מגשימים

תכנית הסייבר הלאומית

מסמך פרויקט spybot–
תשפ"ד

צוות: נועם בוקל, נועם איפרגן

כיתה: חדרה 8

ראש צוות: ליאל נגר

מנטור: אודי יאבו

פרק 1: יזום

תיאור כללי

הרעיון שלנו הוא אנטי וירוס. בעצם תוכנה שמורכבת מהרבה כלי סריקות שמזהים וירוסים ורוגלות.
התוכנה תזהה תהליכים וקבצים החשודים כזדוניים באמצעות סוגי סריקות שונות: סטטיות ודינמאיות.

מטרת הפרויקט

בחרנו את הנושא של אנטי וירוס בגלל שהוא מאוד מעניין, הוא כולל בתוכו הרבה נושאים כמו רשתות, מערכות הפעלה, low level וגם high level.
הפרויקט שלנו יענה על הצורך של משתמשים שרוצים להגן על המחשב שלהם מוירוסים ורוגלות. זה בעצם מחליף את הצורך של המשתמש בעצמו לבדוק כל תהליך שרץ על המחשב שלו.

ליבה טכנולוגית

הליבה הטכנולוגית של הפרויקט מורכבת מכמה תחומים ביניהם נמצאת היכולת לפקח ולשלוט על על תהליכים ברמת הקרנל בלינוקס, ניתוח תעבורת רשת של פרוססים בזמן ריצה, חקירה של תוכנות בצורה מעמיקה כדי לנטר ולנתח את הפעולות שהן מבצעות בתוך תחום מערכת ההפעלה. כל זה מה שמאפשר החלטה האם קובץ זדוני או לא.

טכנולוגיות עיקריות ושיקולים עיקריים

על מנת לממש את הליבה הטכנולוגית אנחנו צריכים ללמוד לעומק בראש ובראשונה על מערכת ההפעלה של לינוקס. בחרנו לעבוד בלינוקס מפני שבו דברים שאנחנו רוצים לבדוק כמו מידע על פרוססים ושליטה עליהם הרבה יותר אפשרי מאשר בווינדוס.
נצטרך לחקור על בניית Driver בלינוקס בשפת C אשר באמצעותו נוכל לשלוט בפרוססים עם הרשאות root.
נצטרך לחקור גם על YARA Rules על מנת להשוות בין תכני קבצים עם המאגר של yara. פה התלבטנו האם לממש את ה api של yara בעצמנו, אבל הבנו שזה לא החלק המשמעותי של הפרויקט ואין סיבה לעשות את זה. נשתמש ב api של yara בפייתון.
אז בנוסף נצטרך ללמוד על עבודה עם פייתון בלינוקס בשביל שימוש ב scapy על מנת לראות את התעבורה של תהליכים ולנתח אותה.
הסדר שנצטרך הוא:
Yara לינוקס, driver

אתגרים טכנולוגיים ומקורות

האתגרים הטכנולוגיים שצפויים לנו בתחום החקר הם אי מציאת חומר מתאים, אין הרבה חומר על הנושאים שאנחנו צריכים, במיוחד לא כאלו שעשו את הפרויקט שלנו. בשביל לפתור נצטרך לחקור המון ולהתייעץ עם המנטור שלנו, וזוג שעשו משהו דומה לפרויקט שלנו שנה שעברה. נוכל להתקל גם באתגרי מימוש, כמו קוד שלא עובד. את זה נפתור בדיבוג נכון של שנינו, נסביר אחד לשני את הקוד.

מקורות מידע שנוכל להשתמש בהם:

- הסבר על הבסיס לקרנל מודול
- הדוקומנטציה של Yara
- התיעוד המלא של הקרנל של לינוקס
- זיהוי תעבורה של רוגלה
- איך ליצור דרייבר בלינוקס
- הסנפות של פקטות בשפת c
- הסבר על stubs

סוגי משתמשים / קהל היעד

סוגי המשתמשים אליהם מיועד הפרויקט שלנו הם אנשים שרוצים להגן על המחשב שלהם מפני וירוסים ורוגלות ולבדוק קבצים במחשב שלהם. לא נדרש ידע מקדים לשימוש בתוכנה חוץ מההבנה בסיסית של לינוקס

דרישות חומרה

אין דרישות חומרה מיוחדות

פתרונות קיימים

כן, ישנם מוצרים ופתרונות דומים המתייחסים לחלק מההיבטים שאנו שואפים להתמודד עם הפרויקט שלנו. להלן מספר חלופות קיימות:
תוכנות אנטי-וירוס קלאסיות, כגון Norton, McAfee ו-Windows Defender, מתמקדות בזיהוי והסרה של תוכנות זדוניות ווירוסים ידועים. הם מסתמכים על זיהוי מבוסס חתימה וסריקות היוריסטיות כדי להגן מפני איומים.

פרק 2: אפיון

פיצ'רים ותהליכים עיקריים

השוואת hashes של קבצים עם hashes שמורים:
קלט: קובץ לבדיקת תוכן זדוני.
פלט: אינדיקציה אם ה-hash של הקובץ תואם לאחד במסד הנתונים של hash.

המטרה העיקרית היא לזהות במהירות קבצים שעלולים להיות זדוניים על ידי השוואת ה-hashs שלהם למסד נתונים של קבצים זדוניים ידועים. זו הבדיקה הראשונית של התוכנה.

אנו נשתמש ב-C++ בשילוב עם ספריית SQLite3 ו openssl, התוכנה תחשב את ה-hash SHA-256 של קובץ הקלט ולאחר מכן תשווה את ה-hash הזה מול ה-hash המאוחסנים במסד הנתונים של SQLite(, blacklist). אם נמצאה התאמה בתוך ה blacklist, הקובץ מסומן כזדוני ויעורר שאילה למשתמש האם למחוק את הקובץ או לא. אם לא נמצאה ההתאמה התוכנה ממשיכה כרגיל.

אנחנו עושים זאת באמצעות אלגוריתם ה- hash SHA-256(חלק ממשפחת האלגוריתמים SHA-2, שנועדה להחליף את הסדרה הקודמת, SHA-1. האלגוריתם מייצר hash באורך של 256 סיביות, המספק רמת אבטחה גבוהה יותר מפני התקפות התנגשות. זה אומר שכמעט בלתי אפשרי ליצור שני קבצים שונים שמייצרים את אותו hash.)
הוא מומלץ בשל עמידותו החזקה מפני מתקפות התנגשות והשימוש הנרחב בו באבטחת סייבר.
על ידי שימוש ב-SHA-256, אנו מבטיחים שלכל קובץ יש hash ייחודי וחזק, מה שמקשה על יצירת קובץ אחר עם אותו hash.

במקרה שה-hash של הקובץ אינו קיים במסד הנתונים, התוכנה תמשיך לסריקות היוריסטיות מורכבות יותר, כגון: סריקת YARA

סריקת קבצים לזיהוי תוכנה זדונית על ידי YARA:
קלט: נתיב לקובץ או לתיקייה לסריקה.
פלט: אינדיקציה אם הקובץ/ים נחשבים לחשודים כתוכנה זדונית.

