'Memory Usage'**, **'WorkingSetPrivate'**),  
 (**'IO Data'**, **'IODataBytesPersec'**),  
 (**'IOPS'**, **'IODataOperationsPersec'**),  
 (**'Threads'**, **'ThreadCount'**)  
]  
  
SYS\_PROCESSES = [**'Idle'**, **'System'**, **'csrss'**, **'lsass'**, **'smss'**, **'spoolsv'**, *# list of system* **'svchost'**, **'services'**, **'wininit'**, **'winlogon'**, **'explorer'**] *# processes*  
*# endregion***def** Log(FilePath, string, add\_caller=False):  
 *"""  
 Writes to the log file provided the file path, and also prints the string.  
 Adds time stamp and caller function name (if add\_caller is set to True).  
 """* caller\_str = **'In '** + sys.\_getframe().f\_back.f\_code.co\_name + **' - '  
 if** add\_caller:  
 string = caller\_str + string  
 time\_stamp = **'['** + time.strftime(**'%d/%m/%Y %H:%M:%S'**) + **'] '** f = open(FilePath, **'a'**)  
 f.write(time\_stamp + string + **'\n'**)  
 f.close()  
 **print** string  
  
  
**class** StatsAnalyzer:  
 *"""  
 Analyzes process stats according to the preset rules.  
 Executes followed actions when necessary.  
 An instance is to be created for each client separately.  
 """* **def** \_\_init\_\_(self, ip, gui\_socket, gui\_socket\_lock, client\_socket, AES\_key, DB):  
 *"""  
 Initializes the class fields.  
 """* self.ip = ip  
 self.gui\_socket = gui\_socket  
 self.gui\_socket\_lock = gui\_socket\_lock  
 self.client\_socket = client\_socket  
 self.AES\_key = AES\_key  
 self.DB = DB  
  
  
 **def** Analyze(self, global\_stats):  
 *"""  
 Loads the rules set by admin from the XML settings file.  
 Receives global system stats and analyzes them according to the rules.  
 If they match any rule, it reads from the database complete process stats (most recent).  
 It then analyzes them according to the sub-rules of the matching rules.  
 When there is a match, it executes the corresponding followed action.  
 """* rules = LoadRulesFromXML(SETTINGS\_FILE\_PATH)  
  
 *# Find the rules that match the global stats* matching\_rules = []  
 **for** rule **in** rules:  
 **if** CheckStatsAgainstRule(global\_stats, rule):  
 matching\_rules.append(rule)  
  
 *# If no matches were found, exit* **if not** matching\_rules:  
 **return** None  
  
 *# Retrieve most recent process stats entries from the database* all\_stats = []  
 **for** proc\_stats **in** self.DB.Retrieve():  
 **if** proc\_stats[**'Name'**] == **'\_Total'**: *# \_Total is always the last process so we know we're done* **break** all\_stats.append(proc\_stats)  
  
 *# For every process and every sub-rule of a matching rule, check for violations and execute followed action* **for** proc\_stats **in** all\_stats:  
 Ignore = False  
 **if** proc\_stats[**'Name'**] **in** SYS\_PROCESSES:  
 Ignore = True  
 **else**:  
 **for** sys\_proc **in** SYS\_PROCESSES:  
 **if** sys\_proc.startswith(proc\_stats[**'Name'**]+**'#'**):  
 Ignore = True  
 **break  
  
 if** Ignore: *# ignore windows system processes* **continue  
  
 for** rule **in** matching\_rules:  
 **for** subrule **in** rule[**'subrules'**]:  
 **if** CheckStatsAgainstSubrule(proc\_stats, subrule):  
 self.FollowedAction(subrule[**'Action'**], proc\_stats)  
  
  
 **def** FollowedAction(self, action, proc\_stats):  
 *"""  
 Executes the followed action using the provided parameters.* **:param** *action: the action to be executed.* **:param** *proc\_stats: the stats of the relevant process.  
 """* Log(LOG\_FILE\_PATH, **'Executing followed action "'** + action + **'" on process stats '** + str(proc\_stats))  
  
 *# Get stats from proc\_stats* Name = proc\_stats[**'Name'**]  
 pid = str(proc\_stats[**'IDProcess'**])  
 CreatingPID = str(proc\_stats[**'CreatingProcessID'**])  
 ElapsedTime = str(proc\_stats[**'ElapsedTime'**])  
 Cpu = str(proc\_stats[**'PercentProcessorTime'**])  
 Ram = str(proc\_stats[**'WorkingSetPrivate'**])  
 IODataBytes = str(proc\_stats[**'IODataBytesPersec'**])  
 IOps = str(proc\_stats[**'IODataOperationsPersec'**])  
 Threads = str(proc\_stats[**'ThreadCount'**])  
  
 **if 'Alert' in** action:  
 self.SendAlertToGUI(proc\_stats)  
  
 **if 'Kill Process Tree' in** action:  
 *# Send command to client and get response* kill\_proc\_tree\_msg = KILL\_PROC\_TREE\_PREFIX + pid  
 sendEncrypted(self.client\_socket, self.AES\_key, kill\_proc\_tree\_msg)  
 result = recvDecrypted(self.client\_socket, self.AES\_key)  
  
 *# Send result message to GUI* msg\_prefix = Name + **' (PID '** + pid + **'): '  
 if** result == **'Success.'**:  
 msg = msg\_prefix + **'process tree killed successfully.'  
 else**:  
 msg = msg\_prefix + **'process tree killing failed. '** + result  
 SendToGUI(self.gui\_socket, self.gui\_socket\_lock, self.ip, **'Log'**, msg)  
  
 **elif 'Kill Process' in** action:  
 *# Send command to client and get response* kill\_proc\_msg = KILL\_PROC\_PREFIX + pid  
 sendEncrypted(self.client\_socket, self.AES\_key, kill\_proc\_msg)  
 result = recvDecrypted(self.client\_socket, self.AES\_key)  
  
 *# Send result message to GUI* msg\_prefix = Name + **' (PID '** + pid + **'): '  
 if** result == **'Success.'**:  
 msg = msg\_prefix + **'process killed successfully.'  
 else**:  
 msg = msg\_prefix + **'process killing failed. '** + result  
 SendToGUI(self.gui\_socket, self.gui\_socket\_lock, self.ip, **'Log'**, msg)  
  
 **if 'Shutdown' in** action:  
 *# Set the shutdown parameters* timeout = 120 *# shutdown timeout in seconds* comment = **'Shutting down because process %s (PID %s) is causing trouble.'** % (Name, pid)  
  
 *# Send command to client and get response* shutdown\_msg = SHUTDOWN\_PREFIX + str(timeout) + COMMAND\_SEPARATOR + comment  
 sendEncrypted(self.client\_socket, self.AES\_key, shutdown\_msg)  
 result = recvDecrypted(self.client\_socket, self.AES\_key)  
  
 *# Send result message to GUI* **if** result == **'Success.'**:  
 msg = **'Shut down command successful. Shutting down in %s seconds.'** % timeout  
 **else**:  
 msg = **'Shutting down command failed. '** + result  
 SendToGUI(self.gui\_socket, self.gui\_socket\_lock, self.ip, **'Log'**, msg)  
  
 **if 'Send Email to IT' in** action:  
 *# Convert process stats units* Memory\_units = **'MB'** Memory = str(int(float(Ram) / UNITS\_CONVERSION\_DICT[Memory\_units]))  
 IOData\_units = **'KB'** IOData = str(int(float(IODataBytes) / UNITS\_CONVERSION\_DICT[IOData\_units]))  
  
 *# Define the message parameters* msg\_from = MY\_EMAIL\_ADDRESS  
 msg\_to = IT\_EMAIL\_ADDRESSES  
 msg\_subject = **'[CRM] Suspicious Usage Data'** msg\_body = **'Details:'** + NEW\_LINE\*2  
 msg\_body += **'IP Address: '** + self.ip + NEW\_LINE  
 msg\_body += **'Time: '** + time.strftime(**'%H:%M:%S'**) + NEW\_LINE\*2  
 msg\_body += **'Process Name: '** + Name + NEW\_LINE  
 msg\_body += **'PID: '** + pid + NEW\_LINE  
 msg\_body += **'Creating Process ID: '** + CreatingPID + NEW\_LINE  
 msg\_body += **'Elapsed Time: '** + ElapsedTime + **' seconds'** + NEW\_LINE\*2  
 msg\_body += **'CPU Usage: '** + Cpu + **'%'** + NEW\_LINE  
 msg\_body += **'Memory Usage: '** + Memory + Memory\_units + NEW\_LINE  
 msg\_body += **'I/O Data Per Sec: '** + IOData + IOData\_units + NEW\_LINE  
 msg\_body += **'IOPS: '** + IOps + NEW\_LINE  
 msg\_body += **'Threads: '** + Threads + NEW\_LINE  
  
