Technologic Papers

פירוק Anti-Cheat לגורמים

מאת דור נאמני (Dor00tkit)

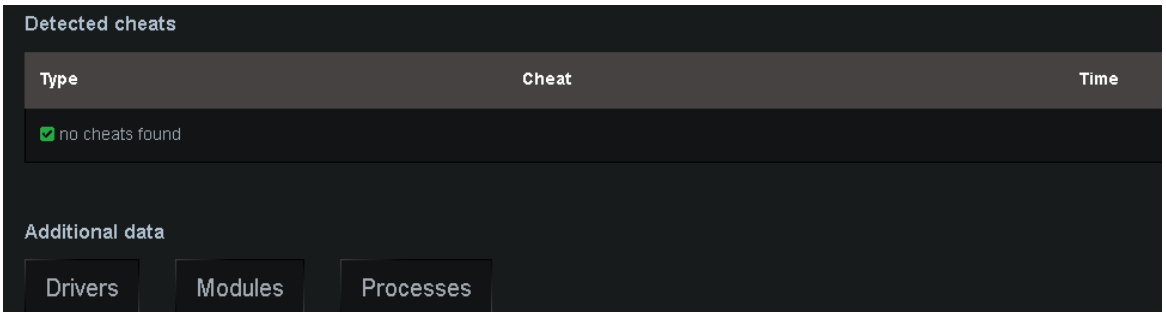
תקציר

אז לא מזמן הייתה לי דודא לשחק Counter-Strike 1.6. וביום בהיר אחד אחרי כמה משחקים טובים האדמין של אותו סרבר חשב שאני משתמש בצ'יטים (דווקא אני שחקן לא רע, כנראה נפלתי על סרבר של נובים) - והוא ביקש ממני להיכנס לאתר fungun.top, להוריד משם תוכנה בשם Easy Cheat Detector, לבצע סריקה ולשלוח לו את התוצאה.

הערה: המאמר מתבסס על ההנחה המוקדמת שיש לקורא היכרות עם הנדסה לאחור, Windows API, שיטות Anti-Debug, שפת C ו-Python.

כמובן שאני לא אוריד תוכנה מפוקפקת למחשב האישי שלי אז הורדתי את זה ל-VM והרצתי. בסיום הסריקה שמתי לב שהתוכנה זיהתה שאני ב-VM:

בנוסף, שמתי לב שבדו"ח יש מידע על התהליכים, ה-DLLs של המשחק, והדרייברים במערכת:



סיקרן אותי להבין איך התוכנה הזאת עובדת מאחורי הקלעים והאם אפשר לערבב אותה. יאללה, בואו נתחיל!

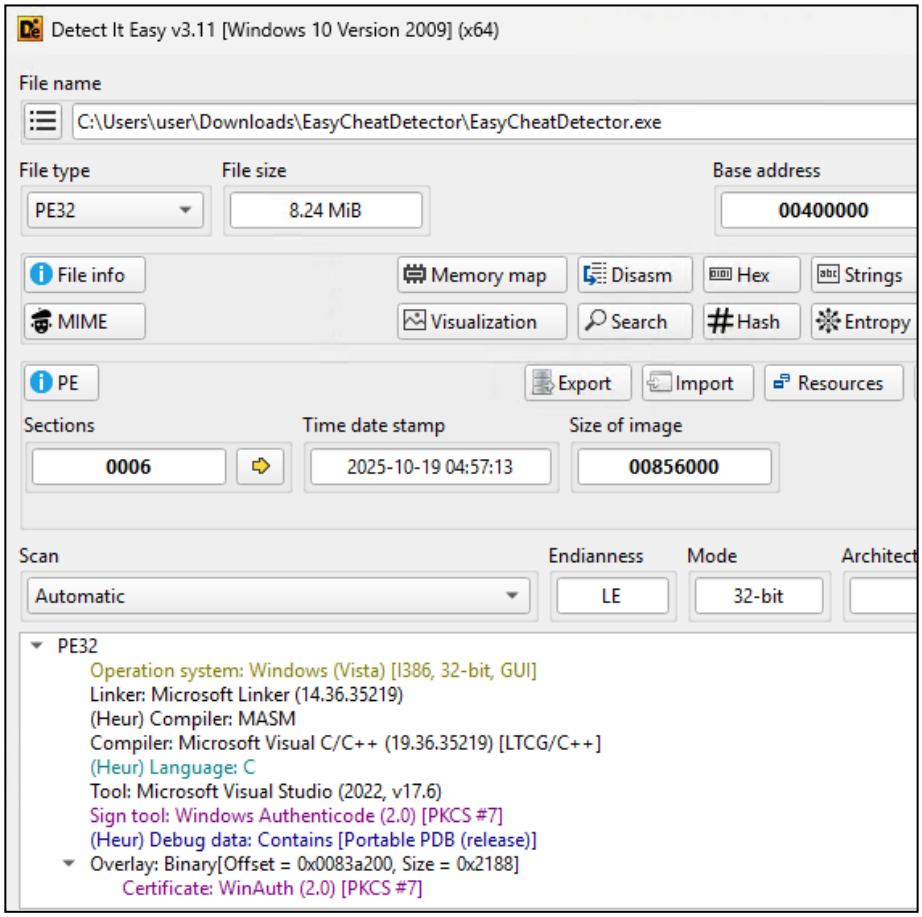
אז עוד לפני שנלכלך את הידיים, שווה להעיף מבט באתר הרשמי (במידה ויש) או בכל מידע אחר אודות אותה תוכנה הזמין באינטרנט. הרבה פעמים זה ייתן לנו לא מעט מידע מקדים. בהסתכלות בעמוד GitHub של התוכנה יש את המידע היבש שציינתי למעלה.

ניתוח מקדים

הערה: המחקר בוצע על גרסאות 2.74 ו-2.82 של Easy Cheat Detector.

הדבר הראשון שעשיתי זה להסתכל על הבינארי באופן באופן יבש ולהבין באיזה שפה הוא נכתב, האם הוא packed, איזה APIs הוא מייבא, האם יש לו עוד Resources וכמובן מחרוזות מעניינות. נראה שהתוכנה נכתבה בשפת C/CPP, קומפלה כ-32 ביט, ויש לה גם חתימה דיגטלית (בתכל'ס מדובר בתעודה חתומה עצמית ככה שזה לא מעיד על האמינות של התוכנה).

לשם כך השתמשתי בכלי Detect It Easy. Detect It Easy (DiE) הוא כלי מתקדם לזיהוי וניתוח סוגי קבצים, הנמצא בשימוש נרחב בקרב חוקרי נוזקות, אנשי אבטחת מידע ומהנדסים לאחור. הכלי משלב מנגנוני זיהוי מבוססי חתימות יחד עם ניתוח היוריסטי, ומאפשר ביצוע בדיקות קבצים מדויקות ויעילות במגוון פלטפורמות, כולל Windows, Linux ו-macOS. כך הוא נראה:



הארכיטקטורה הגמישה שלו, המבוססת על סקריפטים, מספקת יכולת הרחבה והתאמה גבוהה, והופכת אותו לאחד הכלים הוורסטיליים ביותר בתחום. DiE תומך במגוון רחב של פורמטי קבצים, קבצי הרצה, תמונות מערכת הפעלה, אורזים (Packers), מהדרים (Compilers) וסוגי קבצים נוספים, ובכך מאפשר סיווג וניתוח מקיפים של קבצים.

Imports

נעבור לפונקציות המעניינות שהתוכנה מייבאה, יש המון, אחלק את זה לקטגוריות:

- התעסקות עם ה-**Resource**, הפונקציות: FindResource, LoadResource ועוד. בדיקה של המידע הנמצא ב-Resource מראה שיש שם Certificate (הזיהוי לפי ה-Magic number, (הערכים:?? ?? 30 82)). ה-Certificate מוגן בסיסמה. בנוסף יש גם את הפונקציה PFXImportCertStore שיודעת לייבא Certificate.

  התעסקות עם **קבצים**, הפונקציות: CreateFile, ReadFile, SetFilePointer, FindFirstFileEx, FindNextFile - יכולים לשמש לגישה לקבצים של המשחק, למודולים ולדרייברים.
- התעסקות עם **תהליכים**, הפונקציות: First32Process, Next32Process, OpenProcess, VirtualQueryEx, VirtualAllocEx, WriteProcessMemory, ReadProcessMemory - יכולים לשמש לאיסוף מידע על התהליכים הרצים, איסוף מידע על התהליך של המשחק.
- התעסקות עם **Threads**, הפונקציות: Thread32First, Thread32Next, OpenThread, GetThreadContext, SetThreadContext, ResumeThread, SwitchToThread.
- התעסקות עם **Registry**, הפונקציות: RegOpenKeyEx, RegEnumKeyEx, RegCreateKeyEx, RegQueryInfoKey, RegQueryValueEx, RegEnumValue, RegGetValue, RegSetValueEx - יכולים לשמש לאיסוף ואחסון מידע, למשל איסוף מידע על התוכנות המותקנות ודרייברים, שמירת מידע של התוכנה עצמה.
- התעסקות עם Services, הפונקציות: OpenSCManager, OpenService, QueryServiceStatusEx, QueryServiceConfig, EnumDependentServices - יכולים לשמש לאיסוף מידע על הדרייברים וה-Services הקיימים במערכת.
- התעסקות עם **Debugging**, הפונקציות: DebugActiveProcess, WaitForDebugEvent, ContinueDebugEvent, CheckRemoteDebuggerPresent, IsDebuggerPresent - יכולים לשמש לביצוע Debugging על תהליך מסוים, זיהוי של Debugging.

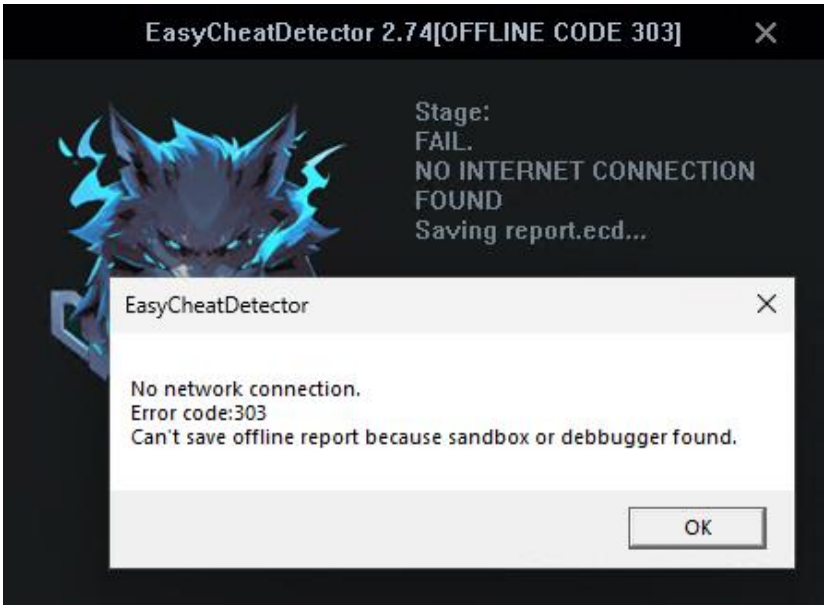
מחרוזות מעניינות

(hacked client?) Antihack executable is corrupted! ECD ANTIHACK RULE ECD Antihack binary is corrupted. FUCKINGHACKERS THIS CUTE BEAST CAN PROTECT ANTIHACK EXECUTABLE FROM BAD HACKERS This antihack tool now support next games: Minecraft (not all clients!), RUST (withot EAC), CS 1.6, CS SOURCE, CS GO, CS 2 **[SELF PROTECT] Found critical error.** blackbone tao::json

ניתן לשים לב שאחת המחרוזות מרמזת שיש לתוכנה מנגנון **Self-Protect** שאמור להגן על עצמה מפני חוקרים כמונו. בהמשך נראה איך ניתן לעקוף את אותו המנגנון. בנוסף ניתן לראות שהמחרוזת **blackbone** מופיעה כמה פעמים, חיפוש בגוגל מגלה שזו למעשה ספרייה נוחה המאפשרת לגשת לזיכרון של תהליכים. ולסיום, המחרוזת האחרונה ברשימה מגלה שנעשה שימוש בספרייה נוספת בשם taoJSON.