המטרה היא לבצע סריקה מהירה ויעילה של קבצים באמצעות חיפוש מחרוזות ספציפיות בתוכנם.

הכלי שבו אנו משתמשים למטרה זו הוא YARA,כלי שמטרתו לעזור לחוקרי תוכנות זדוניות לזהות ולסווג דוגמאות של תוכנות זדוניות. עם YARA אפשר ליצור תיאורים של משפחות תוכנות זדוניות (או כל מה שרוצים לתאר) בהתבסס על תבניות טקסטואליות או בינאריות. כל תיאור, חוק, מורכב מקבוצה של מחרוזות וביטוי בוליאני שקובעים את ההיגיון שלו. דוגמה לחוק כזה:
השימוש ב-API של YARA ב-Python מאפשר לנו לבצע את הסריקה באופן אוטומטי ויעיל.
דוגמה לפונקציה שנצטרך להשתמש בה: הקוד יטען את כללי ה-YARA המקומפלים מהקובץ שצוין, אשר לאחר מכן ניתן להשתמש בו כדי להתאים לנתוני יעד או קבצי יעד.

```
rules = yara.load('/foo/bar/my_compiled_rules')
```

במהלך הפעלת התוכנה, יש להבטיח שיש לנו סט של חוקי YARA שבהם מוגדרים הקריטריונים לזיהוי תוכנות הזדוניות. התוכנה אוטומטית מעדכנת את החוקים מהאינטרנט בכל הרצה.

הבדיקה עצמה משווה את התוכן של הקובץ לסט של חוקים מראש. אם קיימת התאמה לאחד החוקים, התוכנה מסמנת את הקובץ כחשוד.

דוגמה לחוק ב-YARA:
החוק הבא מגדיר שכל קובץ המכיל אחת המחרוזות או הבייטים המוגדרים בו, יתוויג ב-"silent_banker":

```
rule silent_banker : banker {
   meta:
      description = "This is just an example"
      threat_level = 3
      in_the_wild = true

   strings:
      $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
      $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
      $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

   condition:
      $a or $b or $c
}
```

השוואת כתובות שתהליך כלשהו מדבר איתו לכתובות שידועות כחשודות:
קלט: כתובת IP.
פלט: האם IP זה ידוע כזדוני.

המטרה היא לזהות תהליכים החשודים כוירוס\רוגלה על סמך דפוסי התקשורת שלהם ברשת.
תהליכים שמתקשרים על כתובות הידועות כזדוניות מעידים על וירוסים\רוגלות.

נעשה זאת באמצעות שימוש בפיצ'ר נוסף: סריקה של תעבורת הרשת של תהליכים.
משום שפיצ'ר זה משתמש ב scapy ב python על מנת להוציא מידע על התקשורת שמבצע תהליך.
מהמידע שהוא חילץ ניקח את DSET IP.
לאחר שחולצו כתובות ה-IP, התוכנה(הפיצ'ר הזה) יצליב אותן עם מסד הנתונים שנוציא בזמן אמת מהאתרBlocklist.de המכיל כתובות חשודות שנוספות כל כמה שעות. התאמות ישר אומרות שהתהליך הזה זדוני. אם אף כתובת לא נמצאה כחשודה הוא ממשיך לבדיקות הבאות.

את תהליך הוצאת מסד הנתונים נעשה גם בשפת python בשימוש בספריית request ליצירת בקשת HTTP GET לכתובת: https://lists.blocklist.de/lists/all.txt

גם ניצור טבלת sql בשביל שמירת התאמות שמצאנו ואליו גם יעודכנו כתובות הIP הזדוניות מהאתר כל יום ב23:59. נעשה זאת על מנת השוואה נוחה יותר ומסודרת.

סריקה של תעבורת הרשת של תהליכים:
קלט: פקטות נכנסות ויוצאות מהמערכת.
פלט: פרטים על התעבורה עבור כל פרוסס שמשתמש בכרטיס רשת.

הכלי ישתמש בספריה Scapy(ספרייה בPython המאפשרת למשתמש לשלוח, להסניף ולנתח ולזייף פקטות רשת) של פייתון כדי להסניף(קבלה של פקטות) את הפקטות(קבוצות בסיסיות של נתונים המועברים ברשת מחשבים.) ולנתח אותן,
לאחר שהכלי הסניף פקטה הוא עובר עליה ומחלץ ממנה את הפרטים הבאים:

- Destination IP - הכתובת שאיתה מתקשר התהליך שנחקר. את הכתובות הזו הוא משווה אל מול מאגר נתונים לפי פיצ'ר: השוואת כתובות שתהליך כלשהו מדבר איתו לכתובות שידועות כחשודות
- Weight - כמות המידע שמועבר.
- DATA - המידע שהועבר.

בנוסף לכך הכלי גם יבדוק את תדירות שליחת הפקטות בזמן שהפרוסס רץ עבור כל פרוסס שעדיין לא אושר.
את בדיקת תדירות שליחת הפקטות הוא יעשה באופן הבא:
הכלי עוקב באופן רציף אחר הקצב שבו הם משדרים מנות. זה נעשה על ידי חותמת זמן של כל חבילה שהתקבלה מהתהליך.
הכלי מגדיר סף תדירות על סמך התנהגות אופיינית או צפויה. סף זה מגדיר את הגבול העליון של מה שנחשב לתדירות שליחת פקטות רגיל עבור אותו תהליך.
אם הכלי מזהה שתהליך חורג מסף התדירות שהוגדר, הוא יגדיר אותו כחשוד, יעצור אותו עד שהמתמש יענה. המשתמש יענה על אם הוא מכיר את התהליך הזה, במקרה של לא, התהליך יוכר כזדוני ויסגר, במקרה של כן התהליך יוכל להמשיך רגיל.

הכלי גם יבדוק האם יש תהליך במערכת שמנסה לסרוק את הרשת, בפועל תהליך שמנסה לזהות מחשבים נוספים באותה רשת. כדי לזהות תהליך כזה, אנו סורקים מדי פעם את הפורטים והפרוטוקולים הפתוחים במחשב, שבהם יכולים לתקשר תהליכים. המטרה היא לזהות תהליך שמנסה להתחבר פעמים רבות בזמן קצר ומשך ההתחברות קצר. את זה נעשה באמצעות קוד python המסניף את הרשת באמצעות scapy

ונוציא באיזה פרוטוקל כל פקטה משתמשת, ואם מגיעים לזה שיש מספר גבוה של פקטות המתחברות לאותו פורט אז זה אומר שיש משהו חשוד.

בנוסף, כדי לזהות איזה תהליך שלח איזו פקטה נשתמש בספריית psutil(ספרייה המשיגה אינפורמציה על פרוססים רצים) ונביא לה את ה IP ששלח\קיבל את הפקטה.