 *# Create the email message* msg = MIMEText(msg\_body)  
 msg[**'Subject'**] = msg\_subject  
 msg[**'From'**] = msg\_from  
 msg[**'To'**] = COMMA\_SPACE\_SEPARATOR.join(msg\_to)  
  
 *# Send the email message* result = SendEmail(msg)  
  
 *# Send result message to GUI* msg\_prefix = Name + **' (PID '** + pid + **'): '  
 if** result == **'Success.'**:  
 msg = msg\_prefix + **'Email sent successfully to IT.'  
 else**:  
 msg = msg\_prefix + **'Email sending failed. '** + result  
 SendToGUI(self.gui\_socket, self.gui\_socket\_lock, self.ip, **'Log'**, msg)  
  
  
 **def** SendAlertToGUI(self, proc\_stats):  
 *"""  
 Sends alert message details to GUI given the process stats.  
 """* params = [proc\_stats[**'Name'**] + **' ('** + str(proc\_stats[**'IDProcess'**]) + **')'**]  
 **for** resource\_name, attribute **in** RESOURCES\_MATCH:  
 resource\_data = proc\_stats[attribute]  
 **if** resource\_name == **'CPU Usage'**:  
 resource\_data = str(resource\_data) + **'%'  
 elif** resource\_name **in** [**'Memory Usage'**, **'IO Data'**]:  
 units = **'MB'** resource\_data = int(float(resource\_data) / UNITS\_CONVERSION\_DICT[units] \* 10 + 0.5) / 10.0  
 resource\_data = str(resource\_data) + units  
 **else**:  
 resource\_data = str(resource\_data)  
 params.append(resource\_name + **': '** + resource\_data)  
 SendToGUI(self.gui\_socket, self.gui\_socket\_lock, self.ip, **'Alert'**, \*params)  
  
  
**def** LoadRulesFromXML(FilePath):  
 *"""  
 Loads rules from XML settings file.* **:param** *FilePath: the path of the XML file.* **:return***: a list containing all rules loaded.  
 """* Rules = []  
  
 tree = ET.parse(FilePath)  
 root = tree.getroot()  
  
 **for** rule **in** root:  
 SubRules = []  
  
 **for** subrule **in** rule:  
 Whitelist = []  
 **for** process **in** subrule.find(**'whitelist'**):  
 Whitelist.append(process.get(**'name'**))  
  
 Blacklist = []  
 **for** process **in** subrule.find(**'blacklist'**):  
 Blacklist.append(process.get(**'name'**))  
  
 subrule\_dict = subrule.attrib.copy()  
 subrule\_dict.update({**'whitelist'**: Whitelist, **'blacklist'**: Blacklist})  
 SubRules.append(subrule\_dict)  
  
 rule\_dict = rule.attrib.copy()  
 rule\_dict.update({**'subrules'**: SubRules})  
 Rules.append(rule\_dict)  
  
 **return** Rules  
  
  
**def** GetBytesFromStr(string):  
 *"""  
 Converts a string which consists of a number and a suffix 'KB', 'MB' or 'GB' to its value in bytes.  
 Returns -1 in case of an invalid string.  
 """* **try**:  
 val = int(string[:-2])  
 conversion\_multiple = UNITS\_CONVERSION\_DICT[string[-2:]]  
 **return** val \* conversion\_multiple  
 **except** (ValueError, KeyError):  
 **return** -1  
**def** CheckStatsAgainstRule(stats, rule):  
 *"""  
 Checks whether the stats satisfy the rule.  
 Returns True/False accordingly.  
 """* stats\_CPU = int(stats[**'PercentProcessorTime'**])  
 stats\_RAM = int(stats[**'WorkingSetPrivate'**])  
 stats\_IO = int(stats[**'IODataBytesPersec'**])  
  
 rule\_CPUmin = int(rule[**'CPUmin'**])  
 rule\_CPUmax = int(rule[**'CPUmax'**])  
 rule\_RAMmin = int(rule[**'RAMmin'**])  
 rule\_RAMmax = int(rule[**'RAMmax'**])  
 rule\_IOmin = GetBytesFromStr(rule[**'IOmin'**])  
 rule\_IOmax = GetBytesFromStr(rule[**'IOmax'**])  
  
 **if** rule\_IOmin == -1:  
 rule\_IOmin = 0  
 **if** rule\_IOmax == -1:  
 rule\_IOmax = MAX\_BYTES\_VALUE  
  
 **if not** rule\_CPUmin <= stats\_CPU <= rule\_CPUmax:  
 **return** False  
 **if not** rule\_RAMmin <= stats\_RAM <= rule\_RAMmax:  
 **return** False  
 **if not** rule\_IOmin <= stats\_IO <= rule\_IOmax:  
 **return** False  
  
 **return** True  
  
  
**def** CheckStatsAgainstSubrule(proc\_stats, subrule):  
 *"""  
 Checks whether proc\_stats satisfy the sub-rule.  
 Returns True/False accordingly.  
 """* stats\_CPU = int(proc\_stats[**'PercentProcessorTime'**])  
 stats\_RAM = int(proc\_stats[**'WorkingSetPrivate'**])  
 stats\_IO = int(proc\_stats[**'IODataBytesPersec'**])  
 stats\_IOPS = int(proc\_stats[**'IODataOperationsPersec'**])  
 stats\_Threads = int(proc\_stats[**'ThreadCount'**])  
  
 subrule\_CPUmin = int(subrule[**'CPUmin'**])  
 subrule\_CPUmax = int(subrule[**'CPUmax'**])  
 subrule\_RAMmin = GetBytesFromStr(subrule[**'RAMmin'**])  
 subrule\_RAMmax = GetBytesFromStr(subrule[**'RAMmax'**])  
 subrule\_IOmin = GetBytesFromStr(subrule[**'IOmin'**])  
 subrule\_IOmax = GetBytesFromStr(subrule[**'IOmax'**])  
 subrule\_IOPSmin = subrule[**'IOPSmin'**]  
 subrule\_IOPSmax = subrule[**'IOPSmax'**]  
 subrule\_ThreadsMin = subrule[**'ThreadsMin'**]  
 subrule\_ThreadsMax = subrule[**'ThreadsMax'**]  
  
 *# Fix sub-rule values where necessary* **if** subrule\_RAMmin == -1:  
 subrule\_RAMmin = 0  
 **if** subrule\_RAMmax == -1:  
 subrule\_RAMmax = MAX\_BYTES\_VALUE  
  
 **if** subrule\_IOmin == -1:  
 subrule\_IOmin = 0  
 **else**:  
 subrule\_IOmin = int(subrule\_IOmin)  
 **if** subrule\_IOmax == -1:  
 subrule\_IOmax = MAX\_BYTES\_VALUE  
 **else**:  
 subrule\_IOmax = int(subrule\_IOmax)  
  
 **if** subrule\_IOPSmin == **''**:  
 subrule\_IOPSmin = 0  
 **else**:  
 subrule\_IOPSmin = int(subrule\_IOPSmin)  
 **if** subrule\_IOPSmax == **''**:  
 subrule\_IOPSmax = MAX\_IOPS\_VALUE  
 **else**:  
 subrule\_IOPSmax = int(subrule\_IOPSmax)  
  
 **if** subrule\_ThreadsMin == **''**:  
 subrule\_ThreadsMin = 0  
 **else**:  
 subrule\_ThreadsMin = int(subrule\_ThreadsMin)  
 **if** subrule\_ThreadsMax == **''**:  
 subrule\_ThreadsMax = MAX\_THREADS\_VALUE  
 **else**:  
 subrule\_ThreadsMax = int(subrule\_ThreadsMax)  
  