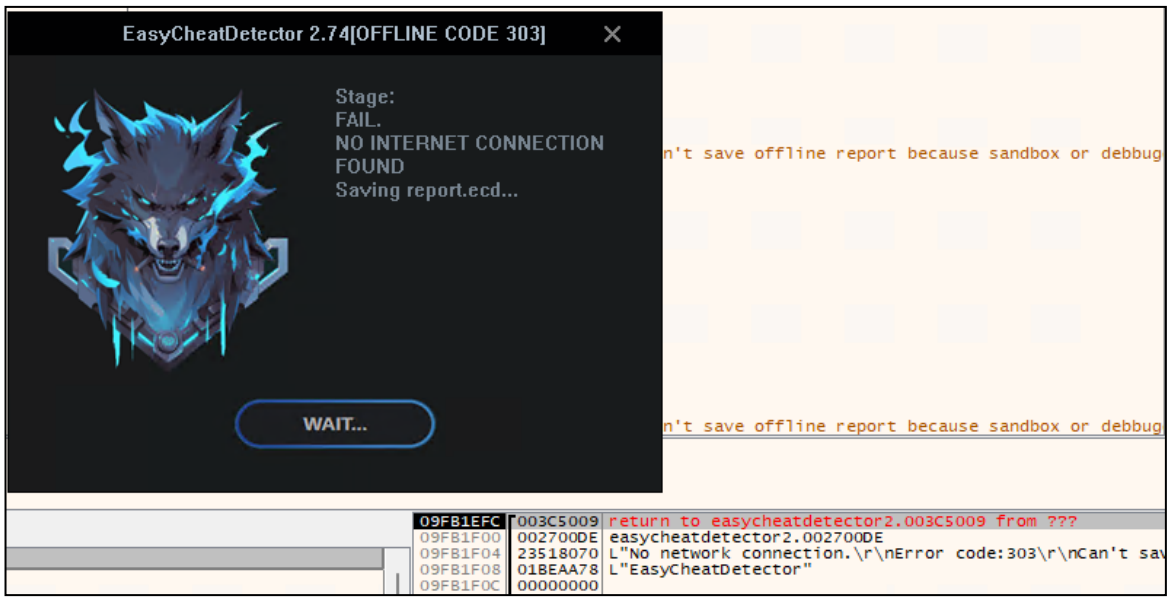
ניתוח משולב - סטטי ודינמי

את המחקר הסטטי נבצע באמצעות IDA Pro, ואת המחקר הדינמי בעזרת x32dbg. הדבר השני שניסיתי זה לראות איך התוכנה מתנהגת שאין אינטרנט. ככה לרוב אני מעדיף להתחיל מחקר בסביבה מבודדת, שלא תדליף מידע על הסביבה שאני עובד בה. נריץ את התוכנה, נלחץ Scan, התוכנה תרוץ ותסרוק ו..



נראה שהתוכנה מזהה שאנחנו רצים בתוך VM ולכן מסרבת לשמור את הדו"ח מקומית.

חיפוש אחרי המחרוזת הזאת באופן סטטי לא מביאה תוצאות. כנראה היא מפוענחת רק בזמן ריצה. לפי החלון שקפץ נראה שנעשה שימוש בפונקציה MessageBox על מנת להציג את השגיאה הזאת. נפתח Debugger, נשים Breakpoint על MessageBox ונריץ...



כעת אפשר להתחיל לעקוב וללכת קצת אחורה.

כשניסיתי לעקוב אחרי הקטע קוד ב-IDA Pro אז קיבלתי את השגיאה הידועה לשמצה " decompilation failure too big function". נסדר את זה, נחכה כמה דקות טובות עד ש IDA תסיים לדקמפל את הפונקציה הענקית. ונוכל להמשיך במחקר. נראה שקצת לפני שמגיעים לקוד שקורא לפונקציה MessageBox נבדקים כמה משתנים גלובלים:

```
else if ( byte_6E97AD || byte_6D292A || byte_6E97AF || v7129 )
{
  v2935 = sub_246430(dword_6E9854);
  v2949 = sub_2219C0(v8498);
  v2948 = v2949;
  v8896 = 0xC30;
  v2939 = v2949;
  v2947 = sub_1253F0(v8497, dword_6E97B0);
  v2946 = v2947;
  LOBYTE(v8896) = 0x31;
  v2943 = v2947;
  v2945 = sub_221800(v8496);
  v2944 = v2945;
  LOBYTE(v8896) = 0x32;
  v2942 = v2945;
  v2941 = sub_235ED0(v8495, v2945, v2943);
  v2940 = v2941;
  LOBYTE(v8896) = 0x33;
  v2938 = v2941;
  v2937 = sub_235ED0(v8494, v2941, v2939);
  v2936 = v2937;
  LOBYTE(v8896) = 0x34;
  v2934 = sub_246430(v2937);
  v2933 = hWndParent;
  MessageBoxW(hWndParent, v2934, v2935, 0);// Can't save offiline report because sandbox or debugger found.
  LOBYTE(v8896) = 0x33;
  sub_246A50(v8494);
  LOBYTE(v8896) = 0x32;
  sub_246A50(v8495);
  LOBYTE(v8896) = 0x31;
```

אם נעקוב אחרי הרפרנסים של כל אחד מהם, נראה שהם נבדקים המון פעמים. נתחיל לפרק אחד אחד.

זיהוי Debugger או VM על פי זמן

נעקוב אחרי המשתנה הגלובלי byte_6E97AD, נראה שהמון קטעי קוד המאתחלים אותו ל-**1** נראים כך:

```
v171 = GetTickCount() - TickCount;
v172 = 0;
if ( v171 > qword_6E97C0 )
  qword_6E97C0 = GetTickCount() - TickCount;
if ( GetTickCount() - TickCount > 0x7D0 && GetTickCount() - TickCount < 0x9C40 )
{
  byte_6E97AD = 1;
  LOBYTE(v274) = 3;
```

השיטה הזו היא שיטת זיהוי Debugger / VM דיי מוכרת. התוכנה קוראת לפונקציה GetTickCount בפעם הראשונה (שלב זה אינו מופיע בתמונה), מבצעת קטע קוד, ולאחר מכן קוראת שוב ל-GetTickCount. בהמשך מתבצעת השוואה בין התוצאות של הקריאות כדי למדוד כמה זמן לקח לקטע הקוד לרוץ.

כאשר מריצים את התוכנה תחת Debugger, ובמיוחד אם משתמשים ב-Breakpoints או ב-Single-step, הביצוע מואט באופן משמעותי. באופן דומה, גם ריצה בתוך VM עשויה להיות איטית יותר עקב Overhead של וירטואליזציה, שתלוי בקונפיגורציית ה-VM ובמערכת ה-Host. אם הזמן שנמדד חורג מהזמן הצפוי להרצת אותו קטע קוד, התוכנה יכולה להסיק שהיא רצה תחת Debugger או בתוך סביבה וירטואלית.

דרכים אפשריות לעקוף את הבדיקה:

1. פשוט, לא לשים Breakpoints ולא לבצע Single-step בטווח הקוד של הבדיקות.
2. לקנפג את ה-VM עם חומרה טובה. אצלי הבדיקה עוברת.
3. הוק לפונקציה GetTickCount שתחזיר ערכים מפוברקים.
4. פיצפוץ הקוד. נוכל לפצפץ את החלק שבודק, את החלק המאתחל את המשתנה הגלובלי ל-1 שיאתחל אותו ל-0, וכד'.

המשתנה הגלובלי השני הוא byte_6D292A, הוא גם כן נמצא באותם אזורים שבודקים את byte_6E97AD. בניגוד ל-byte_6E97AD שמאותחל ל-0, המשתנה הגלובלי byte_6D292A מאותחל ל-1, ורוב הכתיבות אליו מאתחלות אותו ל-0:

xrefs to byte_6D292A

Direct	Type	Address	Text
Up	r	sub_1702C0+27E	mov al, byte_6D292A
Up	w	sub_168B70+63	mov byte_6D292A, 0
Up	w	sub_1702C0:loc_170537	mov byte_6D292A, 0
...	w	sub_1B6AF0+63	mov byte_6D292A, 0
...	w	sub_1B9680+63	mov byte_6D292A, 0
...	w	sub_1D6340+63	mov byte_6D292A, 0
...	w	sub_1F5970+63	mov byte_6D292A, 0
...	w	sub_1F9280+63	mov byte_6D292A, 0
...	w	sub_20CC00+63	mov byte_6D292A, 0
...	w	sub_2106E0+63	mov byte_6D292A, 0
Up	r	sub_1702C0+44832	movzx eax, byte_6D292A
Up	r	sub_1702C0:loc_1ADAF1	movzx ecx, byte_6D292A
...	r	sub_1702C0+45482	movzx ecx, byte_6D292A
Up	r	sub_1702C0+3C0BC	movzx edx, byte_6D292A
Up	r	sub_1702C0+44153	movzx edx, byte_6D292A
Up	r	sub_1702C0+44BC6	movzx edx, byte_6D292A

רוב האתחולים של byte_6D292A נראים ככה:

```
v92 = this;
v273 = 0;
v272 = 0;
v170 = sub_55B799() % 0x32 + 0x19;
byte_6D292A = 0;
v1 = sub_55B799();
v3 = v1 % 0x32;
result = v1 / 0x32;
if ( v3 <= 0x14 )
{
  for ( i = 0; ; ++i )
  {
    result = i;
    if ( i >= v170 )
      break;
    if ( sub_55B799() % 0x32 <= 0xF )
    {
      TickCount = GetTickCount();
      v266 = sub_55B799();
      sub_54B7A0(v326, 0, 0x100);
      sub_54B7A0(v325, 0, 0x200);
```

אבל יש אחד שמשך את תשומת הלב שלי והוא נראה ככה:

```
sub_50360();
qword_6E7768 = sub_2E59C0(dword_6E77A0);
if ( sub_31E050() )
{
  byte_6D292A = 0;
  byte_6E97AD = 0;
}
```

למעשה ניתן לראות שגם byte_6E97AD מאותחל ל-0 במידה והפונקציה sub_31E050 מחזירה ערך שונה מ-0.