אפשרות שינוי מצב של תהליך(סגירה, השהייה, המשך):
קלט: PID של התהליך שרוצים לשנות את מצבו.
פלט: מצב התהליך שצוין משתנה על פי ההחלטה שהתקבלה לאחר הניתוח.

המטרה היא לספק כלי רב-תכליתי המציע שליטה על תהליכים שעלולים להיות זדוניים. הכלי מאפשר:

- להשהות תהליכים לבדיקה מפורטת.
- להמשיך תהליכים עפ"י החלטת המשתמש.
- להפסיק תהליכים שאושרו כזדוניים או לא רצויים.

המימוש יתבצע באמצעות דרייבר בשפת C באמצעות ה-API קרנל של לינוקס. הדרייבר הוא רכיב תוכנה ברמה נמוכה המקיים אינטראקציה ישירה עם ליבת מערכת ההפעלה. הדרייבר מציע יתרונות בולטים:

- גישה מוסמכת: הדרייבר יכול לפעול עם הרשאות גבוהות, המאפשרות לו אינטראקציה ישירה עם הליבה (כדי לקבל את הגישה לעבוד בתוך ה - Kernel התוכנה תצטרך לקבל הרשאות Root ולהתקין את הדרייבר ל - Kernel,). את ההרשאות הגבוהות התוכנה תקבל ברגע הורדת ע"י המשתמש. שם המשתמש יתבקש להכניס את הסיסמה שלו בשביל sudo.
- יעילות: הדרייבר מבצע את הפעולות בצורה יעילה ומהירה.
- שליטה בזמן אמת: הדרייבר מספק שליטה מדויקת ובזמן אמת על תהליכים.

בשביל לעשות את הפעולות האלו הדרייבר משתמש בפונקציה sendsig()
כדי לשלוח את ה - Signal (השם המקביל ל - interrupts ב Windows) לקרנל של לינוקס שיבצע את הפעולות שהדרייבר צריך.

ה - Signals שבהם הדרייבר ישתמש הם:

SIGSTOP: להשהיית התהליך.
SIGCONT: להמשך תהליך שהושהה בעבר.
SIGKILL: להפסקת התהליך באופן סופי.

שמירת האשים של קבצים זדוניים:
קלט: קובץ לשמירה.

פלט: האש של הקובץ מתווסף לבסיס הנתונים SQLite לאחסון האשים החשודים(blacklist).

המטרה היא להבטיח שלכל קובץ זדוני יתווסף ה-hash שלו למסד הנתונים שלנו. במקרה שהקובץ יסגר, עדיין יהיה לנו את ה-hash שלו לזיהוי עתידי או השוואה מול hash אחרים.

אנו נשתמש ב-C++ לצד ספריית SQLiteכ כדי להזין את ה-hash של הקובץ החשוד למסד הנתונים.

את מסד הנתונים נעדכן כל פעם כשנמצא קובץ זדוני. ככה זה יחסוך בדיקות מעמיקות וישר ידע שהקובץ הזה חשוד.

נעשה זאת באמצעות אלגוריתם ה-hash SHA-256 הממומלץ בשל עמידותו החזקה מפני מתקפות התנגשות והשימוש הנרחב בו באבטחת סייבר.
על ידי שימוש ב-SHA-256, אנו מבטיחים שלכל קובץ יש hash ייחודי וחזק, מה שמקשה על יצירת קובץ אחר עם אותו hash.

אם ה-hash כבר קיים במסד הנתונים, הקובץ לא יתווסף פעמיים, מה שמבטיח את יעילות הכלי.

[הטבלה שבה נשמרים הקבצים]

ניתוח system calls של תהליכים שרצים:
קלט: כתובת של טבלת ה system calls
פלט: PID של התהליך הנמצא

הכלי מקבל כקלט את כתובת טבלת ה- system calls, בה רשומות כתובות לכל קריאות המערכת. באמצעות הטבלה, אנו יכולים להשפיע על הפעולות של קריאות open ו-exec, ולהחליפן בקריאות לפונקציות שלנו. זה מאפשר לנו לזהות אילו תהליכים מבצעים את הפעולות הללו. בפונקציות שלנו, אנו קוראים את ה-PID של התהליכים ומכניסים אותם לזיכרון שמשותף לכלי האנטי וירוס שלנו. כך, אנו יכולים לזהות אילו תהליכים מתנהגים כמו וירוס ולחסום את הפעולות שלהם על ידי שינוי ההרשאות. מובן שהבחינה הזו מספקת מידע נוסף על התהליכים ואינה קובעת האם הם באמת וירוסים,אלא מספקת מידע נוסף על הוירוסים.

זיהוי של Packers - קטעי קוד שתפקידם להסתיר את פעילות קטעי הקוד בתוכנה:
קלט: PID של פרוסס.
פלט: האם הפרוסס עובד עם Packer והאם הוא זדוני או שהוא תקין.

הכלי קודם שולח את ה Hash של הקובץ הרצה ששייך לאותו פרוסס כדי לוודא אם הוא לא כבר מזוהה בצורה זדונית, אם הקובץ לא מזוהה כזדוני הכלי יבדוק אם קובץ ההרצה של הפרוסס מכיל packer.

אם הפרוסס אינו עובד עם Packer הכלי יחזיר אותו לניתוח של שאר הכלים ולא ידווח עליו,
לעומת זאת אם יזוהה Packer כלשהו בתוך קטע הקוד של התכנית הכלי יתריע על כך שפרוסס זה משתמש ב Packer כדי להסתיר את עצמו ולאחר מכן הוא יתחיל לנתח את קטעי הקוד אחרי שהם עוברים unpacking ונטענים לזיכרון, במהלך הניתוח אם הכלי יזהה חלק זדוני בקטע הקוד הוא יתריע על כך ויסמן את הפרוסס כזדוני.

#הערה - פיצ'ר זה מורכב משאר הפיצ'רים ומכיוון שכך הוא גם צריך ספרינט מחקר וביצוע פרטי רק על הפיצ'ר הזה לבד, ולכן כדי שיהיה לנו ברור כל המידע על הניתוח והטכנולוגיות בהן נצטרך להשתמש נצטרך להקדיש זמן ללמידת הפיצ'ר והטכנולוגיות שבו לעומק, לדוגמה: הדרך שבה נזהה את ה Packer מתוך קובץ ההרצה, הדרך שבה נקרא מתוך הזיכרון אליו נטען הפיצ'ר (RAM) וננתח את הקוד לצורך זיהוי אם הוא זדוני.

הרצה ברקע של התוכנה:
קלט - קבצי הרצה של התוכנה
פלט - אין

המטרה היא להריץ את השרת המקומי של התוכנה ברקע, כך הוא ירוץ כל עוד המחשב דולק ולקוח יכול להתחבר כל הזמן.
עוד מטרה להרצה ברקע היא סריקות בזמן אמת. השרת המקומי ברגע שנדלק המחשב יבצע, בלי בקשה מהלקוח, סריקה ברשת/במערכת הקבצים ויכתוב על דברים מעניינים שמצא לתוך קובץ מסויים. מקובץ זה יקרא הלקוח ויקפיץ למשתמש הודעה על וירוסים שנמצאו או לא נמצאו.