 *# Check for mismatches of stats with sub-rule* **if not** subrule\_CPUmin <= stats\_CPU <= subrule\_CPUmax:  
 **return** False  
 **if not** subrule\_RAMmin <= stats\_RAM <= subrule\_RAMmax:  
 **return** False  
 **if not** subrule\_IOmin <= stats\_IO <= subrule\_IOmax:  
 **return** False  
 **if not** subrule\_IOPSmin <= stats\_IOPS <= subrule\_IOPSmax:  
 **return** False  
 **if not** subrule\_ThreadsMin <= stats\_Threads <= subrule\_ThreadsMax:  
 **return** False  
  
 *# Check whitelist/blacklist* **if** subrule[**'Type'**] == **'whitelist'**:  
 **for** procName **in** subrule[**'whitelist'**]:  
 **if** proc\_stats[**'Name'**].startswith(procName):  
 **return** False  
 **else**:  
 found = False  
 **for** procName **in** subrule[**'blacklist'**]:  
 **if** proc\_stats[**'Name'**].startswith(procName):  
 found = True  
 **break  
 if not** found:  
 **return** False  
  
 **return** True  
  
  
**def** SendEmail(msg):  
 *"""  
 Sends an email using Gmail's SMTP server.* **:param** *msg: the message to be sent.* **:return***: whether email sending was successful or not.  
 """* **try**:  
 server = smtplib.SMTP(SMTP\_SERVER)  
 server.ehlo()  
 server.starttls()  
 server.login(MY\_USERNAME, MY\_PASSWORD)  
 server.sendmail(msg[**'From'**], msg[**'To'**].split(COMMA\_SPACE\_SEPARATOR), msg.as\_string())  
 server.quit()  
 **return 'Success.'  
 except** Exception **as** e:  
 **return** type(e).\_\_name\_\_ + **':'**, e

**MainForm.cs:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Diagnostics;

using System.Xml;

using System.IO;

namespace Computer\_Resources\_Monitoring

{

public partial class MainForm : Form

{

#region Data

private const string ActionSeparator = " + ";

private const string PythonEnginePath = "Server.py";

private const string SettingsFilePath = @"Python\settings.xml";

private const string DefaultSettingsFilePath = @"Python\default.xml";

public PythonListener pythonListener;

private Process pythonEngineProcess;

#endregion

#region Constructor

public MainForm()

{

InitializeComponent();

}

#endregion

#region Python Process

/// <summary>

/// Starts the python engine

/// </summary>

public void StartPythonEngine()

{

try

{

// Start python engine

pythonEngineProcess = new Process();

pythonEngineProcess.StartInfo.FileName = @"C:\Python27\python.exe";

pythonEngineProcess.StartInfo.Arguments = PythonEnginePath;

pythonEngineProcess.StartInfo.WorkingDirectory = AppDomain.CurrentDomain.BaseDirectory + @"Python\";

pythonEngineProcess.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;

pythonEngineProcess.Start();

pythonListener = new PythonListener(this);

}

catch (Win32Exception e)

{

EndProcess(e, "Win32 Exception");

}

catch (ObjectDisposedException e)

{

EndProcess(e, "Object Disposed Exception");

}

catch (InvalidOperationException e)

{

EndProcess(e, "Invalid Operation Exception");

}

catch (Exception e)

{

EndProcess(e, "Unknown Type Exception");

}

}

/// <summary>

/// Ends the python process (in case of an exception)

/// </summary>

private void EndProcess(Exception e, string ExceptionType)

{

PrintToLog("[StartPythonEngine] " + ExceptionType + ": " + e.Message, Color.Red);

StopPythonEngine();

}

/// <summary>

/// Stops the python engine

/// </summary>

public void StopPythonEngine()

{

try

{

if (pythonEngineProcess != null)

{

pythonEngineProcess.Kill();

pythonEngineProcess.Dispose();

pythonEngineProcess = null;

}

if (pythonListener != null)

{

pythonListener.Close();

pythonListener = null;

}

this.Invoke((MethodInvoker)delegate

{

WaitingServerLabel.Hide();

ServerRunningLabel.Hide();

ClientsConnectedLabel.Hide();

ServerDownLabel.Show();

});

}

catch (Win32Exception e)

{

PrintToLog("[StopPythonEngine] Win32 Exception: " + e.Message, Color.Red);

pythonEngineProcess = null;

pythonListener = null;

}

catch (InvalidOperationException e)

{

PrintToLog("[StopPythonEngine] Invalid Operation Exception: " + e.Message, Color.Red);

pythonEngineProcess = null;

pythonListener = null;

}

catch (Exception e)

{

PrintToLog("[StopPythonEngine] Unknown Type Exception: " + e.Message, Color.Red);

pythonEngineProcess = null;

pythonListener = null;

}

}

#endregion

#region Update

/// <summary>

/// This method prints log to GUI

/// </summary>

/// <param name="msg">message to print</param>

/// <param name="color">color in view</param>

public void PrintToLog(string msg, Color color)

{

this.Invoke((MethodInvoker)delegate

{

logBox.SelectionColor = color;

this.logBox.AppendText(msg + Environment.NewLine);

this.logBox.ScrollToCaret();

});

}

/// <summary>

/// Returns the ClientControl corresponding to the given IP address

/// </summary>

public ClientControl GetClientControl(string ip)

{

foreach (ClientControl client in ClientsPanel.Controls)

{

if (client.IpLabel.Text.Equals(ip))

{

return client;

}

}

return null;

}

/// <summary>

/// Connects a client with given IP address

/// Creates and adds a new ClientControl in case it doesn't already exist

/// </summary>

public void ConnectClient(string ip)

{

this.Invoke((MethodInvoker)delegate

{

ClientControl client = GetClientControl(ip);

if (client == null)

{

client = new ClientControl(ip, this);

ClientsPanel.Controls.Add(client);

}

else

{

client.ComputerPicture.Image = Image.FromFile(@"..\..\Resources\Computer.png");

}

client.connected = true;

PrintToLog("[+] " + ip + " connected.", Color.Green);

ClientsCountLabel.Text = (int.Parse(ClientsCountLabel.Text) + 1).ToString();

});

}

/// <summary>

/// Disconnects a client by given IP address

/// </summary>

public void DisconnectClient(string ip)

{

this.Invoke((MethodInvoker)delegate

{

ClientControl client = GetClientControl(ip);

if (client != null && client.connected)

{

client.connected = false;

client.ComputerPicture.Image = Image.FromFile(@"..\..\Resources\ComputerDisconnected.png");

client.CPUbar.Value = 0;

client.RAMbar.Value = 0;

client.IODataLabel.Text = "--------------------";

ClientsCountLabel.Text = (int.Parse(ClientsCountLabel.Text) - 1).ToString();

PrintToLog("[-] " + ip + " disconnected.", Color.DarkRed);

}

});

}

#endregion

#region Load Rules From XML

/// <summary>

/// Loads rules from the XML settings file given the file path

/// </summary>

private void LoadRulesFromXML(string FilePath)

{

XmlDocument xmlDoc = new XmlDocument();

// Load XML file into xmlDoc

try

{

xmlDoc.Load(FilePath);

}

catch (XmlException e)

{

PrintToLog("Xml Exception: " + e.Message, Color.Red);

return;

}

catch (FileNotFoundException e)

{

PrintToLog("File Not Found Exception: " + e.Message, Color.Red);

return;

}

catch (DirectoryNotFoundException e)

{

PrintToLog("Directory Not Found Exception: " + e.Message, Color.Red);

return;

}

catch (UnauthorizedAccessException e)

{

PrintToLog("Unauthorized Access Exception: " + e.Message, Color.Red);

return;

}

catch (Exception e)

{

PrintToLog("Unknown Type Exception: " + e.Message, Color.Red);

return;

}

// Delete all RuleControls in RulesPanel

for (int i = RulesPanel.Controls.Count - 1; i >= 0; i--)

{

RuleControl control = RulesPanel.Controls[i] as RuleControl;

if (control != null)

control.Dispose();

}

// Reset maxID

RuleControl.maxID = 0;

SubRuleControl.maxID = 0;

// Traverse XML file

foreach (XmlNode RuleNode in xmlDoc.DocumentElement.ChildNodes)