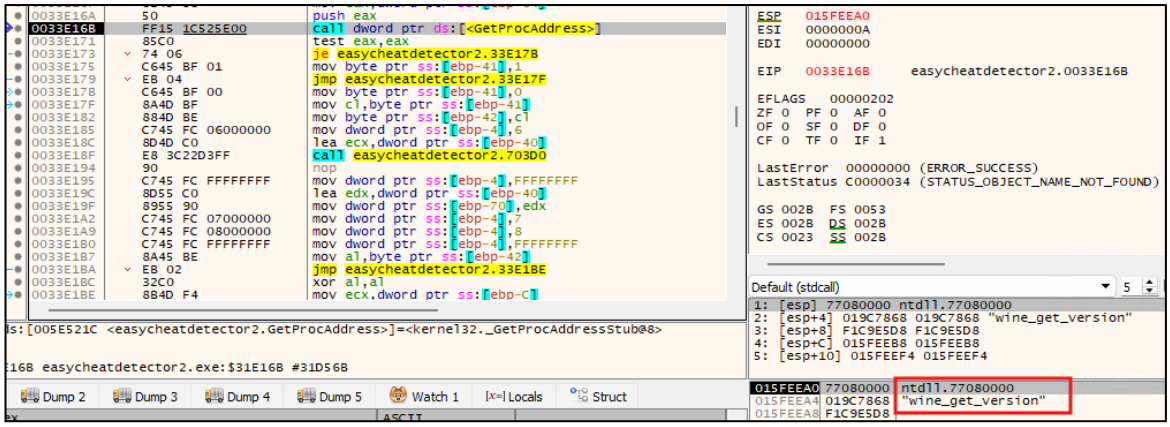
יין ישמח לבב אנוש

חיפוש רפרנסים לפונקציה sub_31E050 מראה שהיא נקראת לא מעט פעמים. ככה הפונקציה נראת:

```
bool sub_31E050()
{
  const CHAR *lpProcName; // [esp+14h] [ebp-60h]
  int v2; // [esp+1Ch] [ebp-58h]
  const WCHAR *lpModuleName; // [esp+24h] [ebp-50h]
  _DWORD *v4; // [esp+28h] [ebp-4Ch]
  bool v5; // [esp+32h] [ebp-42h]
  _BYTE v6[24]; // [esp+34h] [ebp-40h] BYREF
  _BYTE v7[24]; // [esp+4Ch] [ebp-28h] BYREF
  int v8; // [esp+70h] [ebp-4h]

  if ( dword_6FB7F0 > *(*NtCurrentTeb()->ThreadLocalStoragePointer + 4) )
  {
    sub_549732(&dword_6FB7F0);
    if ( dword_6FB7F0 == 0xFFFFFFFF )
    {
      v8 = 0;
      v4 = sub_31E1E0(v7);
      LOBYTE(v8) = 1;
      lpModuleName = sub_125A90(v4);
      dword_6FBA08 = GetModuleHandleW(lpModuleName);
      LOBYTE(v8) = 2;
      sub_2622E0(v7);
      v8 = 0xFFFFFFFF;
      sub_5496E1(&dword_6FB7F0);
    }
  }
  if ( !dword_6FBA08 )
    return 0;
  v2 = sub_31E2B0(v6);
  v8 = 5;
  lpProcName = sub_219FC0(v2);
  v5 = GetProcAddress(dword_6FBA08, lpProcName) != 0;
  v8 = 6;
  sub_503D0(v6);
  v8 = 0xFFFFFFFF;
  return v5;
}
```

נראה שהפונקציה מנסה למצוא פונקציה מסוימת ע"י שימוש בפונקציה GetProcAddress, מכיוון שהמחרוזות מפוענחות בזמן ריצה הדרך המהירה למצוא את השם של הפונקציה שאותה מחפשים היא בזמן ריצה, למשל באמצעות Debugger:



אז נראה שהתוכנית מחפשת אחרי הפונקציה wine_get_version בתוך Ntdll. לפי מה שזכור לי לא אמורה להיות פונקציה כזאת ב-Ntdll. האם Wine אומר לכם משהו? (גם אם לא, אז חיפוש קצר בגוגל מביא תוצאות טובות) לי כן. Wine הוא כלי שמאפשר להריץ תוכנות Windows במערכות מבוססות Linux ללא צורך ב-VM. האם יש מצב שכותב התוכנה משתמש במערכת מבוססת Linux בשביל לדבג את התוכנה בלי כל/חלק מההגנות, או על מנת לדלג על חלקים מסוימים? או שיש שחקנים שמריצים את המשחק על גבי Wine?

זיהוי Debugger באמצעות Debug Registers

נמשיך עם המשתנה הגלובלי השלישי, byte_6E97AF, חיפוש רפרנסים אליו מביא את הרפרנס המעניין הבא:

```
CurrentThreadId = GetCurrentThreadId();
v6797 = OpenThread(0x1FFFFFu, 0, CurrentThreadId);
if ( v6797 )
{
  sub_54B7A0(&v7164, 0, 0x2CC);
  v7164.ContextFlags = 0x10010;
  memset(&v7164.Dr0, 0, 0xC);
  v7164.Dr3 = &v7164;
  v7164.Dr7 &= 0xFFFFFFAA;
  SetThreadContext(v6797, &v7164);
  GetThreadContext(v6797, &v7164);
  if ( v7164.Dr0 || v7164.Dr1 || v7164.Dr2 || v7164.Dr3 != &v7164 )
    byte_6E97AF = 1;
  memset(&v7164.Dr0, 0, 0x10);
  v7164.Dr7 &= 0xFFFFFFAA;
  SetThreadContext(v6797, &v7164);
  CloseHandle(v6797);
}
```

גם זאת שיטה דיי מוכרת לזיהוי Debugger. בקצרה, שיטה זאת בודקת התערבות של Debugger ע"י כתיבה וקריאה חוזרת של Debug Registers ב-Thread הנוכחי, כאשר כל שינוי בלתי צפוי בערכים מצביע על שימוש ב-Hardware breakpoints המעיד על הרצת התוכנה תחת Debugger.

דרכים לעקוף את הבדיקה

1. פשוט, לא להשתמש ב-Hardware breakpoints רק עד לאחר סיום ריצת אותו קטע קוד של הבדיקה.
2. לבצע הוק לפונקציה GetThreadContext ולפברק את הערכים שהיא מחזירה. בנוסף נצטרך גם להגן על ה-Debug Registers מפני שינוי לא רצוי באמצעות הוק לפונקציה SetThreadContext.
3. פיצפוץ הקוד. נוכל לפצפץ את החלק שדורס, שבודק, ואת החלק המאתחל את המשתנה הגלובלי ל-**1** שיאתחל אותו ל-**0**, וכד'.

יש לא מעט רפרנסים שמאתחלים את המשתנה v7129. נתחיל עם הפונקציה sub_2F3030:

```
char sub_2F3030()
{
  HANDLE hObject; // [esp+0h] [ebp-4h]

  hObject = CreateFileW(L"\\\\?\\\\A3E64E55_fl", GENERIC_READ, 0, 0, OPEN_EXISTING, 0, 0);
  if ( hObject == 0xFFFFFFFF )
    return FALSE;
  CloseHandle(hObject);
  return TRUE;
}
```

מה זאת הגישה הזאת לנתיב המוזר הזה?

```
\\?\\A3E64E55_fl
```

חיפוש בגוגל מראה שזה שייך ל-ANY.RUN שהוא Sandbox מוכר מאוד.

```
int sub_2F3070()
{
  _DWORD v1[6]; // [esp-20h] [ebp-8Ch] BYREF
  int v2; // [esp-8h] [ebp-74h]
  int v3; // [esp-4h] [ebp-70h]
  _BYTE *v4; // [esp+8h] [ebp-64h]
  _DWORD *v5; // [esp+Ch] [ebp-60h]
  _DWORD *v6; // [esp+10h] [ebp-5Ch]
  int v7; // [esp+14h] [ebp-58h]
  _DWORD *v8; // [esp+18h] [ebp-54h]
  _DWORD *v9; // [esp+1Ch] [ebp-50h]
  int v10; // [esp+20h] [ebp-4Ch]
  int v11; // [esp+24h] [ebp-48h]
  char v12; // [esp+29h] [ebp-43h] BYREF
  unsigned __int8 v13; // [esp+2Ah] [ebp-42h]
  char v14; // [esp+2Bh] [ebp-41h]
  _BYTE v15[24]; // [esp+2Ch] [ebp-40h] BYREF
  _BYTE v16[24]; // [esp+44h] [ebp-28h] BYREF
  int v17; // [esp+68h] [ebp-4h]

  v7 = 0;
  v3 = 0;
  v2 = 0;
  v11 = sub_2F3190(&v12, v15);
  v10 = v11;
  v17 = 0;
  v6 = v1;
  v5 = sub_2D85D0(v1, v11, &unk_6E99A0);
  LOBYTE(v17) = 0;
  v9 = sub_2B9E30(v16, v1[0], v1[1], v1[2], v1[3], v1[4]);
  v8 = v9;
  LOBYTE(v17) = 2;
  v14 = sub_2C1C90(v9, v2, v3);
  v13 = v14;
  LOBYTE(v17) = 3;
  sub_2622E0(v16);
  v4 = v16;
  v17 = 6;
  sub_2622E0(v15);
  return v13;
}
```

הפונקציה מפענחת בזמן ריצה מחרוזת ומנסה לבדוק האם הנתיב הבא קיים:

```
C:\Program Files\KernelLogger
```

שגם הוא מסתבר שייך ל-ANY.RUN. הפונקציה sub_2F3400:

```
v23 = 0;
sub_145030(v29, L"srvpost.exe");
dwProcessId = sub_2D9B50(v29);
sub_2622E0(v29);
v24 = 0xFFFFFFFF;
if ( !dwProcessId )
  return 0;
sub_3696D0(v25);
v24 = 4;
if ( sub_369C80(dwProcessId, 0x438u) < 0 )
{
  v24 = 0xFFFFFFFF;
  sub_369B00(v25);
  return 0;
}
v14 = v26;
v13 = v26;
sub_145030(v27, L"winanr.dll");
LOBYTE(v24) = 5;
v23 |= 1u;
v12 = sub_36CCC0(v1, v27, 0, 2);
v11 = v12;
v24 = 6;
v23 |= 2u;
v10 = *v12;
v17 = v10 != 0;
v22 = v10 != 0;
if ( v10 )
  goto LABEL_6;
v9 = v26;
v8 = v26;
sub_145030(v28, L"winsanr.dll");
```

הפונקציה בודקת האם יש במערכת תהליך בשם srvpost.exe והאם המודולים winanr.dll ו-winanr.dll טעונים. שוב, חיפוש בגוגל מביא עוד פעם תוצאה שזה שייך ל-ANY.RUN.

בהמשך מגיעים לקטע קוד הבא:

```
v3549 = (void *)sub_20C050((int)v7868);
v3548 = v3549;
LOBYTE(v8466) = 0xD5;
v268 = (const WCHAR *)sub_246430(v3549);
ModuleHandleW = GetModuleHandleW(v268);       // cmdvrt32.dll
LOBYTE(v8466) = 0xC8;
sub_246A50(v7868);
if ( ModuleHandleW )
{
  v3546 = 7;
  v466[0xF4] = &v331;
  v466[0xF3] = sub_236F80(&v3546);
  LOBYTE(v8466) = 0xD6;
  v466[0xF2] = &v327;
  v466[0xF1] = sub_20C140(&v327);
  LOBYTE(v8466) = 0xD6;
  v3545[1] = sub_2425E0(&unk_6E99D8, v327, v328, v329, *(int *)v330, *(int *)&v330[4], *(int *)&v330[8]);
  LOBYTE(v8466) = 0xC8;
  sub_2429F0(v331, v332, v333, v334);
  sandbox_vm_checks = 1;
}
```

בזמן ריצה מפוענחת המחרוזת cmdvrt32.dll ומנסים לבדוק האם ה-DLL הזה טעון בתהליך הנוכחי. גם הפעם נעזרתי בגוגל כדי להבין לאיזה מוצר זה שייך והתוצאות הצביעו על Comodo Antivirus.