את ההרצה ברקע אנחנו מתכוונים לממש באמצעות daemon, שהוא תהליך רקע שמתחיל באתחול המערכת וממשיך לפעול עד כיבוי המערכת.
ה daemon הוא קובץ מסוג service שחייב להמצא במיקום: /etc/systemd/system/
בקובץ זה רשומות הגדרות לגבי ה daemon כגון איזה קובץ להריץ, כמה פעמים לנסות שוב אם לא עובד, מי רשאי להריץ אותו וכו'. דוגמה לקובץ זה:

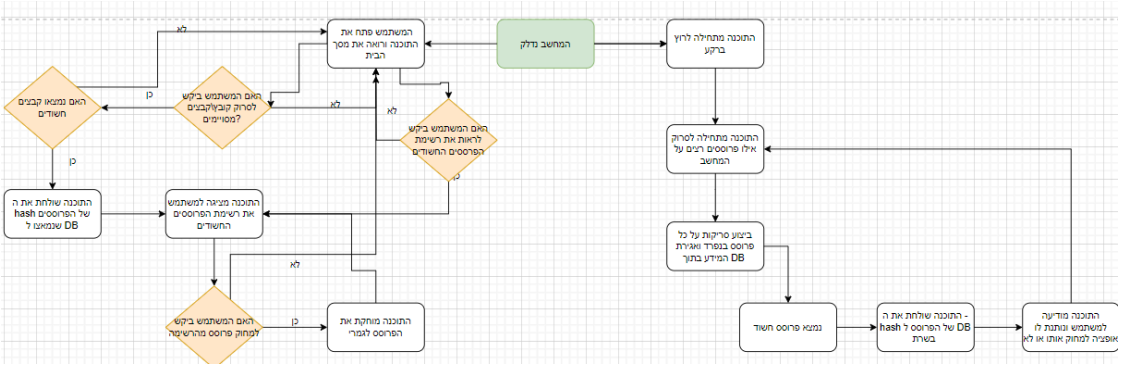
```
[Unit]
Description=My first daemon!
[Service]
User=<yourUser>
#Code to execute
#Can be the path to an executable or code itself
WorkingDirectory=/home/ubuntu/mydaemon
ExecStart=/home/ubuntu/mydaemon/executable
Type=simple
TimeoutStopSec=10
Restart=on-failure
RestartSec=5
[Install]
WantedBy=multi-user.target
```

טכנולוגיות (Backend, Server, Driver)

פיצ'ר/תהליך	טכנולוגיות ושפות תכנות	משאבים נדרשים ושירותים חיצוניים
השוואת hashes של קבצים עם hashes שמורים	נכתב ב cpp בשימוש בספריית sqlite3. את יצירת ה hash נעשה באמצעות אלגוריתם SHA-256	חיבור לאינטרנט כדי לגשת לטבלאות בשרת

שמירת האשים של קבצים בין אם זדוניים או לא	נכתב ב cpp בשימוש בספריית sqlite3. את יצירת ה hash נעשה באמצעות אלגוריתם SHA-256	חיבור לאינטרנט כדי לגשת לטבלאות בשרת
סריקת קבצים לזיהוי תוכנה זדונית על ידי YARA	נכתב בpython בשימוש ב YARA API	
סריקה של תעבורת הרשת של תהליכים	סריקת תעבורת הרשת של תהליכים באמצעות Python והספרייה Scapy לקריאה וניתוח חבילות הרשת.	כרטיס רשת, וחיבור לאינטרנט.
השוואת כתובות שתהליך כלשהו מדבר איתו לכתובות שידועות כחשודות	שימוש בטכנולוגיות של פיצ'ר קודם, ובנוסף שימוש ב requests לשליחת בקשת מסד נתונים מהאתר blocklist.	כרטיס רשת, וחיבור לאינטרנט.
אפשרות שינוי מצב של תהליך(סגירה, השהייה, המשך)	פיתוח מנהל התקן (driver) בשפת C ללינוקס באמצעות ממשק התכנות הגרעיני (Kernel API) של לינוקס ושליחת בקשות SIG.	גישת Root כדי להפעיל את ה Driver (תינתן ע"י מתקין התוכנה)
ניתוח system calls של תהליכים שרצים	המשך ל driver, מבצע הוק לטבלת קריאות המערכת ומשנה כתובות לפעולות מסוימות.	
זיהוי של Packers - קטעי קוד שנוספים לתוכנות ותפקידם להצפין את קוד התוכנה הזדונית.	ממומש ב c \ c++ זיהוי ה packers באמצעות ניתוח קוד בצורה סטטית, וניתוח קוד באמצעות מיפוי זיכרון (memory mapping) על ה RAM.	
הרצה ברקע של התוכנה		גישת root כדי לערוך קבצים הנמצאים במיקום הנדרש.

תרשים זרימה



מבנה בסיס נתונים

בפרויקט זה החלטנו לממש את מסד הנתונים באמצעות Access DB של מיקרוסופט מסוג SQL.

א. טבלה של כתובות זדוניות:

- ID (INTEGER, PRIMARY KEY, AUTOINCREMENT, NOT NULL): A unique identifier for each entry.
- IPAddress: The malicious IP address.
- DateAdded: When the IP was added to the database. It's set to automatically take the current timestamp when a new entry is made.
- Source: Where the IP was fetched from (defaulted to 'Blocklist.de' but can be changed if you fetch from multiple sources).
-

ב. טבלת hash של תהליכים שהתוכנה זיהתה כזדוניים:

- ID (INTEGER, PRIMARY KEY, AUTOINCREMENT, NOT NULL): A unique identifier for each entry.
- FileName (TEXT, NOT NULL): The name of the suspicious file.
- FilePath (TEXT): The path to the suspicious file in the system.
- FileType(TEXT): The type of the file(elf, data, stub, txt)
- FileHash_SHA256 (TEXT, NOT NULL): The SHA-256 hash of the file.
- DateAdded (DATETIME): The date and time when the hash was added to the database.
- Encrypted (BOOLEAN, NOT NULL): A flag indicating whether the hash in the database is encrypted.