{

int gID = int.Parse(RuleNode.Attributes["ID"].Value);

string gName = RuleNode.Attributes["Name"].Value;

string gCPUmin = RuleNode.Attributes["CPUmin"].Value;

string gCPUmax = RuleNode.Attributes["CPUmax"].Value;

string gRAMmin = RuleNode.Attributes["RAMmin"].Value;

string gRAMmax = RuleNode.Attributes["RAMmax"].Value;

string gIOmin = RuleNode.Attributes["IOmin"].Value;

string gIOminValue = gIOmin.Substring(0, gIOmin.Length - 2);

string gIOminUnits = gIOmin.Substring(gIOmin.Length - 2);

string gIOmax = RuleNode.Attributes["IOmax"].Value;

string gIOmaxValue = gIOmax.Substring(0, gIOmax.Length - 2);

string gIOmaxUnits = gIOmax.Substring(gIOmax.Length - 2);

List<SubRule> subrules = new List<SubRule>();

foreach (XmlNode SubRuleNode in RuleNode.ChildNodes)

{

int ID = int.Parse(SubRuleNode.Attributes["ID"].Value);

string CPUmin = SubRuleNode.Attributes["CPUmin"].Value;

string CPUmax = SubRuleNode.Attributes["CPUmax"].Value;

string RAMmin = SubRuleNode.Attributes["RAMmin"].Value;

string RAMminValue = RAMmin.Substring(0, RAMmin.Length - 2);

string RAMminUnits = RAMmin.Substring(RAMmin.Length - 2);

string RAMmax = SubRuleNode.Attributes["RAMmax"].Value;

string RAMmaxValue = RAMmax.Substring(0, RAMmax.Length - 2);

string RAMmaxUnits = RAMmax.Substring(RAMmax.Length - 2);

string IOmin = SubRuleNode.Attributes["IOmin"].Value;

string IOminValue = IOmin.Substring(0, IOmin.Length - 2);

string IOminUnits = IOmin.Substring(IOmin.Length - 2);

string IOmax = SubRuleNode.Attributes["IOmax"].Value;

string IOmaxValue = IOmax.Substring(0, IOmax.Length - 2);

string IOmaxUnits = IOmax.Substring(IOmax.Length - 2);

string IOPSmin = SubRuleNode.Attributes["IOPSmin"].Value;

string IOPSmax = SubRuleNode.Attributes["IOPSmax"].Value;

string ThreadsMin = SubRuleNode.Attributes["ThreadsMin"].Value;

string ThreadsMax = SubRuleNode.Attributes["ThreadsMax"].Value;

string[] Actions = SubRuleNode.Attributes["Action"].Value.Split(new string[] { ActionSeparator }, StringSplitOptions.None);

string type = SubRuleNode.Attributes["Type"].Value;

XmlNode Whitelist = SubRuleNode.SelectSingleNode("whitelist");

List<string> whitelist = new List<string>();

foreach (XmlNode ProcessNode in Whitelist.ChildNodes)

{

whitelist.Add(ProcessNode.Attributes["name"].Value);

}

XmlNode Blacklist = SubRuleNode.SelectSingleNode("blacklist");

List<string> blacklist = new List<string>();

foreach (XmlNode ProcessNode in Blacklist.ChildNodes)

{

blacklist.Add(ProcessNode.Attributes["name"].Value);

}

// Create the sub-rule and add it to the list

SubRule subrule = new SubRule(ID, CPUmin, CPUmax, RAMminValue, RAMminUnits, RAMmaxValue, RAMmaxUnits, IOminValue, IOminUnits, IOmaxValue, IOmaxUnits, IOPSmin, IOPSmax, ThreadsMin, ThreadsMax, Actions[0], Actions[1], type, whitelist, blacklist);

subrules.Add(subrule);

}

// Create the rule and the RuleControl, and add it to the RulesPanel

Rule rule = new Rule(gID, gName, gCPUmin, gCPUmax, gRAMmin, gRAMmax, gIOminValue, gIOminUnits, gIOmaxValue, gIOmaxUnits, subrules);

RuleControl newRule = new RuleControl(this, rule);

RulesPanel.Controls.Add(newRule);

newRule.BringToFront();

}

PrintToLog("Rules successfully loaded from " + FilePath, Color.Blue);

}

#endregion

#region Save Rules To XML

/// <summary>

/// Saves rules to the XML settings file given the file path

/// </summary>

private void SaveRulesToXML(string FilePath)

{

// Configure indentation settings

XmlWriterSettings XmlSettings = new XmlWriterSettings();

XmlSettings.Indent = true;

XmlSettings.IndentChars = "\t";

// Create the writer and start writing

XmlWriter writer = XmlWriter.Create(FilePath, XmlSettings);

writer.WriteStartDocument();

writer.WriteStartElement("settings");

// Go over all rules and write them (turns out reverse order is the right one to preserve rules order)

for (int i = RulesPanel.Controls.Count - 1; i >= 0; i--)

{

// Obtain the rule

RuleControl ruleControl = RulesPanel.Controls[i] as RuleControl;

if (ruleControl == null)

continue;

Rule rule = ruleControl.rule;

// Write rule

writer.WriteStartElement("rule");

writer.WriteAttributeString("ID", rule.ID.ToString());

writer.WriteAttributeString("Name", rule.Name);

writer.WriteAttributeString("CPUmin", rule.CPUmin);

writer.WriteAttributeString("CPUmax", rule.CPUmax);

writer.WriteAttributeString("RAMmin", rule.RAMmin);

writer.WriteAttributeString("RAMmax", rule.RAMmax);

writer.WriteAttributeString("IOmin", rule.IOminValue + rule.IOminUnits);

writer.WriteAttributeString("IOmax", rule.IOmaxValue + rule.IOmaxUnits);

// Go over all sub-rules and write them (again, reverse order turns out to be the right one)

for (int j = ruleControl.SubRulesPanel.Controls.Count - 1; j >= 0; j--)

{

// Obtain the sub-rule

SubRuleControl subruleControl = ruleControl.SubRulesPanel.Controls[j] as SubRuleControl;

if (subruleControl == null)

continue;

SubRule subrule = subruleControl.subrule;

// Write sub-rule

writer.WriteStartElement("subrule");

writer.WriteAttributeString("ID", subrule.ID.ToString());

writer.WriteAttributeString("CPUmin", subrule.CPUmin);

writer.WriteAttributeString("CPUmax", subrule.CPUmax);

writer.WriteAttributeString("RAMmin", subrule.RAMminValue + subrule.RAMminUnits);

writer.WriteAttributeString("RAMmax", subrule.RAMmaxValue + subrule.RAMmaxUnits);

writer.WriteAttributeString("IOmin", subrule.IOminValue + subrule.IOminUnits);

writer.WriteAttributeString("IOmax", subrule.IOmaxValue + subrule.IOmaxUnits);

writer.WriteAttributeString("IOPSmin", subrule.IOPSmin);

writer.WriteAttributeString("IOPSmax", subrule.IOPSmax);

writer.WriteAttributeString("ThreadsMin", subrule.ThreadsMin);

writer.WriteAttributeString("ThreadsMax", subrule.ThreadsMax);

writer.WriteAttributeString("Action", subrule.Action1 + ActionSeparator + subrule.Action2);

writer.WriteAttributeString("Type", subrule.Type);

// Write whitelist

writer.WriteStartElement("whitelist");

foreach (string procName in subrule.Whitelist)

{

writer.WriteStartElement("process");

writer.WriteAttributeString("name", procName);

writer.WriteEndElement();

}

writer.WriteEndElement();

// Write blacklist

writer.WriteStartElement("blacklist");

foreach (string procName in subrule.Blacklist)

{

writer.WriteStartElement("process");

writer.WriteAttributeString("name", procName);

writer.WriteEndElement();

}

writer.WriteEndElement();

// End of sub-rule

writer.WriteEndElement();

}

// End of rule

writer.WriteEndElement();

}

// End of XML document

writer.WriteEndDocument();

writer.Close();

PrintToLog("Rules successfully saved to " + FilePath, Color.Blue);

}

#endregion

#region Events

private void MainForm\_Load(object sender, EventArgs e)

{

StartPythonEngine();

LoadRulesFromXML(SettingsFilePath);

}

private void MainForm\_Closing(object sender, FormClosingEventArgs e)

{

StopPythonEngine();

}

private void AddRule\_Click(object sender, EventArgs e)

{

RuleControl newRule = new RuleControl(this);

RulesPanel.Controls.Add(newRule);

newRule.BringToFront();