בהמשך יש את הקטע קוד הבא:

```
 }
 if ( (unsigned __int8)SetCalendarInfoA != 0xE9
   && (unsigned __int8)SetLocaleInfoW == 0xE9
   && (unsigned __int8)SetLocaleInfoA == 0xE9 )
 {
   v3545[0] = 1;
   v466[0xF0] = &v331;
   v466[0xEF] = sub_236F80(v3545);
   LOBYTE(v8466) = 0xD8;
   v466[0xEE] = &v327;
   v466[0xED] = sub_20C1E0(&v327);
   LOBYTE(v8466) = 0xD8;
   v3544[1] = sub_2425E0(&unk_6E99D8, v327, v328, v329, *(int *)v330, *(int *)&v330[4], *(int *)&v330[8]);
   LOBYTE(v8466) = 0xC8;
   sub_2429F0(v331, v332, v333, v334);
   sandbox_vm_checks = 1;
 }
 else if ( (unsigned __int8)SetCalendarInfoA != 0xE8
        && (unsigned __int8)SetLocaleInfoW == 0xE8
        && (unsigned __int8)SetLocaleInfoA == 0xE8 )
 {
   v3544[0] = 1;
   v466[0xEC] = &v331;
   v466[0xEB] = sub_236F80(v3544);
   LOBYTE(v8466) = 0xDA;
   v466[0xEA] = &v327;
   v466[0xE9] = sub_20C360(&v327);
   LOBYTE(v8466) = 0xDA;
   v3543[1] = sub_2425E0(&unk_6E99D8, v327, v328, v329, *(int *)v330, *(int *)&v330[4], *(int *)&v330[8]);
   LOBYTE(v8466) = 0xC8;
   sub_2429F0(v331, v332, v333, v334);
   sandbox_vm_checks = 1;
```

הקטע קוד הזה בודק האם יש הוקים לפונקציות הבאות:

SetCalendarInfoA, SetLocaleInfoW, SetLocaleInfoA

אולי זה גם רלוונטי ל-Comodo Antivirus או ל-Sandbox כלשהו שמבצע הוק לפונקציות האלה.

VMware

הפונקציה sub_2F4630:

```
v5 = 0;
v13 = sub_2F4810(v22);                          // "vmtoolsd.exe"
v12 = v13;
v3 = 0;
v11 = sub_2D9C10(v1, v13);
v10 = v11;
LOBYTE(v3) = 1;
v15[2] = v11;
v9 = (v11[1] - *v11) >> 2;
v8 = v9;
LOBYTE(v3) = 0;
sub_2597B0(v1);
v3 = 2;
sub_2622E0(v22);
v4 = v22;
v3 = 0xFFFFFFFF;
if ( v8 )
  return 1;
v16 = 0;
v3 = 5;
v7 = sub_2F4900(v21);                           // "SOFTWARE\\VMware, Inc.\\VMware Tools"
v6 = v7;
LOBYTE(v3) = 6;
sub_13DF20(&v16, v15, HKEY_LOCAL_MACHINE, v7, 0x20119u);
```

ניסיון לבדוק האם רצים מעל VMware באמצעות חיפוש אחרי תהליך vmtoolsd.exe. ובדיקה נוספת אחרי המפתח הבא ברגסטרי:

```
HKLM\SOFTWARE\VMware, Inc.\VMware Tools
```

VirtualBox

הפונקציה sub_2F4190:

```
char sub_2F4190()
{
  _BYTE v1[12]; // [esp+4h] [ebp-88h] BYREF
  _BYTE *v2; // [esp+14h] [ebp-78h]
  int v3; // [esp+18h] [ebp-74h]
  int v4; // [esp+1Ch] [ebp-70h]
  _DWORD *v5; // [esp+20h] [ebp-6Ch]
  _DWORD *v6; // [esp+24h] [ebp-68h]
  int v7; // [esp+28h] [ebp-64h]
  int v8; // [esp+2Ch] [ebp-60h]
  LPCWSTR lpFileName; // [esp+30h] [ebp-5Ch]
  _DWORD *v10; // [esp+34h] [ebp-58h]
  _DWORD *v11; // [esp+38h] [ebp-54h]
  BOOL v12; // [esp+3Ch] [ebp-50h]
  _DWORD *v13; // [esp+40h] [ebp-4Ch]
  HANDLE hObject; // [esp+44h] [ebp-48h]
  bool v15; // [esp+48h] [ebp-41h]
  _BYTE v16[24]; // [esp+4Ch] [ebp-40h] BYREF
  _BYTE v17[24]; // [esp+64h] [ebp-28h] BYREF
  int v18; // [esp+88h] [ebp-4h]

  v3 = 0;
  v11 = sub_2F4320(v17);                        // "\\.\VBoxMiniRdrDN"
  v10 = v11;
  v18 = 0;
```

התת מחרוזת "VBox" המופיעה במחרוזת ככל הנראה דיי מסגירה את VirtualBox אך כדי להיות בטוח גם פה השתמשתי בגוגל.

זיהוי VM גנרי

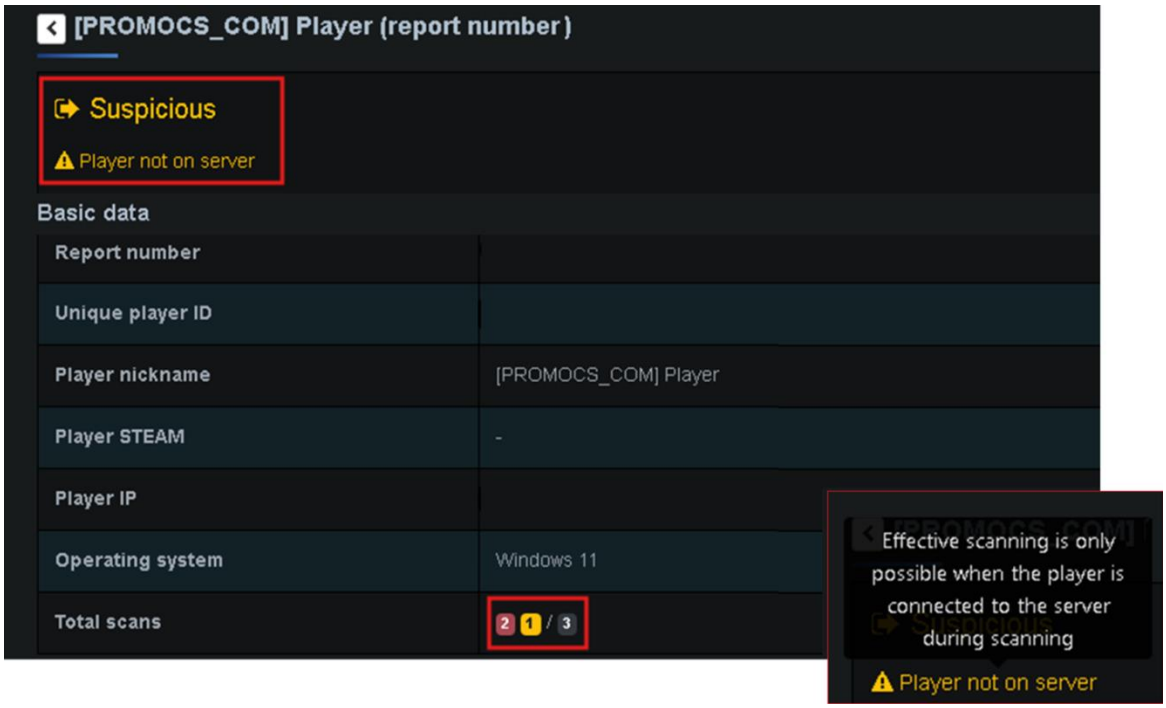
הפונקציה sub_2F3720:

```
v49 = 1;
v13 = sub_2F3BD0(v54);                  // "SYSTEM\HardwareConfig\Current"
v12[1] = v13;
sub_13DF20(&v38, v12, HKEY_LOCAL_MACHINE, v13, 0x20119u);// KEY_READ | KEY_WOW64_64KEY
LOBYTE(v49) = 2;
sub_2622E0(v54);
v6 = v54;
LOBYTE(v49) = 0;
v37 = v12[0] == 0;
v41 = v12[0] == 0;
v47 = v12[0] == 0;
if ( !v12[0] )
{
  v11 = sub_2F3D60(v53);                // "SystemFamily"
  v10 = v11;
  LOBYTE(v49) = 5;
  sub_13E0A0(&v38, v9, v11);            // call to RegGetValueW()
  LOBYTE(v49) = 8;
  sub_2622E0(v53);
  v5 = v53;
  LOBYTE(v49) = 7;
  if ( !sub_238850(v9) )
    goto LABEL_5;
  v34 = sub_2F3E50(v52);                // "Virtual Machine"
  v33 = v34;
```

פונקציה זאת מבצעת בדיקה על סוג המשפחה של היצרן של המחשב (לדוגמה: Latitude, רלוונטי ל-DELL) ע"י שליפת המידע מתוך המפתח הבא בר'גסטרי:

```
HKLM\SYSTEM\HardwareConfig\Current
```

ה-VM שלי רץ על Hyper-V ואכן הערך של SystemFamily אצלי הוא "Virtual Machine". בהמשך, הפונקציה גם בודקת את הערך של SystemProductName. נערוך את הערכים האלה - ננסה לסרוק שוב:



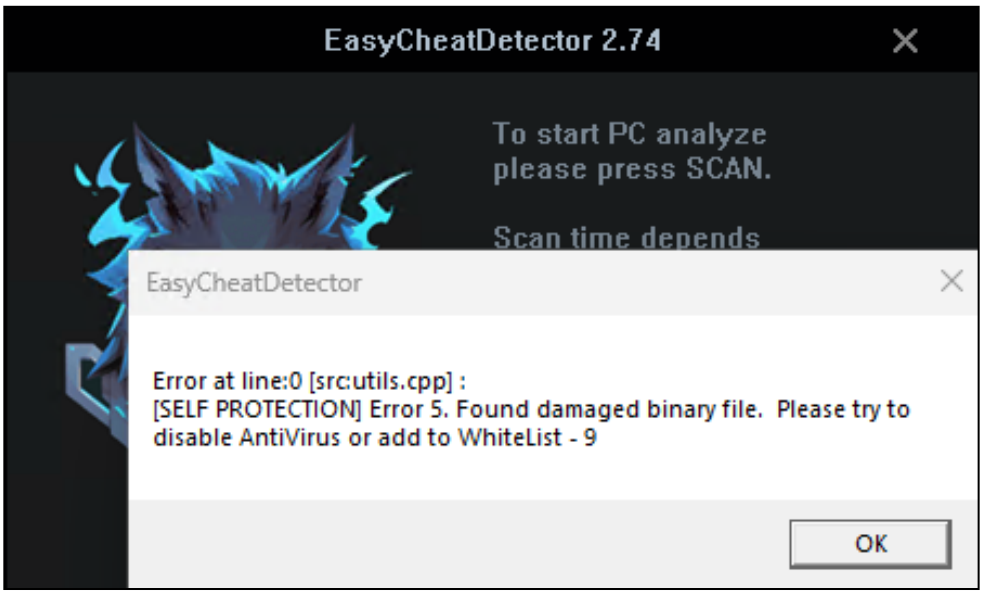
הצלחנו לעקוף את הזיהוי של ה-VM! בנוסף, ניתן לראות את מספר הפעמים שאותו שחקן ביצע סריקה, כלומר שומרים את ההיסטוריה של הסריקות.