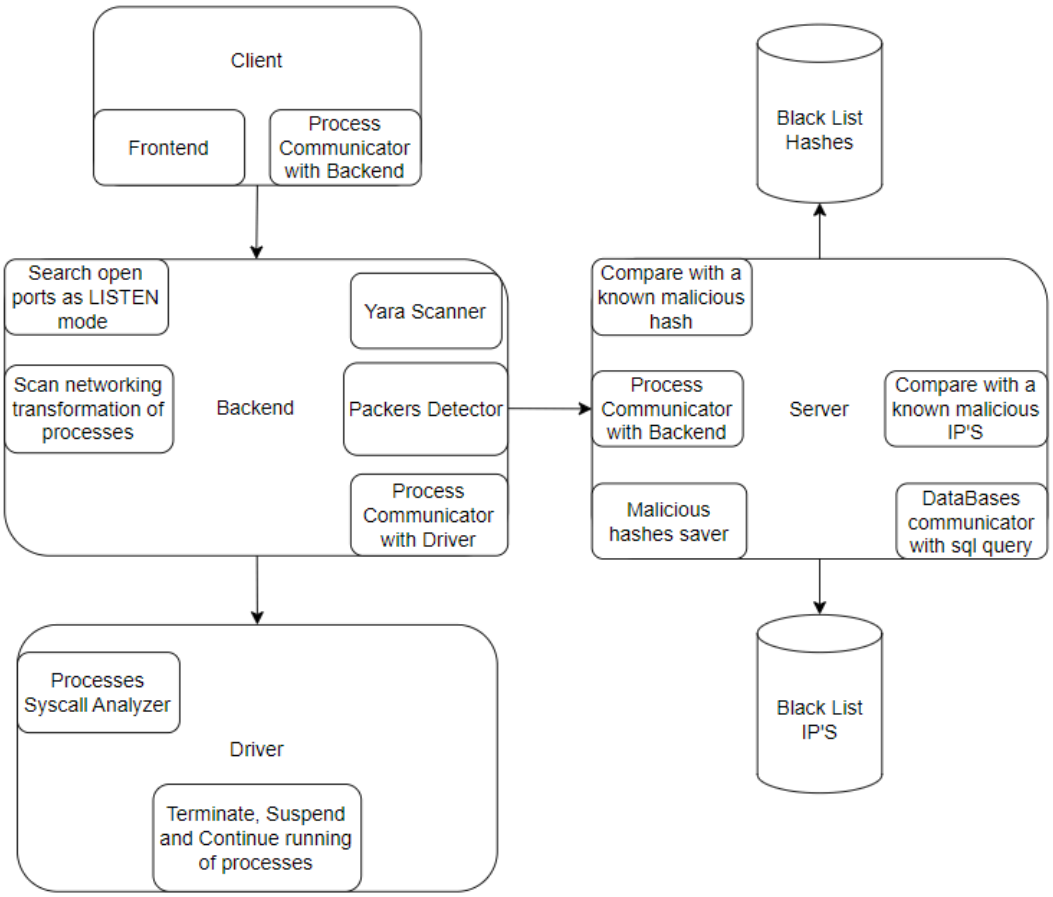
- SourceDetection (TEXT): The tool/method that identified the file as suspicious (e.g., "YARA", "Network", etc.). (this is mostly for us to check each tool for false positives)
- IsolationStatus (TEXT): Indicates whether the file is isolated, quarantined, or active.

ג. טבלת תהליכים וקריאות המערכת שמבצעים:

- ID (INTEGER, PRIMARY KEY, AUTOINCREMENT, NOT NULL): A unique identifier for each entry.
- Path(TEXT, NOT NULL): The location in the memory of the running process.
- PID(INTEGER, NOT NULL): The pid of the runner process.

פרק 3: ארכיטקטורה

מבט על



הסבר:

- Client-
  - frontend: רכיב האחראי על הצגת המידע למשתמש בצורה נוחה וקלה לשימוש.
    process communicator with backend: אחראי על ביצוע התקשורת מול הbackend.
    רכיב זה יפותח בשפת c#.

- Backend-
  - רכיב זה עושה את כל סוגי הסריקות הדינאמיות(היוריסטיות) שיש לנו במקרה והפרוסס לא נמצא ב white list.
    הוא עושה באמצעות פתיחת פרוסס של כלי הסריקות (כל סריקה בכלי\solution משל עצמה)
  - חלק זה יפותח באמצעות שפת C בעזרת פונקיות exec ו fork המשמשים להרצת הכלים.
  - Search open ports as LISTEN mode – חלק מהסריקה ההיוריסטית. כלי שיפותח בשפת python ומטרתו לבדוק האם יש פורטים פתוחים כצד שרת על המחשב וקשורים בתוכנה הספציפי.
  - Scan networking transformation of processes- חלק מהסריקה ההיורוסטית. כלי שיפותח בשפת python בשימוש בספריית Scapy. מטרתו היא לבדוק מה פרוססים עושה ברשת.
  - peckers detector – חלק מהסריקה ההיורסטית. רכיב אשר אחראי על סריקה של קובץ ובודק האם הוא הצפין בתוכו קטע קוד (הוא הורד כשקטע הקוד הזה מוצפן על מנת שלא יתגלה) והאם הוא מפענח אותו ויכול להריץ אותו. יממוש בשפת python.
  - yara scanner – חלק מהבדיקות הסטטיות שמתמש מבקש לעשות על פרוסס שלא בהכרח רץ. מטרתו לבדוק האם הקובץ שקיבל הוא זדוני ע"י בדיקה של תוכנו עם מאגר החוקים של yara. רכיב זה יפותח בpython בשימוש ב yara API.
- Driver-
  - רכיב אשר מטפל בפרוססים. יכתב בשפת C באמצעות linux kernel API.
  - controling processes- הרכיב העקרי בדרייבר. מטרתו היא לשלוט בפרוסס. הרכיב יכול לעצור פרוסס, להמשיך פרוסס שנעצר, וגם להרוג פרוסס.
  - Process syscalls analyzer – חלק מהסריקה ההיוריסטית. כלי זה מטרתו לנתח קריאות שפרוסס עושה לקרנל על דברים שאין לו הרשאה לעשותם. יפותח בשפת python.
  - process communicator with driver – רכיב אשר אחראי על תקשורת והעברת בקשות לדרייבר, יממוש בשפת C (יכתוב לתוך קובץ משותף ל backend ולdriver מה שהdriver צריך לעשות)
- Server-
  - רכיב אשר מטפל בבקשות ממסדי הנתונים ומהסריקות הסטטיות.
  - compare with know malicious hash- רכיב האחראי על הבדיקה הראשונית כל כל פרוסס שרץ. הוא משווה את חתימת הפרוסס עם מאגר נתונים של האשיפש spybot אסף בכל סריקותיו. רכיב זה יפותח בשפת cpp ושימוש בספריית sqlite3.
  - malicious hashes saver – רכיב אשר אחראי על שמירות hash של קבצים/פרוססים אשר נמצאו כזדוניים ע"י spybot. יפותח בשפת cpp בשימוש בספריית sqlite3.
  - process communicator with backend- רכיב אשר אחראי על תקשורות עם ה backend: מענה לבקשות שלו. יפותח בשפת cpp.
  - database communicator – רכיב האחראי על בקשות ממסדי הנתונים של spybot. יפותח בשפת cpp.