}

private void RestoreDefault\_Click(object sender, EventArgs e)

{

DialogResult result = MessageBox.Show("Are you sure you want to restore default settings?", "Restore Default Settings", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result.ToString() == "Yes")

{

LoadRulesFromXML(DefaultSettingsFilePath);

}

}

private void Reset\_Click(object sender, EventArgs e)

{

DialogResult result = MessageBox.Show("Are you sure you want to reset to last saved settings?", "Reset To Last Saved Settings", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result.ToString() == "Yes")

{

LoadRulesFromXML(SettingsFilePath);

}

}

private void Save\_Click(object sender, EventArgs e)

{

SaveRulesToXML(SettingsFilePath);

}

private void Help\_Click(object sender, EventArgs e)

{

string HelpMsg = "";

HelpMsg += "The software settings consist of rules.\n";

HelpMsg += "\n";

HelpMsg += "Rules define the global scenario in which they are applicable.\n";

HelpMsg += "They allow to specify the global usage stats which activate them.\n";

HelpMsg += "Rules consist of sub-rules.\n";

HelpMsg += "\n";

HelpMsg += "Sub-rules define the usage parameters of specific processes.\n";

HelpMsg += "In addition, they define the actions to be taken whenever a process fits the parameters set by the sub-rule.\n";

HelpMsg += "They also allow to define a whitelist / blacklist of processes.\n";

HelpMsg += "Sub-rules apply only when the rule containing them is activated.\n";

HelpMsg += "\n";

HelpMsg += "Don't forget to save changes to the settings using the button at the top!";

MessageBox.Show(HelpMsg, "Help", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

#endregion

}

}

**PythonListener.cs:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Net;

using System.Net.Sockets;

using System.IO;

using System.IO.Pipes;

using System.Threading;

using System.Diagnostics;

using System.Windows.Forms;

using System.Drawing;

namespace Computer\_Resources\_Monitoring

{

public class PythonListener

{

#region Constants

const string LOCALHOST = "127.0.0.1"; // localhost IP

readonly string[] END\_OF\_MSG = new string[] { "###" }; // end of message string in the protocol

readonly string[] SEPARATOR = new string[] { "$$$" }; // separator string in the protocol

#endregion

#region Data

MainForm mainForm; // reference to the main form

TcpListener listener; // listening socket

TcpClient pythonClient; // client (python engine)

Thread pythonListenerThread; // thread for the communication session

bool running; // flag to indicate whether the listener is running

int port; // port number

#endregion

/// <summary>

/// Constructor, initializes the reference to main form and the listener, and starts the thread

/// </summary>

public PythonListener(MainForm mainForm)

{

bool config\_ok = false;

this.mainForm = mainForm;

try

{

port = int.Parse(File.ReadAllLines(@"Python\config.txt")[1]);

config\_ok = true;

}

catch

{

mainForm.PrintToLog("Bad configuration file! Can't connect to engine.", Color.Red);

mainForm.StopPythonEngine();

config\_ok = false;

}

if (config\_ok)

{

listener = new TcpListener(IPAddress.Parse(LOCALHOST), port);

pythonListenerThread = new Thread(new ThreadStart(Run));

pythonListenerThread.Start();

}

}

/// <summary>

/// Initialize the listener and wait for connection with python engine

/// </summary>

private void Run()

{

try

{

// Start listening and wait for a connection

listener.Start();

listener.BeginAcceptTcpClient(new AsyncCallback(AcceptTcpClient\_Callback), new object());

}

catch (Exception e)

{

mainForm.PrintToLog("Initializing listener to connect with engine failed: " + e.Message, Color.Red);

Close();

}

}

/// <summary>

/// Connect with python engine and begin receiving data

/// </summary>

public void AcceptTcpClient\_Callback(IAsyncResult ar)

{

// Accept connection and stop listening

pythonClient = listener.EndAcceptTcpClient(ar);

listener.Stop();

running = true;

mainForm.Invoke((MethodInvoker)delegate

{

mainForm.WaitingServerLabel.Hide();

mainForm.ServerRunningLabel.Show();

mainForm.ClientsConnectedLabel.Show();

mainForm.ClientsCountLabel.Show();

});

// Begin to asynchronously receive data from the socket using a StateObject

StateObject stateObject = new StateObject();

stateObject.workSocket = pythonClient.Client;

pythonClient.Client.BeginReceive(stateObject.buffer, 0, StateObject.BufferSize, 0, new AsyncCallback(Read\_Callback), stateObject);

}

/// <summary>

/// Asynchronously read (receive data) from input stream of connected socket

/// </summary>

/// <param name="ar">Stores state information and any user defined data for this asynchronous operation</param>

public void Read\_Callback(IAsyncResult ar)

{

try

{

StateObject stateObject = (StateObject)ar.AsyncState;

if (pythonClient != null && running)

{

// Ends a pending asynchronous read

int read = stateObject.workSocket.EndReceive(ar);

if (read > 0)

{

string data = Encoding.ASCII.GetString(stateObject.buffer, 0, read); // Read data

string[] msgs = data.Split(END\_OF\_MSG, StringSplitOptions.RemoveEmptyEntries);

foreach (string msg in msgs)

{

string[] items = msg.Split(SEPARATOR, StringSplitOptions.RemoveEmptyEntries);

string ClientIP = items[0];

ClientControl client = mainForm.GetClientControl(ClientIP);

switch (items[1]) // command

{

case "ClientConnected":

mainForm.ConnectClient(ClientIP);

break;

case "ClientDisconnected":

mainForm.DisconnectClient(ClientIP);

break;

case "Log":

mainForm.PrintToLog("[" + ClientIP + "] " + items[2], Color.OrangeRed);

break;

case "UsageData":

int CPU\_usage = int.Parse(items[2]);

int RAM\_usage = int.Parse(items[3]);

int IO\_bytes = int.Parse(items[4]);

int IO\_usage\_KB = IO\_bytes / 1024;

mainForm.Invoke((MethodInvoker)delegate

{

if (client != null)

{

client.CPUbar.Value = CPU\_usage;

client.RAMbar.Value = RAM\_usage;

client.IODataLabel.Text = IO\_usage\_KB + " KB";

}

});

break;

case "Alert":

mainForm.Invoke((MethodInvoker)delegate

{

if (client != null)

{

mainForm.tabControl.SelectTab("Monitor");

string[] ResourceStats = new string[items.Length - 2];

for (int i = 2; i < items.Length; i++)

ResourceStats[i - 2] = items[i];

client.ShowAlert(ResourceStats);

}

});

break;

}

}

}

if (running)

// Continue receiving new data

stateObject.workSocket.BeginReceive(stateObject.buffer, 0, StateObject.BufferSize, 0, new AsyncCallback(Read\_Callback), stateObject);

}

}

catch(Exception e)

{

mainForm.PrintToLog("Exception during data reading: " + e.Message, Color.Red);

if (pythonClient == null || !pythonClient.Connected)

Close();

}

}

/// <summary>

/// Close the connection and stop the current thread

/// </summary>

public void Close()

{

if (pythonClient != null)

{

pythonClient.Close();

pythonClient = null;

foreach (ClientControl client in mainForm.ClientsPanel.Controls)

{

mainForm.DisconnectClient(client.IpLabel.Text);

}

}

running = false;

if (pythonListenerThread.IsAlive)

pythonListenerThread.Abort();

mainForm.Invoke((MethodInvoker)delegate

{

mainForm.ServerRunningLabel.Hide();

mainForm.ClientsConnectedLabel.Hide();

mainForm.ClientsCountLabel.Hide();

mainForm.ServerDownLabel.Show();

});

mainForm.PrintToLog("[" + DateTime.Now.ToShortTimeString() + "] Server shut down.", Color.DarkViolet);

}

}

/// <summary>

/// State object for asynchronous data reading

/// </summary>

public class StateObject

{

public Socket workSocket = null; // Socket to read data from

public const int BufferSize = 1024; // Size of receive buffer

public byte[] buffer = new byte[BufferSize]; // Receive buffer

public StringBuilder sb = new StringBuilder(); // Received data string

}

}

**ClientControl.cs:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Data;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace Computer\_Resources\_Monitoring

{

public partial class ClientControl : UserControl

{

private string ip;

private int index;

public bool connected;

private MainForm mainForm;