מנגנון Self Protect

אגב אם אתם זוכרים, אחת המחרוזות המעניינות שהופיעה היא:

```
[SELF PROTECT] Found critical error.
```

בינתיים כל הבדיקות שסקרנו מטריגים את ההודעה על "Can't save offline report because sandbox or debugger found" בהתחלה, אך לא את המחרוזות שמכילה את ה-"SELF PROTECT". לאחר ששמתי Breakpoint בקוד של התוכנה עצמה (בניגוד ל-Breakpoint ששמתי ב-DLL, למשל **kernel32.dll**) השגיאה הבאה קפצה:



כזכור, בעת שימוש ב-Software breakpoint ה-Debugger למעשה מפצפץ את ההוראה ומחליף אותה בהוראה INT3.

מנגנון ה-SELF PROTECT של התוכנה מזהה שהקוד פוצפץ ומקפיץ את השגיאה. בניתוח דינמי באמצעות Debugger אחד הפיצ'רים הבסיסיים והחיוניים הוא שימוש ב-Software breakpoint, כמובן שאפשר להשתמש גם ב-Hardware breakpoint אך כזכור החיסרון העיקרי הוא שהם מוגבלים ל-4 בלבד (מגבלה זאת מתייחסת למעבדי Intel). ישנה סוג של "תחלופה" (לא לגמרי) נוספת והיא שימוש ב-EPT Hooking אבל זה נראלי יותר מתאים למרחיקי לכת.

אז כדי שנוכל לעבוד כמו בני אדם נצטרך למצוא היכן המנגנון הזה ולהשבית אותו. כדי למצוא את המנגנון הזה אני בחרתי באותה השיטה שהצגתי בהתחלה, והיא לשים Breakpoint על הפונקציה MessageBox ולבחון את ה-Stack trace ולראות מי קרא לה.

זאת הפונקציה:

```
if ( sub_2CDEF0(&v42) )
{
  qword_6E97D0 = 0LL;
  memset(&Msg, 0, sizeof(Msg));
  sub_145030(v88, L"-noupdate");
  LOBYTE(v76) = 0x33;
  v70 = sub_2D16F0(v89, v88);
  LOBYTE(v76) = 0x34;
  sub_2622E0(v88);
  v15 = v88;
  LOBYTE(v76) = 0x23;
  if ( v70 )
  {
    byte_6E7782 = 1;
    dword_6E7710 = GetTickCount() + 0x64;
  }
  hThread = CreateThread(0, 0, sub_224BF0, 0, 0, 0);
  while ( GetMessageW(&Msg, 0, 0, 0) )
  {
    TranslateMessage(&Msg);
    DispatchMessageW(&Msg);
  }
  GdiplusShutdown(v43);
  LOBYTE(v76) = 0x37;
  sub_2622E0(v89);
  return 0;
}
else
{
  v30 = v42 + 0x64;
  LOBYTE(v76) = 0x24;
  v17 = sub_22FB30(v87, v42 + 0x64);
  v69 |= 1u;
  v31 = v87;
  v32 = sub_232A10(v81, "Error ", v87);
  v33 = v32;
  LOBYTE(v76) = 0x25;
  v34 = sub_219FC0(v32);
  v46 = v34;
  v35 = sub_22F800(v80);
  v41 = v35;
  LOBYTE(v76) = 0x26;
  v47 = sub_219FC0(v35);
  lpText = v47;
  MessageBoxA(0, v47, v46, 0);  // "Found damaged binary file. Please try to disable AntiVirus or add to WhiteList"
```

אז הפונקציה שעושה את כל הבלאגן היא sub_2CDEF0 ולפי הרפרנסים אליה היא נקראת לא מעט פעמים. מדובר בפונקציה גדולה יחסית, כאשר החלק המרכזי שלה מבצע קריאה של קובץ התוכנה מהדיסק לזיכרון, ולאחר מכן משווה מקטעים מהקוד שרץ בתהליך אל מול המידע המקביל בקובץ שנקרא מהדיסק. במקרה שבו מזוהה פיצפוץ, הפונקציה מחזירה FALSE, והתוכנה תציג את השגיאה שראינו למעלה.

דרכים לעקוף את הבדיקה

1. פיצפוץ פקודת ההשוואה בין האוגרים, כך שתוצאת בדיקת השוויון תמיד תעבור בהצלחה.
2. פיצפוץ הפונקציה שתחזיר **TRUE** כבר בהתחלה.
3. פיצפוץ הקריאות לפונקציה.

פיצ'רים מעניינים

בחלקים הקרובים נסקור דברים מעניינים שהתוכנה עושה. התוכנה עצמה דיי עוזרת לנו ואומרת לנו בכל שלב מה היא עושה (אמנם היא מפענחת את המחרוזות בזמן ריצה אבל זה לא באמת מקשה עלינו):

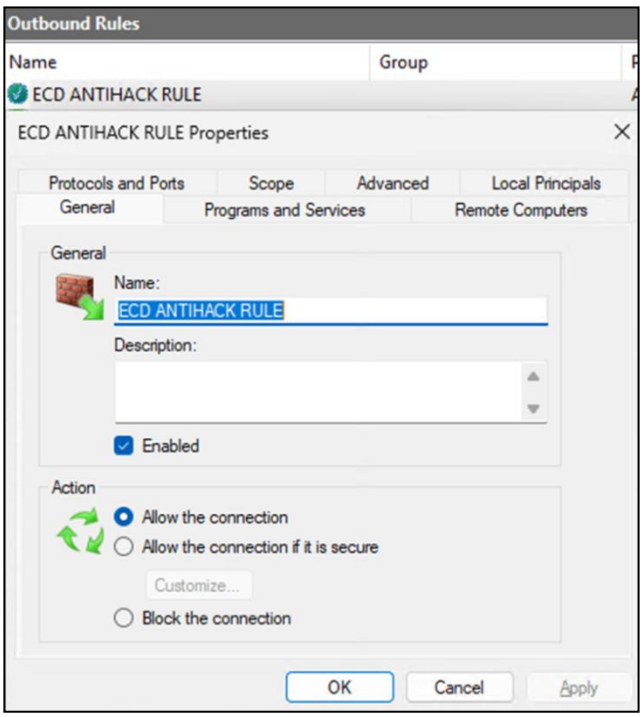
Place	Description
00170F3E: scan+C7E	"01 - Stage: Search game process"
00172F22: scan+2C62	"02 - Stage: Search game installation"
00178C4B: scan+898B	"03 - Stage: Game found PID: "
00179195: scan+8ED5	"04 - Stage: Check for admin [4]"
00179742: scan+9482	"05 - Stage: Adding firewall rules"
00179841: scan+9581	"06 - Stage: Scan processes"
0017BE0E: scan+BB4E	"07 - Stage: Access WMI repository..."
0017C063: scan+BDA3	"08 - Stage: Preparing req"
0017C94F: scan+C68F	"09 - Stage: Read db"
0017DCCA: scan+DA0A	"10 - Stage: Connect web"
00180317: scan+10057	"11 - Stage: Search for mirrors..."
00180530: scan+10270	"12 - Stage: ERROR Mirror not found. Go to offline mode..."
001835FA: scan+1333A	"13 - Stage: Check cheat db"
001840AB: scan+13DEB	"14 - Stage: Get path list..."
00184803: scan+14543	"15 - Stage: Scan registry..."
00184D33: scan+14A73	"16 - Stage: Get OS info..."
001851D8: scan+14F18	"17 - Stage: Try to open process"
002DE800: sub_2DE050+7B0	"18 - Stage: Scan memory 1/3...Progress: 0% (x B of y MB)"
0018638B: scan+160CB	"19 - Stage: Scan modules 1/3"
00186A22: scan+16762	"20 - Stage: Scan modules 2/3"
0018A8C5: scan+1A605	"21 - Stage: Scan modules 3/3"
0018BE29: scan+1BB69	"22 - Stage: Read game server"
0018C963: scan+1C6A3	"23 - Stage: Detect game multiple proc"
0018E965: scan+1E6A5	"24 - Stage: Retrive childs..."
001904CC: scan+2020C	"25 - Stage: Scan for injectors...Progress: 0%" loop
0018EF09: scan+1EC49	"25 - Stage: Scan for injectors...Progress: 0%..."
00192263: scan+21FA3	"26 - Stage: Scan network..."
00192BCB: scan+2290B	"27 - Stage: Scan cpu"
00196AF9: scan+26839	"28 - Stage: Scan modules"
001980C4: scan+27E04	"29 - Stage: Scan virtual mem"
0019C01F: scan+2BD5F	"30 - Stage: Scan executables"
0019C3C0: scan+2C100	"31 - Stage: Scan patches"
0019CCED: scan+2CA2D	"32 - Stage: Read user game info"
0019D5A1: scan+2D2E1	"33 - Stage: Retrive game server..."
0019D95C: scan+2D69C	"34 - Stage: Try to debugging 1/3..."
0019DC91: scan+2D9D1	"34.1 Stage: Try to debugging 1/3 Search hidden cheats..."
0019FE6E: scan+2FBAE	"34.2 Stage: Try to debugging 3/3..."
001A1933: scan+31673	"34.3 Stage: Try to debugging 3/3 Cheat list dumping..."
001A23CB: scan+3210B	"35 - Stage: Scan histories"
001A35DD: scan+3331D	"36 Stage: Scan fastrun processes.."
001A5646: scan+35386	"37 - Stage: Cheat [0x0003AF] Scanning. Please wait...Processed cheats 0%" loop
001A65F4: scan+36334	"38.1 - Stage: Write scan 1-5"
001A928E: scan+38FCE	"38.1.1 - Stage: End preparing"
001A9D3B: scan+39A7B	"38.2 - Stage: Write scan 2-5"
001AAD9E: scan+3AADE	"38.3 - Stage: Write scan 3-5"
001AAFF9: scan+3AD39	"38.4 - Stage: Write scan 4-5"
001AB4A0: scan+3B1E0	"38.5 - Stage: Write scan 5-5"
001ABB77: scan+3B8B7	"39 - Stage: Log actions"
001ABCE5: scan+3BA25	"40 - Stage: Write access info"
001AC0C2: scan+3BE02	"41.1 Stage: Prepare data 1 / 3"