עיצוב הנתונים ויישויות מידע

המידע המועבר בתוך המערכת שלנו הוא בקשות.
הבקשות נחלקות לשני סוגים: בקשות בין הbackend ל server\driver ובין client ל backend.
בקשות ומענות בין ה clientל backend:

- בקשת הפעלת סריקה (דינאמית. אם ירצה לגבי קובץ\ים ספציפיים זה יהיה סטטי)
- מענה של נתוני הסריקה.
- בקשה לקבלת רשימת פרוססים שנעצרו מחשד כזדוניים.
- מענה עם רשימת פרוססים חשודים מושהים.
- בקשה לעידכון רשימות
- בקשה להמשך ריצה של פרוסס שנחסם.
- בקשה לחסימת פרוסס.
- בקשה להריגת פרוסס
- בקשה לעדכון\קבלת נתיב לתיקיית סריקה פריודית
- בקשה לעדכון\קבלת זמן סריקה פיריודית

בקשות ומענות בין הclient ל server:

- בקשה לסריקה ראשונית של כתובת IP שהפרוסס מתקשר איתה.
- מענה אם הכתובת חשודה או מאושרת.
- בקשה לסריקה ראשונית של hash של פרוסס.
- מענה אם הפרוסס זדוני או לא לפי ה – hash.
- בקשה לשמירת כתובת IP ב DataBase של ה Black List IP’S.
- מענה אם השמירה של ה IP הצליחה.
- בקשה לשמירת hash ב DataBase של Black List Hashs.
- מענה אם השמירה של ה hash הצליחה.

בקשות ומענות בין הbackend ל driver:

- בקשה להשהיית ריצה של פרוסס.
- בקשה להמשך ריצה של פרוסס.
- בקשה לסיום ריצה של פרוסס.

טכנולוגיות עיקריות

- רכיב השרת יבנה בשפת C++ מכיוון שאנחנו מכירים כבר את הספריה Sqlite שבה נשתמש בעבודה עם ה DataBaeים, וגם מכיוון שאנחנו מנוסים בבניית Sockets בשביל לתקשר עם הלקוח ב C++, ובנוסף גם יתקשר עם Yara באמצעות Python.
- רכיב הBackend יבנה בשפות: C, C++, Python.
  את הסריקות שקשורות בתעבורת רשת ובניתוח פקטות נבנה ב Python.
  את הסריקה שקשורה בניתוח Syscalls נבנה ב python.
  את הרכיב שמקשר בין כל הסריקות ומתקשר עם השרת נבנה ב C++ מאותה סיבה שאנחנו בונים את השרת בשפה זו, מכיוון שאנחנו רגילים לבנות Sockets ב C++.
- רכיב הדרייבר יבנה בשפת C ויתקשר עם ה Kernel של מערכת ההפעלה כדי לבצע את הפעולות שה Backend לא יכול לבצע כגון:
  - השהיית ריצה של פרוסס.
  - המשך ריצה של פרוסס.
  - סיום ריצה של פרוסס.
- רכיבי ה DataBase של ה Black List IP'S & Black List Hashes יווצרו דרך השרת ויקושרו לשרת.

התאמה לאפיון

פיצ'ר	רכיבים רלוונטים
סריקת Hashes באמצעות Yara	Backend (סריקת ה Hash של הקובץ > תקשורת עם השרת) > שרת (תקשורת עם ה Backend והלקוח > Yara Server)
סריקת Hashes לוקאלית על DataBase של Black List Hases שנמצא על השרת	Backend (סריקת ה Hash של הקובץ > תקשורת עם השרת) > שרת (תקשורת עם ה Backend והלקוח > Local Black List Hashes DataBase)
השוואת כתובות שתהליך כלשהו מדבר איתן לכתובות שידועות כחשודות באמצעות DataBase של Black List IP'S	Backend (סריקת כתובות IP שתהליך מתקשר איתן > תקשורת עם השרת) > שרת (תקשורת עם ה Backend והלקוח > Local Black List IP'S DataBase)
סריקת תעבורת רשת של תהליכים	Backend > scan networking transformation
שינוי מצב של תהליך (סגירה, השהייה, המשך)	Backend > Driver
שמירת האשים של קבצים שאנחנו גילנו כחשודים	Backend > Server > DataBases communicator with sql query
זיהוי system calls של תהליך בזמן ריצה	Backend > Process syscalls analyzer(using ptrace)

פרק 4: תוכנית עבודה

מתן עדיפות לפיצ'רים, חלוקה לשלבים מסודרים

1. השוואת hashes של קבצים עם hashes שמורים
2. שמירת האשים של קבצים בין אם זדוניים או לא
3. סריקת קבצים לזיהוי תוכנה זדונית על ידי YARA
4. סריקה של תעבורת הרשת של תהליכים
5. השוואת כתובות שתהליך כלשהו מדבר איתו לכתובות שידועות כחשודות
6. פיתוח daemom
7. אפשרות שינוי מצב של תהליך(סגירה, השהייה, המשך)
8. ניתוח system calls של תהליכים שרצים
9. זיהוי של Packers – קטעי קוד שנוספים לתוכנות ותפקידם להצפין את קוד התוכנה הזדונית.

מ'ס פיצ'ר	משימה	תלויות תשתית (האם המשימה תלויה במשימה אחרת)	נימוק והערות	רכיב רלוונטי
1	השוואת hashes של קבצים עם hashes שידועים כזדוניים	המשימה תלויה בבניה של מסדי הנתונים	נשווה באמצעות אלגורתים sha256	server
2	שמירת hash של קבצים שהתגלו בין אם זדוניים או לא	המשימה תלויה בבניה של מסדי הנתונים	ניצור את ה hash באמצעות אלגוריתם SHA256	server
3	סריקת קבצים לזיהוי תוכנה זדונית על ידי YARA	המשימה אינה תלויה במשימה אחרת	נעשה זאת ע"י שימוש ב API של YARA	server

4	סריקה של תעבורת הרשת של תהליכים	המשימה אינה תלויה במשימה אחרת	שימוש ב SCAPY	backend
5	השוואת כתובות שתהליך כלשהו מדבר איתו לכתובות שידועות כחשודות	המשימה תלויה בסריקת תעבורת הרשת של תהליכים	ניקח באופן דינאמי את מסד נתוני IP זדוניים מהאתר blocklist.de	server
6	פיתוח daemon	אינה תלויה במשימה אחרת		backend
7	אפשרות שינוי מצב של תהליך(סגירה, השהייה, המשך)	המשימה אינה תלויה במשימה אחרת	אנחנו נשתמש בשלוש סוגי בקשות: SIGSTOP, SIGCONT, SIGKILL	Driver
89	ניתוח system calls של תהליכים שרצים	המשימה אינה תלויה במשימה אחרת		backend
9	זיהוי של Packers – קטעי קוד שנוספים לתוכנות ותפקידם להצפין את קוד התוכנה הזדונית.	המשימה אינה תלויה במשימה אחרת		backend

חלוקה לאיטרציות (ספרינטים)

<u>איטרציה 1 (אינה כוללת אף פיצ'ר):</u>

- פיתוח מערכת התקשורת שדרכה יפעלו הרכיבים (client, backend, server)
- החלטה על פרוטוקל שבו ידברו וישלחו בקשות רכיבים.