/// <summary>

/// Constructor, initializes the control parameters

/// </summary>

public ClientControl(string ip, MainForm form)

{

InitializeComponent();

this.mainForm = form;

this.ip = ip;

this.index = mainForm.ClientsPanel.Controls.Count;

this.IpLabel.Text = ip;

this.Location = new Point(this.index \* (Width + 50) + 5, 10);

this.connected = true;

}

/// <summary>

/// Shows an alert in the AlertsPanel according to the data supplied

/// </summary>

/// <param name="ResourceStats">The data to be displayed in the alert box</param>

public void ShowAlert(string[] ResourceStats)

{

AlertsLabel.Show();

if (AlertsPanel.Controls.Count > 0)

ClearAlerts.Show();

foreach (Panel panel in AlertsPanel.Controls)

{

panel.Location = new Point(panel.Location.X, panel.Location.Y + 155);

}

Panel AlertBoxPanel = new Panel();

AlertBoxPanel.BackgroundImage = Image.FromFile(@"..\..\Resources\AlertBox.png");

AlertBoxPanel.BackgroundImageLayout = ImageLayout.Stretch;

AlertBoxPanel.Size = new Size(193, 150);

AlertBoxPanel.Location = new Point(0, 0);

AlertBoxPanel.BackColor = Color.Transparent; // a trick to make transparent labels

for (int i = 0; i < ResourceStats.Length; i++)

{

string ResourceData = ResourceStats[i];

Label label = new Label();

label.Text = ResourceData;

label.AutoSize = true;

if (i == 0)

{

label.Font = new Font("Microsoft Sans Serif", (float)10, FontStyle.Bold | FontStyle.Underline);

label.ForeColor = Color.Red;

label.Location = new Point(10, 10);

}

else

{

label.Font = new Font("Microsoft Sans Serif", (float)8.5);

label.Location = new Point(10, 21 \* i + 15);

}

AlertBoxPanel.Controls.Add(label);

}

AlertsPanel.Controls.Add(AlertBoxPanel);

}

#region Events

private void ClearAlerts\_Click(object sender, EventArgs e)

{

AlertsLabel.Hide();

// Delete all alerts in AlertsPanel

for (int i = AlertsPanel.Controls.Count - 1; i >= 0; i--)

{

Control control = AlertsPanel.Controls[i];

control.Dispose();

}

ClearAlerts.Hide();

}

#endregion

}

}

**RuleControl.cs:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Collections;

namespace Computer\_Resources\_Monitoring

{

public partial class RuleControl : UserControl

{

public int index;

private MainForm mainForm;

public Rule rule;

public static int maxID = 0;

#region Constructors

/// <summary>

/// Constructor, creates a new RuleControl with default values

/// </summary>

/// <param name="form">The MainForm containing this RuleControl</param>

public RuleControl(MainForm form)

{

InitializeComponent();

this.mainForm = form;

this.index = 0;

foreach (Control control in mainForm.RulesPanel.Controls)

{

if (control.GetType() == typeof(RuleControl))

this.index++;

}

this.Position();

this.rule = new Rule(++maxID, "Rule #" + maxID.ToString(), "00", "100", "00", "100", "", "KB", "", "MB", new List<SubRule>());

}

/// <summary>

/// Constructor, creates a new RuleControl according to the values supplied

/// </summary>

/// <param name="form">The MainForm containing this RuleControl</param>

/// <param name="rule">the rule defining the values for this RuleControl</param>

public RuleControl(MainForm form, Rule rule)

{

InitializeComponent();

this.mainForm = form;

this.index = 0;

foreach (Control control in mainForm.RulesPanel.Controls)

{

if (control.GetType() == typeof(RuleControl))

this.index++;

}

this.Position();

this.rule = rule;

if (rule.ID > maxID) // update max ID if necessary

maxID = rule.ID;

}

#endregion

/// <summary>

/// Positions the control in the correct location according to its index

/// </summary>

public void Position()

{

this.Location = new Point(0, this.index \* (Height + 3) + mainForm.RulesPanel.AutoScrollPosition.Y);

}

#region Events

private void RuleControl\_Load(object sender, EventArgs e)

{

RuleNameLabel.Text = rule.Name;

RuleNameTextBox.Text = rule.Name;

gCPUmin.SelectedItem = rule.CPUmin;

gCPUmax.SelectedItem = rule.CPUmax;

gRAMmin.Text = rule.RAMmin;

gRAMmax.Text = rule.RAMmax;

gIOminValue.Text = rule.IOminValue;

gIOminUnits.SelectedItem = rule.IOminUnits;

gIOmaxValue.Text = rule.IOmaxValue;

gIOmaxUnits.SelectedItem = rule.IOmaxUnits;

foreach (SubRule subrule in rule.subrules)

{

SubRuleControl newSubrule = new SubRuleControl(this, subrule);

this.SubRulesPanel.Controls.Add(newSubrule);

newSubrule.BringToFront();

}

}

private void DeleteRuleX\_Click(object sender, EventArgs e)

{

mainForm.RulesPanel.Controls.Remove(this);

// Move up the RuleControls from below

foreach (Control control in mainForm.RulesPanel.Controls)

{

RuleControl ruleControl = control as RuleControl;

if (ruleControl != null)

{

if (ruleControl.index > this.index)

{

ruleControl.index--;

ruleControl.Position();

}

}

}

// Update maxID

if (rule.ID == maxID)

{

maxID = 0;

foreach (Control control in mainForm.RulesPanel.Controls)

{

RuleControl ruleControl = control as RuleControl;

if (ruleControl != null)

{

if (ruleControl.rule.ID > maxID)

maxID = ruleControl.rule.ID;

}

}

}

this.Dispose();

}

private void AddSubrule\_Click(object sender, EventArgs e)

{

SubRuleControl newSubrule = new SubRuleControl(this);

this.SubRulesPanel.Controls.Add(newSubrule);

newSubrule.BringToFront();

this.rule.subrules.Add(newSubrule.subrule);

}

private void DeleteAll\_Click(object sender, EventArgs e)

{

DialogResult result = MessageBox.Show("Are you sure you want to delete all sub-rules?", "Delete All Sub-rules", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result.ToString() == "Yes")

{

// Find all SubRuleControls in SubRulesPanel and delete them

List<Control> SubRuleControls = new List<Control>();

foreach (Control control in SubRulesPanel.Controls)

{

if (control.GetType() == typeof(SubRuleControl))

{

SubRuleControls.Add(control);

}

}

foreach (Control control in SubRuleControls)

{

SubRulesPanel.Controls.Remove(control);

control.Dispose();

}

// Reset maxID

SubRuleControl.maxID = 0;

// Clear the subrules list

rule.subrules.Clear();

}

}

private void RuleNameLabel\_Click(object sender, EventArgs e)

{

RuleNameTextBox.Show();

RuleNameTextBox.Focus();

RuleNameTextBox.SelectionStart = RuleNameTextBox.Text.Length;

RuleNameTextBox.DeselectAll();

RuleNameLabel.Hide();

}

private void RuleNameTextBox\_Leave(object sender, EventArgs e)

{

rule.Name = RuleNameTextBox.Text;

RuleNameLabel.Text = RuleNameTextBox.Text;

RuleNameLabel.Show();

RuleNameTextBox.Hide();

}

private void gCPUmin\_SelectedIndexChanged(object sender, EventArgs e)

{

rule.CPUmin = gCPUmin.SelectedItem.ToString();

}

private void gCPUmax\_SelectedIndexChanged(object sender, EventArgs e)

{

rule.CPUmax = gCPUmax.SelectedItem.ToString();

}

private void gRAMmin\_SelectedIndexChanged(object sender, EventArgs e)

{

rule.RAMmin = gRAMmin.SelectedItem.ToString();

}

private void gRAMmax\_SelectedIndexChanged(object sender, EventArgs e)

{

rule.RAMmax = gRAMmax.SelectedItem.ToString();

}

private void gIOminValue\_Leave(object sender, EventArgs e)

{

double NumIOminValue;

if (double.TryParse(gIOminValue.Text, out NumIOminValue))

rule.IOminValue = NumIOminValue.ToString();

else if (gIOminValue.Text == "")

rule.IOminValue = "";

else

{

MessageBox.Show("Field must be a number!", "Invalid Value", MessageBoxButtons.OK, MessageBoxIcon.Warning);

gIOminValue.Focus();

gIOminValue.SelectAll();