הוספת חוק ל-Firewall

על מנת שהתוכנה תוכל לבצע תקשורת ברשת היא מוסיפה חוק ל-Firewall באמצעות שימוש ב-COM Objects. הפונקציה sub_324640 משתמשת ב-INetFwPolicy2, INetFwRule:

```
app_name_ = SysAllocString(app_name);
hr = CoCreateInstance(
       &rclsid_HNetCfg_FwPolicy2,
       NULL,
       CLSCTX_INPROC_SERVER,
       &riid_INetFwPolicy2,
       (LPVOID *)&ppv);
if ( hr >= S_OK )
{
  get_Rules = (int (__stdcall *)(LPVOID, INetFwRules **))ppv->lpVtbl->get_Rules;
  hr = get_Rules(ppv, &rules);
  if ( hr >= S_OK )
  {
    rule = 0;
    Item = rules->lpVtbl->Item;
    hr = Item(rules, name, &rule);
    if ( hr >= S_OK && rule )
    {
      put_ApplicationName = rule->lpVtbl->put_ApplicationName;
      hr = put_ApplicationName(rule, app_name_);
      ((void (__thiscall *)(INetFwRule *, INetFwRule *))rule->lpVtbl->Release)(rule, rule);
      v32 = hr >= S_OK;
      v54 = hr >= S_OK;
    }
    else
    {
      hr = CoCreateInstance(
             &rclsid_HNetCfg_FwRule,
             NULL,
             CLSCTX_INPROC_SERVER,
             &riid_INetFwRule,
             (LPVOID *)&ppv_);
      if ( hr >= S_OK )
      {
        v22 = (void (__stdcall *)(LPVOID, BSTR))ppv_->lpVtbl->put_Name;
        v22(ppv_, name);
        v21 = (void (__stdcall *)(LPVOID, BSTR))ppv_->lpVtbl->put_ApplicationName;
        v21(ppv_, app_name_);
        v20 = (void (__stdcall *)(LPVOID, int))ppv_->lpVtbl->put_Action;
        v20(ppv_, 1);
        v19 = (void (__stdcall *)(LPVOID, unsigned int))ppv_->lpVtbl->put_Enabled;
        v19(ppv_, 0xFFFFFFFF);
        v18 = (void (__stdcall *)(LPVOID, int))ppv_->lpVtbl->put_Protocol;
        v18(ppv_, 6);
        v17 = (void (__stdcall *)(LPVOID, int))ppv_->lpVtbl->put_Direction;
        v17(ppv_, 2);
        v16 = (void (__stdcall *)(LPVOID, int))ppv_->lpVtbl->put_Profiles;
        v16(ppv_, 0x7FFFFFFF);
        Add = rules->lpVtbl->Add;
        hr = Add(rules, ppv_);
        v31 = hr >= S_OK;
        v54 = hr >= S_OK;
```

נעזרתי בפלאגין ComIDA על מנת לקבל פלט דיקומפייל אצתטי:



חישוב מזהה מחשב (HWID)

הפונקציה calc_hwid שנמצאת ב-0x3095A0 (לפי RVA) מבצעת את שאילתות ה-WMI הבאות:

```
SELECT UniqueId, ProcessorId, Name FROM Win32_Processor
SELECT Manufacturer, SerialNumber FROM Win32_BIOS
SELECT Product, Model, Manufacturer, SerialNumber FROM Win32_BaseBoard
```

לאחר מכן היא משרשרת את התוצאות לפי קבוצות של מחלקות (Classes). לדוגמה, עבור המחלקה Win32_Processor מתקבלת המחרוזת:

```
"0000000000000000Intel(R) Core(TM) 7 160UL"
```

זה שרשור של הערכים: UniqueId (שבדר"כ הוא ריק, נבדק גם מחוץ ל-VM) בתוספת הערך של ProcessorId (המחרוזת: 0000000000000000) ובתוספת הערך של Name (המחרוזת: Intel(R) Core(TM) 7 160UL). על כל מחרוזת שמתקבלת מקבוצת שדות כזו, הפונקציה מחשבת גיבוב MD5. מתוך כל תוצאת MD5 נלקחים 12 הבתים הראשונים, ולאחר מכן כל התוצאות משורשרות יחד למחרוזת אחת, כאשר התו - משמש כמפריד בין הקבוצות.

```
MD5(UniqueId,ProcessorId,Name)[:12] -
MD5(Manufacturer,SerialNumber)[:12] -
MD5(Product,Model,Manufacturer,SerialNumber)[:12]
```

לתוכנה יש גם מנגנון זיהוי לבדיקת האם יש WMI spoofing ע"י בדיקה האם יש הוק על המתודה IWbemClassObject::Get הנמצאת ב-fastprox.dll.

חישוב מזהה HDD ID

תחילה מנסים לחשב את המזהה של ה-HDD באמצעות שימוש ב-IOCTL_STORAGE_QUERY_PROPERTY IOCTL (הערך 0x2D1400). המבנה STORAGE_PROPERTY_QUERY מאותחל כך:

- השדה PropertyId מאותחל ל-StorageDeviceProperty (הערך 0), ערך זה מציין ביצוע שאילתה ל-Device descriptor, שאמור להחזיר לנו מבנה מסוג STORAGE_DEVICE_DESCRIPTOR ב-באפר הפלט (lpOutBuffer) המסופק לפונקציה DeviceIoControl.
- השדה QueryType גם כן מאותחל לערך 0 המציין ביצוע שאילתה (בניגוד לערך 1 המציין "האם בקשה זאת נתמכת?").
- השדה AdditionalParameters שגם כן מאותחל לאפסים.

לאחר מכן מתבצעת שליפת המחרוזת של SerialNumber באמצעות השדה SerialNumberOffset המציין את ה-offset של SerialNumber ביחס לתחילתו של הבאפר:

```
hDevice = CreateFileW(lpFileName, 0xC0000000, 3u, 0, 3u, 0, 0);// "\\.\PhysicalDrive0"
if ( hDevice == 0xFFFFFFFF )
{
  sub_147580(a1, word_664F88);
  v47 |= 1u;
  sub_26C5B0(v35);
  v6 = v35;
  v26 = 0;
  return a1;
}
else
{
  memset(InBuffer, 0, 0xC);
  v5 = &v45;
  BytesReturned[4] = &v37;
  v43 = &v37;
  v37 = 0;
  v38 = 0;
  v39 = 0;
  sub_263FF0(0x400);
  LOBYTE(v26) = 7;
  BytesReturned[0] = 0;
  v34 = &v37;
  BytesReturned[3] = v38 - v37;
  nOutBufferSize = v38 - v37;
  BytesReturned[2] = v37;
  BytesReturned[1] = v37;
  lpOutBuffer = v37;
  if ( DeviceIoControl(hDevice, &loc_2D1400, InBuffer, 0xCu, v37, v38 - v37, BytesReturned, 0) )
  {
    v28 = v37;
    v27 = v37;
    v33 = v37;
    if ( v37->SerialNumberOffset )
    {
      v25 = v37;
      v24 = v37;
      lpMultiByteStr = v37 + v33->SerialNumberOffset;
      cchWideChar = MultiByteToWideChar(0, 0, lpMultiByteStr, 0xFFFFFFFF, 0, 0);
```

במידה ושיטת ה-IOCTL נכשלת (למשל בריצה ב-VM מעל Hyper-V השדה SerialNumberOffset חוזר ריק), החישוב מזהה של ה-HDD יתבצע באמצעות שאילתת WMI:

```
SELECT Model, SerialNumber FROM Win32_DiskDrive
```

בדומה לתוצאה הסופית של חישוב ה-HWID, גם כאן מתבצע חישוב MD5 על המזהה HDD, ולבסוף מחזירים את ה-8 בתים הראשונים של ה-MD5.

מי שרוצה להרחיב עוד על HWID ושיטות הסוואה מוזמן לקרוא את החומרים הבאים:

- https://www.unknowncheats.me/forum/anti-cheat-bypass/333662-methods-retrieving-unique-identifiers-hwids-pc.html
- https://www.unknowncheats.me/forum/apex-legends/751041-hwid-spoofer-technical-architecture.html
- https://www.unknowncheats.me/forum/anti-cheat-bypass/323218-smbios-kernel-hook-source.html
- https://www.unknowncheats.me/forum/anti-cheat-bypass/287926-kernelmode-smbios-hardware-id-spoofing.html
- https://www.unknowncheats.me/forum/anti-cheat-bypass/560809-firmwaretablehandler.html
- https://www.unknowncheats.me/forum/anti-cheat-bypass/310941-HDD-serial-spoofer-hooking.html

תקשורת ראשונית עם השרת

התוכנה מנסה לתקשר עם הדומיין fungun.top ואם הדומיין כרגע לא זמין אז היא תנסה עוד כמה דומיינים כמו fungun.net במידה וגם הם לא זמינים אז התקשורת תתבצע מול כתובות IP. לתוכנה יש תמיכה בעדכון גרסה. הבקשה הראשונה שנשלחת מבוססת על המידע הבא:

```
{
"base": "00000000",
"HDD_id": "ffffffff",
"hw_id": "ffffffffffffffffffffffff",
"parent": "explorer.exe",
"scan": "63fd0706",
"scanner": "2.74",
"spoof": 0,
"xpbuild": 0
}
```

חלק מהשדות כבר דיי ברורים ולא צריך להסביר.

- השדה hw_id זה גרסה מקוצרת של ה-HWID שראינו למעלה.
- השדה scan מכיל את תוצאת החישוב (ליתר דיוק - רק את 8 התווים הראשונים) MD5 של הקובץ מהדיסק.
- השדה spoof יכיל את הערך 1 במידה וזוהה WMI spoofing.
- השדה xpbuild אני מניח, מציין האם נעשה שימוש בגרסה של Windows XP.

אך לפני שהבקשה נשלחת, היא מוצפנת באמצעות מפתח RSA ציבורי המוטמע בתוך התוכנה:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuJnBZglM3aQVOb89gEhk
tJqvd64hWQLv6aGAeQiI2OWFDgSAVoYEEciNpSa6eFnQB6C3oeZFVFuthRhkS3hH
4wu4cX8akCBVMGzARUJ2Ra9Qrqw5PNXv9PGXgxl73tz9QpoTFMRPS++DExnL9z7W
+PRiJrEmfT1ok3IXSeH7MLaAllyIwu3AK+UpE4bCtHlFU8LcwyznfuizLwge0zcP
oQQKhRw14gr+drW2E4LKvdk1B0ueRxGMxcv93L147CEytpw2gFXnucxThrlH3yFk
HEYtnDWFbkZ+tzQHCuwCywCc0mJXfb+DmAOktc7dkDZLwC+3yHYYcr1dfseMHqXI
sQIDAQAB
-----END PUBLIC KEY-----
```

דוגמה מלאה של הבקשת POST:

```
POST /ska/ecd.php?method=update HTTP/1.1
Accept-Encoding: deflate, gzip
Host: fungun.top
Accept: application/json
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.5) Gecko/20100101 Firefox/45.0
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Content-Length: 403

data=rsnWJUgIniBImmOWOMPlDzhiuj7FAtF9q8xl6PCYP%25252B76SvcR87wN9yL50u1%25252F2
sS%25252F3UjYHCPRoUwUNORV0RyNFm26dfZVDnBBeCibSMjvO7aQSQJm0HWVkNbDsVUWoko
Jtuyw45CNr2EQm188XxIIoeij0ZbqbRtP7u5RF4kgJPOqAqeLS91Pc%25252BaqC7qzsS6tdBqDwu1w
VkwS7%25252FADGhJQV6NGOPg7JrRKZpQlTzBZefz6q8Qz3h1fb3VHXASAOYYH7e9VlDjT1GOHZA
7CgovBSXNawIIE1Rt%25252FfG8CFQpSLaCGQPYyJi7Ymrqe8Di%25252F2aMm29ciymXyEqaeYjTX
Kea0Hg%25253D%25253D
```

במידה והבקשה תקינה (וגם אם לא שלחנו יותר מדי בקשות בזמן קצר), נקבל תשובה כזאת:

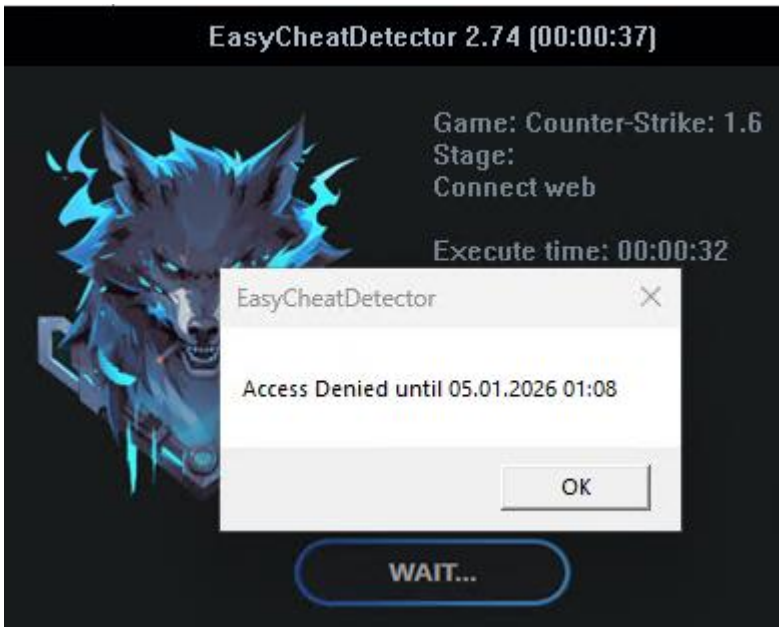
```
{
"success":true,
"unixtime":0,
"scanner1":"https:\/\/github.com\/UnrealKaraulov\/EasyCheatDetector\/releases\/download\/
2.82\/EasyCheatDetector.zip",
"scanner2":"https:\/\/fungun.net\/ska\/ecd.php?method=download",
"base":"Base64(AES-256(cheats_db))",
"scanner3":"https:\/\/fungun.net\/ska\/ecd.php?method=download&exe=1",
"scan_hash":"d5bb165ea0e591aac04294d0c98cada9"
}
```

אחרי קצת מחקר סטטי ודינמי גיליתי שאת המידע הנמצא בשדה base, ניתן לפענח באמצעות מפתח ידוע המוטבע בתוך התוכנה. כאן ניתן למצוא דוגמה לפלט המפוענח - נראה שהוא מכיל חתימות לזיהוי ציטים.

במידה ושלחנו בקשה לא תקינה או יותר מדי בקשות - נקבל תשובה כזאת:

```
{"success": false, "error": "Access Denied until 05.01.2026 01:08"}
```

והתוכנה תציג את זה גם למשתמש:

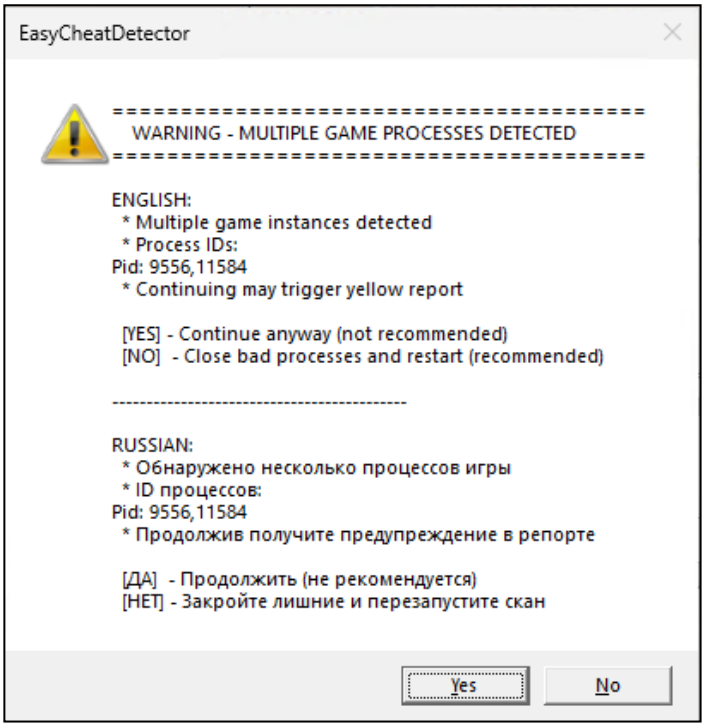


מציאת תהליך המשחק

במידה ורצים מעל Wine, החיפוש של תהליך המשחק יתבצע באמצעות CreateToolhelp32Snapshot, Process32First, Process32Next. במידה ולא רצים מעל Wine, אז החיפוש נעשה באמצעות הפונקציה NtQuerySystemInformation ע"י שימוש ב-SystemExtendedProcessInformation - לא זוכר שנתקלתי בזה בעבר, זאת אינה שיטה נפוצה. התוכנה תחפש אחרי התהליכים הבאים:

```
hl.exe
cstrike.exe
czero.exe
cs.exe
cs_new.exe
cstrike_new.exe
xash.exe
game_start.exe
css.exe
rustclient.exe
cs2.exe
hl2.exe
csgo.exe
left4dead2.exe
```

במידה ונמצאים כמה תהליכים מהרשימה פתוחים ברקע, התוכנה תציג את האזהרה הבאה:



במידה ונמצא התהליך של המשחק (במקרה שלנו - **hl.exe**), לאחר מכן מתבצע חיפוש האם אחד מה-DLLs הבאים טעונים באותו תהליך של המשחק:

```
hw.dll
v8.dll
sw.dll
```

ולבסוף חיפוש תיקיית valve תחת התיקייה הראשית שממנה רץ התהליך של המשחק.

שם השחקן

מציאת שם השחקן מתבצעת באמצעות 3 דרכים שונות:

- שליפה מהנתיב הבא ברגסטרי:

```
HKCU\Software\Valve\Steam\LastGameNameUsed
```

```
update_window_text(v5488, v5486, 0xF);        // "32 - Stage: Read user game info"
LOBYTE(v10587) = 0xEC;
wrap_free_0(v1183);
v10587 = 0x3B9;
wrap_free_0(v1182);
v5484 = sub_209650(v1179);
v5483 = v5484;
v10587 = 0x7EE;
v5482 = sub_2D78B0(v1180, v5484, 0);
v5481 = v5482;
...
  wrap_free_0(v1177);
  v10587 = 0x3B9;
  sub_4F860(v1176);
  v9990 = 0;
  sub_174A80(&v9990);
  v10587 = 0x7F4;
  v5472 = sub_209730(v1175);
  v5471 = v5472;
  LOBYTE(v10587) = 0xF5;
  v10538 = *wrap_RegOpenKeyExW(&v9990, v2923, HKEY_CURRENT_USER, v5472, &g_samDesired);// "Software\Valve\Steam"
  LOBYTE(v10587) = 0xF4;
  wrap_free_0(v1175);
  if ( sub_13BC80(&v10538) )
  {
    something_with_string_3(v2442, &aLastgamenameus);
    LOBYTE(v10587) = 0xF6;
    wrap_RegGetValueW_sz(&v9990, v2549, v2442);// "LastGameNameUsed"
```

- שליפה מתוך הקובץ config.cfg.
- שליפה מהזיכרון של המשחק ע"י שימוש באופסטים ידועים. יש לא מעט מידע על זה ברשת.

מזהה השחקן (steamid)

נשלף באמצעות המידע שנמצא ברגסטרי במפתח הבא:

```
HKCU\Software\Valve\Steam\ActiveProcess\ActiveUser
```

חיבורי רשת

התוכנה משתמשת ב-2 פונקציות דיי דומות על מנת לאסוף את החיבורי רשת מהתהליך של המשחק.
הפונקציה הראשונה enumerate_tcp_connections_for_pid נמצאת ב-0x2D2A50 (לפי RVA).
הפונקציה השנייה enumerate_udp_connections_for_pid נמצאת ב-0x2D2EB0 (לפי RVA).

הפונקציות GetExtendedTcpTable, GetExtendedUdpTable מאפשרות לקרוא את טבלת חיבורי ה-TCP\UDP הפעילים, כולל שיוך ל-PID של תהליך מסוים. הפרמטר המעניין הוא TableClass, הפונקציות מעבירת את הערך TCP_TABLE_OWNER_PID_CONNECTIONS ו-UDP_TABLE_OWNER_PID בהתאמה:

```
pTcpTable = 0;
ExtendedTcpTable = 0;
pdwSize = 0x1C;
if ( pid )
{
  pTcpTable = wrap_alloc(pdwSize);
  if ( pTcpTable )
  {
    if ( GetExtendedTcpTable(pTcpTable, &pdwSize, 0, 2u, TCP_TABLE_OWNER_PID_CONNECTIONS, 0) != 0x7A
      || (wrap_free_6(pTcpTable), (pTcpTable = wrap_alloc(pdwSize)) != 0) )
    {
      ExtendedTcpTable = GetExtendedTcpTable(pTcpTable, &pdwSize, 0, AF_INET, TCP_TABLE_OWNER_PID_CONNECTIONS, 0);
      if ( !ExtendedTcpTable )
      {
        for ( i = 0; i < pTcpTable->dwNumEntries; ++i )
        {
          if ( pTcpTable->table[i].dwOwningPid == pid )
          {
            pStringBuf = 0;
            wrap_memset(v2, 0, 0x800u);
            LOWORD(v14) = 2;
            pAddr[0] = pTcpTable->table[i].dwRemoteAddr;
            HIWORD(v14) = pTcpTable->table[i].dwRemotePort;
            InetNtopW(2, pAddr, &pStringBuf, 0x401u);
            sub_147430(v4);
```

איסוף מידע על התהליכים

התוכנה אוספת מידע על כל התהליכים במערכת באותה דרך שבה היא ניסתה למצוא את התהליך של המשחק. כפי שנראה בהמשך בחלק של הדו"ח המלא, זה המידע שנאסף עבור תהליך:

- הגודל של הקובץ על הדיסק.
- הנתיב המלא שלו.
- כמה מופעים יש לאותו תהליך (רלוונטי לתהליכי svchost.exe המייצגים Services במערכת).

התוכנה משתמשת בשתי שיטות לשליפת הנתיב המלא של כל תהליך במערכת.

- הראשונה מבוססת על GetModuleFileNameEx, שמקבלת HANDLE של תהליך, וגם HMODULE של מודול מסוים, במקרה הזה התוכנה מעבירה NULL, כדי לחלץ את הנתיב של התהליך עצמו. מאחורי הקלעים הפונקציה GetModuleFileNameEx משתמשת בפונקציה NtQueryInformationProcess עם הפרמטר ProcessImageFileNameWin32 כדי לשלוף את הנתיב של התהליך.

- השיטה השנייה מבוססת על הפונקציה GetProcessImageFileName, שגם משתמשת בפונקציה NtQueryInformationProcess אך היא מעבירה לה את הפרמטר ProcessImageFileName.

קצת internals למי שמעוניין: בקריאה עם הפרמטר ProcessImageFileName הנתיב המלא נשלף מתוך השדה SeAuditProcessCreationInfo.ImageFileName הנמצא במבנה EPROCESS. הנתיב המלא חוזר בפורמט NT Path, דוגמה:

```
\Device\HarddiskVolume4\Windows\System32\smss.exe
```

בקריאה לפונקציה עם הפרמטר ProcessImageFileNameWin32, הנתיב המלא נשלף דרך השדה ImageFilePointer במבנה EPROCESS. השדה ImageFilePointer הוא למעשה מצביע למבנה נתונים FILE_OBJECT של התהליך והוא מייצג את הקובץ, איתו ניתן לשלוף את הנתיב המלא באמצעות הפונקציה IoQueryFileDosDeviceName. הנתיב המלא חוזר בפורמט Win32 Path, דוגמה:

```
C:\Windows\System32\smss.exe
```

ההבדל המרכזי בין השניים הוא שמקור המידע שונה.

לאחר שליפת הנתיבים המלאים בשתי השיטות, התוכנה מבצעת בדיקה של שמות הקבצים בלבד, כדי לזהות אי התאמה שעשויות להעיד על מניפולציה שנעשתה ע"י תהליך מסוים, כנראה, על מנת לרמות ועל הדרך להסתיר עקבות. במידה וזוהתה אי התאמה, המידע הזה יישמר כן גם בדו"ח המלא, יירשם תחת קטגוריית "injectors", דוגמה:

```
"injectors": {
  "1": {
    "path": "C:\\Windows\\System32\\calc.exe",
    "old_path": "c:\\windows\\system32\\smss.exe"
  }
}
```

השדה "path" מכיל את הנתיב שהפונקציה GetModuleFileNameEx החזירה. והשדה "old_path" מכיל את הנתיב (אחרי שעבר המרה ל-Win32 Path ע"י שימוש בפונקציה QueryDosDevice) שהפונקציה GetProcessImageFileName החזירה.

איסוף מידע על הדרייברים

את המידע על הדרייברים במערכת התוכנה אוספת באמצעות מעבר על המפתח הבא ברגסטרי:

```
HKLM\SYSTEM\CurrentControlSet\Services
```

על מנת להבדיל בין קרנל דרייבר לבין סתם Service התוכנה מוודא שאכן הערך של Type הוא 1.