<u>איטרציה 2:</u>

- יצירת מנגנון ה hash
- יצירת מסדי הנתונים השונים (פיצ'ר 2)
- שמירת hash במסדי הנתונים (פיצ'ר 2)
- השוואת hash עם מסדי הנתונים (פיצ'ר 1)
- שמירת IP במסדי הנתונים (פיצ'ר 5)
- השוואת כתובות IP עם מסדי נתונים מהאתר blocklist (פיצ'ר 5)
- סריקת קבצים ע"י yara (פיצ'ר 3)

<u>איטרציה 3:</u>

- יצירת driver (פיצ'ר 8)
- בניית daemon (פיצ'ר 7)

<u>איטרציה 4:</u>

- מימוש ניתוח system calls (פיצ'ר 8)
- סריקה של תעבורת הרשת של תהליכים (פיצ'ר 4)
- בדיקה האם יש פורטים פתוחים כצד שרת על המחשב וקשורים בתוכנה הספציפית (פיצ'ר 6)

<u>איטרציה 5:</u>

- בדיקה האם יש פורטים פתוחים כצד שרת על המחשב וקשורים בתוכנה הספציפית (פיצ'ר 6) (המשך)
- זיהוי של Packers – קטעי קוד שנוספים לתוכנות ותפקידם להצפין את קוד התוכנה הזדונית (פיצ'ר 10) – פיצ'ר זה בסימן שאלה לפי הזמן שישאר לנו כי הוא הכי מסובך.
- ממשק גרפי

פרק 5: עיצוב – מדי איטרציה

איטרציה 1 – מתמקדת בהכנת תשתית להמשך הפרויקט

<u>החזון שלנו לסיום הספרינט:</u>
עד סוף הספרינט הזה, אנו שואפים ליצור תקשורת בין הלקוח ל-backend, ולאפשר לשלוח ולקבל בקשות ספציפיות.

<u>חלוקת עבודה ולוח זמנים:</u>
במהלך הספרינט הזה, ההתמקדות שלנו תהיה בהחלטה על פרוטוקול התקשורת ובניית התשתית, השרת, backend והלקוח. אנו נשתף פעולה כדי שנהיה מתואמים בנוגע לבקשות בין הרכיבים.

<u>תקציר, הסבר ותוצר:</u>
בספרינט זה, המטרה העיקרית שלנו היא להקים מסגרת תקשורת המאפשרת אינטראקציה משמעותית בין הלקוח לבין ה backend. המשימות כוללות הגדרת פרוטוקולי תקשורת, הגדרת תקשורת שרת-לקוח ושילוב רכיבים חיוניים לשליחת וקבלת בקשות ספציפיות. שלב זה צפוי להימשך שבוע, במהלכו נחקור שיטות תקשורת בין רכיבים, תוך התמקדות ביעילות ובמהירות, וכיצד יבנו בקשות.

למרות שהספרינט הזה לא יהיה נרחב במיוחד, הוא מייצג היבט מכריע של הפרויקט. בלעדיו, הפרויקט חסר את המבנה הבסיסי שלו. לכן, אנו שואפים להשקיע זמן בחקירת הדרכים היעילות ביותר להשגת מטרה זו.

<u>תכולות טכנולוגיות:</u>
בספרינט נעשה בין היתר שימוש בספריות Windows.h ו- WinSocket2.h לצורך בניית התקשורת באמצעות הפרוטוקול שניצור בין ה - Client ל - Backend ובין ה - Backend ל - Server.

פרק 5: עיצוב – מדי איטרציה

איטרציה 2 – מתמקדת במנגנוני ה hash ובניית מסדי הנתונים

<u>החזון שלנו לסיום הספרינט:</u>
עד סוף הספרינט הזה, אנו שואפים שיהיה לנו מנגון hash, וטבלאות hash כגון: (blacklist, whitelist) שהשרת שלנו יתחיל לקיים בקשות בצורה אמיתית.

<u>חלוקת עבודה ולוח זמנים:</u>

במהלך הספרינט הזה, נעבוד בכמה מישורים: יצירת מנגנון ה hash, טבלאות whitelist ו- blacklist של כתובות hash. נחלק את זה בין שנינו באופן הבא: יצירת מנגון hash והשמה של כל המימוש לתשתית, יצירת הטבלאות וחקר על קבצי מערכת.

<u>תקציר, הסבר ותוצר:</u>
היצירה של מסדי הנתונים תתבצע בעזרת הספריה sqlite שתשמש אותנו גם ליצירת הטבלאות whitelist ו- blacklist ב DataBase של השרת וגם ליצירת ה DataBase שיכיל טבלה של whitelist Hashes ב Backend,

כדי שהסריקה שלנו תיהיה יעילה ולא תצטרך לסרוק קבצים שכבר סרקנו ותוכל לזכור אילו קבצים נחשבים בטוחים וניתנים להרצה כפרוססים או אילו פרוססים עלינו לחסום מבלי לסרוק אנו ניצור את הטבלאות הבאות:

Backend DataBase:

1. whitelist Hashes – טבלה המכילה חתימות של קבצים שנסרקו או שהם ידועים באנטיוירוס כקבצים שקיבלו אישור להמשך ריצה מבלי הפרעות מהאנטי וירוס.

#הערה: בין היתר בנוסף לטבלה נצטרך גם מנגנון שידע לזהות אילו קבצים קשורים למערכת הפעלה באמצעות סריקה ובדיקה אילו קבצים קשורים לחבילות (packages) שהמערכת מתקינה בזמן עדכון או חבילות (packages) מותקנות ובדיקה אם החתימה עליהם תקינה ואם כן אז לסמן את כל הקבצים כקבצי מערכת תקינים שלא נצטרך לסרוק.

Server DataBase:

2. Black Hashes – טבלה המכילה חתימות של קבצים שנסרקו או שהם ידועים באנטיוירוס כקבצים זדוניים ועליו לחסום את ריצתם או להשהות אותה.

3. Blacklist IP's – טבלה המכילה כתובות IP המתקשרות עם גורמים זדוניים שנסרקו או שכך הם ידועים באנטיוירוס, ועליו לעצור או להשהות את ריצתם של הפרוססים המתקשרים עם כתובות אלו.

כדי ליצור אלגוריתם Hash שיחשב hash של קבצים, נשתמש בספריית OpenSSL ב-C++. פונקצית ההמרה לוקחת נתיב קובץ ופונקציית hash נבחרת (למשל, SHA-256) כפרמטרים. באמצעות פונקציות OpenSSL, הפוקנצייה תחזיר את ערך ה-hash עבור הקובץ שצוין.