}

}

private void gIOminUnits\_SelectedIndexChanged(object sender, EventArgs e)

{

rule.IOminUnits = gIOminUnits.SelectedItem.ToString();

}

private void gIOmaxValue\_Leave(object sender, EventArgs e)

{

double NumIOmaxValue;

if (double.TryParse(gIOmaxValue.Text, out NumIOmaxValue))

rule.IOmaxValue = NumIOmaxValue.ToString();

else if (gIOmaxValue.Text == "")

rule.IOmaxValue = "";

else

{

MessageBox.Show("Field must be a number!", "Invalid Value", MessageBoxButtons.OK, MessageBoxIcon.Warning);

gIOmaxValue.Focus();

gIOmaxValue.SelectAll();

}

}

private void gIOmaxUnits\_SelectedIndexChanged(object sender, EventArgs e)

{

rule.IOmaxUnits = gIOmaxUnits.SelectedItem.ToString();

}

#endregion

}

}

**Rule.cs:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Computer\_Resources\_Monitoring

{

public class Rule

{

public int ID;

public string Name;

public string CPUmin;

public string CPUmax;

public string RAMmin;

public string RAMmax;

public string IOminValue;

public string IOminUnits;

public string IOmaxValue;

public string IOmaxUnits;

public List<SubRule> subrules;

public Rule(int ID, string Name, string CPUmin, string CPUmax, string RAMmin, string RAMmax, string IOminValue, string IOminUnits, string IOmaxValue, string IOmaxUnits, List<SubRule> subrules)

{

this.ID = ID;

this.Name = Name;

this.CPUmin = CPUmin;

this.CPUmax = CPUmax;

this.RAMmin = RAMmin;

this.RAMmax = RAMmax;

this.IOminValue = IOminValue;

this.IOminUnits = IOminUnits;

this.IOmaxValue = IOmaxValue;

this.IOmaxUnits = IOmaxUnits;

this.subrules = subrules;

}

}

}

**SubRuleControl.cs:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Computer\_Resources\_Monitoring

{

public partial class SubRuleControl : UserControl

{

public int index;

private RuleControl ContainingRule;

public SubRule subrule;

public static int maxID = 0;

#region Constructors

/// <summary>

/// Constructor, creates a new SubRuleControl with default values

/// </summary>

/// <param name="containingRule">The RuleControl containing this SubRuleControl</param>

public SubRuleControl(RuleControl container)

{

InitializeComponent();

this.ContainingRule = container;

this.index = 0;

foreach (Control control in ContainingRule.SubRulesPanel.Controls)

{

if (control.GetType() == typeof(SubRuleControl))

this.index++;

}

this.Position();

this.subrule = new SubRule(++maxID, "00", "100", "", "MB", "", "GB", "", "KB", "", "MB", "", "", "", "", "None", "None", "whitelist", new List<string>(), new List<string>());

}

/// <summary>

/// Constructor, creates a new SubRuleControl according to the values supplied

/// </summary>

/// <param name="containingRule">The RuleControl containing this SubRuleControl</param>

/// <param name="subrule">the sub-rule defining the values for this SubRuleControl</param>

public SubRuleControl(RuleControl containingRule, SubRule subrule)

{

InitializeComponent();

this.ContainingRule = containingRule;

this.index = 0;

foreach (Control control in ContainingRule.SubRulesPanel.Controls)

{

if (control.GetType() == typeof(SubRuleControl))

this.index++;

}

this.Position();

this.subrule = subrule;

if (subrule.ID > maxID) // update max ID if necessary

maxID = subrule.ID;

}

#endregion

/// <summary>

/// Positions the control in the correct location according to its index

/// </summary>

public void Position()

{

this.Location = new Point(3, this.index \* (Height + 3) + 3 + ContainingRule.SubRulesPanel.AutoScrollPosition.Y);

}

#region Events

private void SubRuleControl\_Load(object sender, EventArgs e)

{

CPUmin.SelectedItem = subrule.CPUmin;

CPUmax.SelectedItem = subrule.CPUmax;

RAMminValue.Text = subrule.RAMminValue;

RAMminUnits.SelectedItem = subrule.RAMminUnits;

RAMmaxValue.Text = subrule.RAMmaxValue;

RAMmaxUnits.SelectedItem = subrule.RAMmaxUnits;

IOminValue.Text = subrule.IOminValue;

IOminUnits.SelectedItem = subrule.IOminUnits;

IOmaxValue.Text = subrule.IOmaxValue;

IOmaxUnits.SelectedItem = subrule.IOmaxUnits;

IOPSmin.Text = subrule.IOPSmin;

IOPSmax.Text = subrule.IOPSmax;

ThreadsMin.Text = subrule.ThreadsMin;

ThreadsMax.Text = subrule.ThreadsMax;

Action1.SelectedItem = subrule.Action1;

Action2.SelectedItem = subrule.Action2;

switch (subrule.Type)

{

case "whitelist":

radioButtonWhitelist.Checked = true;

break;

case "blacklist":

radioButtonBlacklist.Checked = true;

buttonWhitelist.Hide();

buttonBlacklist.Show();

break;

}

}

private void DeleteSubRuleX\_Click(object sender, EventArgs e)

{

ContainingRule.rule.subrules.Remove(subrule);

ContainingRule.SubRulesPanel.Controls.Remove(this);

// Move up the RuleControls from below

foreach (Control control in ContainingRule.SubRulesPanel.Controls)

{

SubRuleControl subruleControl = control as SubRuleControl;

if (subruleControl != null)

{

if (subruleControl.index > this.index)

{

subruleControl.index--;

subruleControl.Position();

}

}

}

// Update maxID

if (subrule.ID == maxID)

{

maxID = 0;

foreach (Control control in ContainingRule.SubRulesPanel.Controls)

{

SubRuleControl subruleControl = control as SubRuleControl;

if (subruleControl != null)

{

if (subruleControl.subrule.ID > maxID)

maxID = subruleControl.subrule.ID;

}

}

}

this.Dispose();

}

private void CPUmin\_SelectedIndexChanged(object sender, EventArgs e)

{

subrule.CPUmin = CPUmin.SelectedItem.ToString();

}

private void CPUmax\_SelectedIndexChanged(object sender, EventArgs e)

{

subrule.CPUmax = CPUmax.SelectedItem.ToString();

}

private void IOminValue\_Leave(object sender, EventArgs e)

{

double NumIOminValue;

if (double.TryParse(IOminValue.Text, out NumIOminValue))

subrule.IOminValue = NumIOminValue.ToString();

else if (IOminValue.Text == "")

subrule.IOminValue = "";

else

{

MessageBox.Show("Field must be a number!", "Invalid Value", MessageBoxButtons.OK, MessageBoxIcon.Warning);

IOminValue.Focus();

IOminValue.SelectAll();

}

}

private void IOminUnits\_SelectedIndexChanged(object sender, EventArgs e)

{

subrule.IOminUnits = IOminUnits.SelectedItem.ToString();

}

private void IOmaxValue\_Leave(object sender, EventArgs e)

{

double NumIOmaxValue;

if (double.TryParse(IOmaxValue.Text, out NumIOmaxValue))

subrule.IOmaxValue = NumIOmaxValue.ToString();

else if (IOmaxValue.Text == "")

subrule.IOmaxValue = "";

else

{

MessageBox.Show("Field must be a number!", "Invalid Value", MessageBoxButtons.OK, MessageBoxIcon.Warning);

IOmaxValue.Focus();

IOmaxValue.SelectAll();

}

}

private void IOmaxUnits\_SelectedIndexChanged(object sender, EventArgs e)

{

subrule.IOmaxUnits = IOmaxUnits.SelectedItem.ToString();

}

private void ThreadsMin\_Leave(object sender, EventArgs e)

{

int NumThreadsMin;

if (int.TryParse(ThreadsMin.Text, out NumThreadsMin))

subrule.ThreadsMin = NumThreadsMin.ToString();

else if (ThreadsMin.Text == "")

subrule.ThreadsMin = "";

else

{

MessageBox.Show("Field must be an integer!", "Invalid Value", MessageBoxButtons.OK, MessageBoxIcon.Warning);

ThreadsMin.Focus();

ThreadsMin.SelectAll();

}

}

private void ThreadsMax\_Leave(object sender, EventArgs e)