השלב הבא במסגרת הספרינט הנוכחי מתמקד בהטמעת פונקציית ה-Hash ובבניית מסדי נתונים בתשתית הפרויקט. בפועל, יתבצע יצירת בקשות סריקת Hash ושמירתן.

על מנת להפוך את בקשת סריקת ה-Hash למציאות, הלקוח ישלח את הנתיב לקובץ שירצה לסרוק לשרת. השרת, באמצעות פונקציה המותאמת לכך (כפי שצוין למעלה), ימיר את הקובץ לכתובת Hash ויבצע השוואה של תוצאת הפונקציה לעומת מסדי הנתונים השונים. יתבצע סינון ראשית והשוואה בין התוצאות של מסד הנתונים השונים, החל מ-Whitelist וסיומת ב-Blacklist. הסינון יתבצע באמצעות ספריית SQLite ויכלול יצירת שאילתות SQL.

השלב האחרון של הספרינט כולל יישום סריקת חוקי YARA. אם הקובץ עובר בהצלחה את סריקות ה-hash שלנו, הוא ממשיך לעבור סריקה דרך ה-API של YARA. בשלב זה, התוכנה משווה את תוכן הקובץ למחרוזות מוגדרות מראש הקשורות לווירוסים, כפי שזוהו על ידי YARA. מכיוון שקוד זה כתוב בסקריפט ב-Python, אנו נשתמש בפונקציות fork ו-exec בצד השרת כדי להפעיל את קוד Python ולאחזר את התגובה.

פרק 5: עיצוב – מדי איטרציה

איטרציה 3 – מתמקדת בהכנות לסריקות הדינאמיות

<u>החזון שלנו לסיום הספרינט:</u>
עד לסוף הספרינט אנחנו מקווים שיהיה לנו דרייבר שעובד ועוד פיצ'ר של בדיקת syscalls ממוש. בנוסף לזה מקווים שהפרויקט שלנו יוכל לרוץ ברקע עם Daemon ולהתחיל להתנהג כמו אנטי וירוס אמיתי

<u>חלוקת עבודה ולוח זמנים:</u>
במהלך הספרינט הזה נעבוד בכמה מישורים: יצירת driver, יצירת daemon, ומימוש סריקת syscalls.
נחלק את רוב המישורים די ביחד. הדברים שצריך לממש הם די מסובכים ואנחנו רוצים להבין ביחד מה צריך לעשות.

<u>תקציר, הסבר ותוצר:</u>
יצירת ה driver תתבצע באמצעות kernel APi בשפת C. בתוכו אנחנו נממש שלוש פונקציות עיקריות:
פונקציית סגירה - SIGKILL
פונקציית השהייה - SIGSTOP
פונקציות המשך ריצה - SIGCONT

בנוסף כדי שיוכל לקבל בקשות מהשרת המקומי שלנו, נעשה שימוש במנגנון IOCTL אשר מאפשר תקשורת בין USRE SPACE לבין ה KERNEL.

יצירת daemon היא תהליך הפעלה אוטומטית של תוכנית ברקע, כאשר המחשב מתחיל. במקרה שלנו, ה-daemon יריץ את הbackend בצורה אוטומטית, ללא פעולה מצד המשתמש. העיקרון הוא שה-daemon מאפשר לתוכנית לפעול ברקע כל עוד המחשב פועל.

ה backend, מתחיל לסרוק את המערכת ויכתוב לקובץ שהלקוח קורא שינויים מעניינים במערכת.

פרק 5: עיצוב - מדי איטרציה

איטרציה 4 - מתמקדת בפיתוח ממשק גרפי וסריקה דינאמית ראשונה

<u>החזון שלנו לסיום הספרינט:</u>
עד לסוף הספרינט אנחנו מקווים שיהיה לנו את הממשק הגרפי של התוכנה שלנו מה שיסמל את כמעט סיום התוכנה. אנחנו גם מקווים לסיים סריקה דינאמית מ שהיא מחפשת פורטים במצב LISTEN והם מקושרים לתוכנה כלשהי, מה שיתן שימוש לדאמון שבנינו ספרינט קודם.

<u>חלוקת עבודה ולוח זמנים:</u>
במהלך הספרינט הזה נעבוד בכמה מישורים: השלמת פערים מספרינט קודם (זיהוי SYSTEM CALLS), פיתוח ממש גרפי וחיפוש פורטים במצב LISTEN.
את העבודה נחלק באופן הבא:

- השלמת פערים
- פיתוח ממש גרפי
- סריקה ברשת

<u>תקציר, הסבר ותוצר:</u>
את השלמת הפערים שאנחנו צריכים לעשות, שזה ניתוח SYSTEM CALLS נעשה באמצעות ה driver וקוד python. קוד הפייתון מוצא את כתובת טבלת קריאות המערכת, ה driver עושה שימוש בטבלה זו ומשנה את הכתובות של הפעולות OPEN, EXEC שיצביעו על פונקציות שלנו, כך נוכל ליצור מסד נתונים עם תהליכים ומי ומריץ אותם, ולדעת מה הם מבצעים כדי שנוכל לחסום את האפשרות הזו במקרה וזה וירוס.

את סריקת תעבורת הרשת של פרוססים נממש באמתעות שפת פייתון וטכונולוגית scapy שתאפשר לנו לחקור את הרשת.
אנחנו נוציא IP, נשווה אותט למאגר כתובות IP הידועות שאיתם מתקשרים וירוסים, ואם נמצאה התאמה, יהיה ניסיון למצוא מי שלח או קיבל את הפקטה באמצעות ספריית psutil.

בנוסף נסרוק פורטים פתוחים במחשב על מנת לבדוק האם יש תהליך שמנסה להתחבר לכמה פורטים בזמן קצר והחיבור קצר. ככה בודקים אם יש תהליך המנסה להשיג את המחשבים הנוספים הנמצאים באותה רשת.

פרק 5: עיצוב - מדי איטרציה

איטרציה 5- מתמקדת בפיתוח ממשק גרפי וסיום סריקות

<u>החזון שלנו לסיום הספרינט:</u>
עד לסוף האיטרציה אנחנו מקווים להיות בסיום הפרויקט.

<u>חלוקת עבודה ולוח זמנים:</u>
במהלך הספרינט נעבוד בכמה מישורים: פיתוח ממשק גרפי, המשך סריקות רשת והתחלת סריקת פקרים.
נחלק את זה בינינו באופן שווה למרות עומס בית ספרי.

<u>תקציר, הסבר ותוצר:</u>
את הממשק הגרפי אנחנו נממש בסביבת הפיתוח avalonia ובה נפתח בסגנון .NET בשפת #c ו axml.
בהמשך סריקות הרשת אנחנו מתכננים לעשות סריקה אשר בודקת איזה פורטים פתוחים על המחשב והאם הם קשורים בתוכנה/תהליך ספציפי. את זה נממש בשפת python ובעצם ננסה להתחבר בעצמנו לפורטים מסויימים כדי לבדוק האם הם פתוחים.