{

int NumThreadsMax;

if (int.TryParse(ThreadsMax.Text, out NumThreadsMax))

subrule.ThreadsMax = NumThreadsMax.ToString();

else if (ThreadsMax.Text == "")

subrule.ThreadsMax = "";

else

{

MessageBox.Show("Field must be an integer!", "Invalid Value", MessageBoxButtons.OK, MessageBoxIcon.Warning);

ThreadsMax.Focus();

ThreadsMax.SelectAll();

}

}

private void RAMminValue\_Leave(object sender, EventArgs e)

{

double NumRAMminValue;

if (double.TryParse(RAMminValue.Text, out NumRAMminValue))

subrule.RAMminValue = NumRAMminValue.ToString();

else if (RAMminValue.Text == "")

subrule.RAMminValue = "";

else

{

MessageBox.Show("Field must be a number!", "Invalid Value", MessageBoxButtons.OK, MessageBoxIcon.Warning);

RAMminValue.Focus();

RAMminValue.SelectAll();

}

}

private void RAMminUnits\_SelectedIndexChanged(object sender, EventArgs e)

{

subrule.RAMminUnits = RAMminUnits.SelectedItem.ToString();

}

private void RAMmaxValue\_Leave(object sender, EventArgs e)

{

double NumRAMmaxValue;

if (double.TryParse(RAMmaxValue.Text, out NumRAMmaxValue))

subrule.RAMmaxValue = NumRAMmaxValue.ToString();

else if (RAMmaxValue.Text == "")

subrule.RAMmaxValue = "";

else

{

MessageBox.Show("Field must be a number!", "Invalid Value", MessageBoxButtons.OK, MessageBoxIcon.Warning);

RAMmaxValue.Focus();

RAMmaxValue.SelectAll();

}

}

private void RAMmaxUnits\_SelectedIndexChanged(object sender, EventArgs e)

{

subrule.RAMmaxUnits = RAMmaxUnits.SelectedItem.ToString();

}

private void IOPSmin\_Leave(object sender, EventArgs e)

{

int NumIOPSmin;

if (int.TryParse(IOPSmin.Text, out NumIOPSmin))

subrule.IOPSmin = NumIOPSmin.ToString();

else if (IOPSmin.Text == "")

subrule.IOPSmin = "";

else

{

MessageBox.Show("Field must be an integer!", "Invalid Value", MessageBoxButtons.OK, MessageBoxIcon.Warning);

IOPSmin.Focus();

IOPSmin.SelectAll();

}

}

private void IOPSmax\_Leave(object sender, EventArgs e)

{

int NumIOPSmax;

if (int.TryParse(IOPSmax.Text, out NumIOPSmax))

subrule.IOPSmax = NumIOPSmax.ToString();

else if (IOPSmax.Text == "")

subrule.IOPSmax = "";

else

{

MessageBox.Show("Field must be an integer!", "Invalid Value", MessageBoxButtons.OK, MessageBoxIcon.Warning);

IOPSmax.Focus();

IOPSmax.SelectAll();

}

}

private void Action1\_SelectedIndexChanged(object sender, EventArgs e)

{

subrule.Action1 = Action1.SelectedItem.ToString();

}

private void Action2\_SelectedIndexChanged(object sender, EventArgs e)

{

subrule.Action2 = Action2.SelectedItem.ToString();

}

private void radioButtonWhitelist\_CheckedChanged(object sender, EventArgs e)

{

RadioButton rb = sender as RadioButton;

if (rb.Checked)

{

subrule.Type = "whitelist";

buttonWhitelist.Show();

}

else

buttonWhitelist.Hide();

}

private void radioButtonBlacklist\_CheckedChanged(object sender, EventArgs e)

{

RadioButton rb = sender as RadioButton;

if (rb.Checked)

{

subrule.Type = "blacklist";

buttonBlacklist.Show();

}

else

buttonBlacklist.Hide();

}

private void buttonWhitelist\_Click(object sender, EventArgs e)

{

(new ManageWhitelist(subrule.Whitelist)).ShowDialog();

}

private void buttonBlacklist\_Click(object sender, EventArgs e)

{

(new ManageBlacklist(subrule.Blacklist)).ShowDialog();

}

#endregion

}

}

**SubRule.cs:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Computer\_Resources\_Monitoring

{

public class SubRule

{

public int ID;

public string CPUmin;

public string CPUmax;

public string RAMminValue;

public string RAMminUnits;

public string RAMmaxValue;

public string RAMmaxUnits;

public string IOminValue;

public string IOminUnits;

public string IOmaxValue;

public string IOmaxUnits;

public string IOPSmin;

public string IOPSmax;

public string ThreadsMin;

public string ThreadsMax;

public string Action1;

public string Action2;

public string Type;

public List<string> Whitelist;

public List<string> Blacklist;

public SubRule(int ID, string CPUmin, string CPUmax, string RAMminValue, string RAMminUnits, string RAMmaxValue, string RAMmaxUnits, string IOminValue, string IOminUnits, string IOmaxValue,

string IOmaxUnits, string IOPSmin, string IOPSmax, string ThreadsMin, string ThreadsMax, string Action1, string Action2, string Type, List<string> Whitelist, List<string> Blacklist)

{

this.ID = ID;

this.CPUmin = CPUmin;

this.CPUmax = CPUmax;

this.RAMminValue = RAMminValue;

this.RAMminUnits = RAMminUnits;

this.RAMmaxValue = RAMmaxValue;

this.RAMmaxUnits = RAMmaxUnits;

this.IOminValue = IOminValue;

this.IOminUnits = IOminUnits;

this.IOmaxValue = IOmaxValue;

this.IOmaxUnits = IOmaxUnits;

this.IOPSmin = IOPSmin;

this.IOPSmax = IOPSmax;

this.ThreadsMin = ThreadsMin;

this.ThreadsMax = ThreadsMax;

this.Action1 = Action1;

this.Action2 = Action2;

this.Type = Type;

this.Whitelist = Whitelist;

this.Blacklist = Blacklist;

}

}

}

**ManageWhitelist.cs:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace Computer\_Resources\_Monitoring

{

public partial class ManageWhitelist : Form

{

private List<string> whitelist;

public ManageWhitelist(List<string> whitelist)

{

InitializeComponent();

this.whitelist = whitelist;

}

private void whitelistGridView\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (whitelistGridView.Rows.Count > 0 && e.RowIndex >= 0 && e.ColumnIndex == -1)

{

if (e.RowIndex < whitelistGridView.Rows.Count - 1)

whitelistGridView.Rows.RemoveAt(e.RowIndex);

else

{

string[] row = new string[] { "" };

whitelistGridView.Rows.Add(row);

}

}

}

private void ManageWhitelist\_Load(object sender, EventArgs e)

{

foreach (string process in whitelist)

{

string[] row = new string[] { process };

whitelistGridView.Rows.Add(row);

}

}

private void SaveButton\_Click(object sender, EventArgs e)

{

whitelist.Clear();

foreach (DataGridViewRow row in whitelistGridView.Rows)

{

if (row.Cells[0].Value != null)

{

whitelist.Add(row.Cells[0].Value.ToString());

}

}

this.Close();

}

}

}

**ManageBlacklist.cs:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace Computer\_Resources\_Monitoring

{

public partial class ManageBlacklist : Form

{

private List<string> blacklist;

public ManageBlacklist(List<string> blacklist)

{

InitializeComponent();

this.blacklist = blacklist;

}

private void blacklistGridView\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (blacklistGridView.Rows.Count > 0 && e.RowIndex >= 0 && e.ColumnIndex == -1)

{

if (e.RowIndex < blacklistGridView.Rows.Count - 1)

blacklistGridView.Rows.RemoveAt(e.RowIndex);

else

{

string[] row = new string[] { "" };

blacklistGridView.Rows.Add(row);

}

}

}

private void ManageBlacklist\_Load(object sender, EventArgs e)

{

foreach (string process in blacklist)

{

string[] row = new string[] { process };

blacklistGridView.Rows.Add(row);

}

}

private void SaveButton\_Click(object sender, EventArgs e)

{

blacklist.Clear();

foreach (DataGridViewRow row in blacklistGridView.Rows)

{

if (row.Cells[0].Value != null)

{

blacklist.Add(row.Cells[0].Value.ToString());

}

}

this.Close();

}

}

}