```
LODWORD(v6) = sub_5467A0(__PAIR64__(v5, v61), 0x989680LL);
v114 = v6 - 0x2B6109100LL;
something_with_string_3(lpValue_1, L"Type");
LOBYTE(v93) = 0xE;
wrap_RegGetValueW_dword(&hKey_3, lstatus_3_and_value, lpValue_1);
LOBYTE(v93) = 0xF;
wrap_free_2(lpValue_1);
v31 = lpValue_1;
LOBYTE(v93) = 0xD;
something_with_string_3(lpValue_2, L"ImagePath");
LOBYTE(v93) = 0x12;
wrap_RegGetValueW_sz(&hKey_3, lstatus_4, lpValue_2);
LOBYTE(v93) = 0x15;
wrap_free_2(lpValue_2);
v52 = lpValue_2;
LOBYTE(v93) = 0x14;
if ( sub_334C50(lstatus_3_and_value) )
{
  if ( sub_249320(lstatus_4) )
  {
    v51 = 1;
    Type_value = sub_249250(lstatus_3_and_value);
    v87 = Type_value;
    if ( *Type_value == 1 )           // is a Kernel driver?
    {
      v50 = 1;
      v86 = sub_2493E0(lstatus_4);
      v85 = v86;
      if ( *(v86 + 0x10) )
      {
```

כפי שנראה בהמשך בחלק של הדו"ח המלא, המידע שנאסף עבור דרייבר:

- שם המפתח ברגסטרי.
- הגודל של הקובץ על הדיסק.
- הנתיב המלא שלו.
- תיאור (Description) של הדרייבר (לדוגמה: "TCP/IP Protocol Driver" המייצג את הדרייבר tcpip).

למה זה בכלל מעניין לדעת איזה דרייברים קיימים במערכת? תוקפים משתמשים לעיתים בדרייברים זדוניים כדי לעקוף מנגנוני הגנה ולרמות במשחקים. מאחר שדרייברים רצים בקרנל הם בעלי הרשאות גבוהות מאוד, דבר זה מקשה משמעותית על מנגנוני זיהוי צ'יטים. זה תמיד היה משחק של חתול ועכבר ומאוד קשה לנצח תוקף שרץ בקרנל.

איסוף מידע על המערכת

שם מחשב

קבלת מידע על שם המחשב מתבצעת באמצעות שאילתת WMI הבאה:

```
SELECT Name FROM Win32_ComputerSystem
```

מערכת הפעלה

קבלת מידע על מערכת ההפעלה מתבצעת באמצעות 2 השאילתות WMI הבאות:

```
SELECT Version FROM Win32_OperatingSystem
SELECT Name FROM Win32_OperatingSystem
```

במידה ומסיבה כלשהי השליפה של המידע באמצעות WMI נכשלה, תתבצע שליפה מהמפתח הבא ברגסטרי:

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName
```

היסטוריית הורדות

התוכנה מחפשת אחרי דפדפנים מסוימים במערכת על מנת לבדוק (בהמשך) את ההיסטוריה של ההורדות.

החיפוש של אותם דפדפנים מתבצע ע"י בדיקה האם הקבצים הרלוונטים שלהם נמצאים בנתיבים הבאים:

```
"%AppData%\Local\360extremebrowser"
"%AppData%\Local\AVG"
"%AppData%\Local\BraveSoftware"
"%AppData%\Local\CCleaner Browser"
"%AppData%\Local\Google"
"%AppData%\Local\Microsoft"
"%AppData%\Local\Vivaldi"
"%AppData%\Local\Yandex"
"%AppData%\Roaming\Mozilla"
"%AppData%\Roaming\Opera Software"
"%AppData%\Roaming\Waterfox"
"%AppData%\Roaming\dolphin_anty"
```

זיהוי צ'יטים

לתוכנה יש לא מעט שיטות לתפוס צ'יטים.

חתימות צ'יטים

באופן דומה למה שראינו בחלק של "תקשורת ראשונית עם השרת", במידה והתוכנה לא מצליחה לגשת לשרתים שלה, היא מפענחת את החתימות מתוך סקשן ה-Resource באופן דיי זהה.

הפונקציה scan_for_cheats שנמצאת ב-0x14C610 (לפי RVA) אחראית על זיהוי של צ'יטים מבוססים חתימות. הפונקציה עוברת על כל רשומה (פורמט JSON) שנמצאת במאגר החתימות, כל רשומה מתארת זיהוי של צ'יט מסוים. חלק מהשדות בכל רשומה מקודדים ע"י Base64 בנוסף ל-XOR, הפונקציה תפענח אותן לפני השימוש. המפתח הבא משמש לפענוח השדות arg1, arg2, arg3, arg4 באמצעות XOR:

```
std::string g_PrivateKey1
```

דוגמה:

```
"arg1":
"XVpLXl8eGwIFDx5FFXE0HgU"

output:
"../demoplayer.dll"

"arg2":
"QEdVDQxKRkU"

output:
"33176927"

"arg3" input:
"HgEITlMbFREC"

output:
"multihack"
```

על מנת לפענח את השדה command, משתמשים באותו מפתח + משרשרים אליו את הערך של השדה cheat_id, לדוגמה:

```
std::string g_PrivateKey11627

"command":
"MBwBWVE0FR8MKg5SLz4jGg"

output:
"CheckGameDirHash"
```

השדה command מציין את שיטת הבדיקה למציאת הצ'יט, יש 12 שיטות שונות:

1. CheckGameDirHash - לוקחים את אותו קובץ מהתיקייה הראשית של המשחק (לרוב זה DLL), מחשבים את ה-MD5 של אותו קובץ, לוקחים רק את ה-8 תווים הראשונים של התוצאה ואז משווים אל מול תוצאה ידועה מראש. בדוגמה למעלה, ההשוואה מתבצעת אל מול הערך שנמצא ב-arg2, כלומר עם הערך 33176927. יש רשומות שלא מכילות MD5 לבדיקה - אלא רק שם קובץ, במקרים כאלה במידה והקובץ קיים זה מספיק בשביל להפליל.
2. CheckGameDirHash2 - נראה שיש רשומה אחת כזאת בלבד. השדה is_test מכיל את הערך 1. השדה מכיל שם של קובץ, והשדה arg2 נראה שמכיל תוצאת חישוב של MD5. נראה ששיטה זו לא באמת ממומשת בקוד ולכן היא אינה רלוונטית.
3. SearchCustomCheats - אין הרבה רשומות שמכילות את השיטה הזאת, נראה שהיא גם לא באמת ממומשת בקוד. רק השדה info מכיל מידע, שאר השדות ריקים. השדה is_test מכיל את הערך 1. בחלק מהרשומות השדה is_future גם כן מכיל את הערך 1.
4. SearchDragonFireMem - חיפוש מחרוזות בתהליך של המשחק. מספיקה מחרוזת אחת שנתפסה על מנת להפליל.
5. SearchFoxPing - עוברת על החיבורי רשת (של המשחק) על פי תבנית Regex ומחפשת האם יש תקשורת לדומיין מסוים.
6. SearchGameDirHash - השדה arg1 יכול להכיל: (1) נתיב המציין את התיקייה בתוך התיקייה של המשחק. או, (2) יכיל את התו "/" - המציין מעבר על כל התיקיות בתיקייה הראשית של המשחק. השדה arg2 מכיל את שם הקובץ אותו מחפשים על פי תבנית Regex. במידה והקובץ קיים זה מספיק בשביל להפליל, אין פה השוואה מול MD5. שיטה זאת קצת מזכירה את שיטה מספר 1.
7. SearchHistoryDirHash - עוברת על ההיסטוריה של ההורדות בדפדפן ומחפשת לפי תבנית Regex האם סיומת של קובץ מסוים ירד מאתר מסוים (למשל unknowncheats.me מככב שם בחתימות).
8. SearchMemOrder - חיפוש מחרוזות לפי תבנית Regex לפי סדר (כלומר המחרוזות צריכות להיות אחת אחרי השנייה) בתהליך של המשחק.
9. SearchProcessRegex - מציאת תהליך במערכת לפי תבנית של Regex.
10. SearchRegDirHash - חיפוש ברגסטרי על תבנית Regex תחת המפתחות הבאים:

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\FeatureUsage\AppSwitchedHKLM
\SYSTEM\ControlSet001\Services\bam\State\UserSettings\<SID>
```

11. SearchRootDirHash - על פי הקוד לא נראה ששיטה זאת ממומשת. על פי הרשומות, נראה שהשדה arg1 יכול לציין את התיקייה שבה יש לחפש את שם הקובץ המאוחסן בשדה arg2.
12. SearchUserDirHash - חיפוש אחרי קובץ בתיקייה מסוימת (למשל Desktop, Downloads) בתיקייה הראשית של המשתמש. דוגמה: C:\Users\[USER]\Downloads\wardgos_clear.exe. ישנה תמיכה גם בתבנית Regex לפי שם הקובץ.

במידה והקובץ קיים זה מספיק בשביל להפליל, אין פה השוואה מול MD5. ניתן לעקוף את כל הבדיקות האלה בקלות ע"י שינוי מחרוזות של שמות הקבצים, של המחרוזות עצמם שנמצאות בתוך צ'יט מסוים.

מציאת חלונות חשודים

התוכנה תבצע מעבר על חלונות המשחק באמצעות הפונקציה EnumWindows ולאחר מכן היא תבצע 2 בדיקות:

- האם המשחק נמצב במצב "Run in a window" (כלומר לא "Full Screen").
- האם אחד החלונות נמצא במצב "נראה" - שימוש בפונקציה IsWindowVisible.

```
MACRO_BOOL __thiscall FindConsoleWindowEnumProc(void *this, HWND hWnd, LPARAM GamePid)
{
  DWORD dwProcessId; // [esp+0h] [ebp-4h] BYREF

  dwProcessId = this;
  GetWindowThreadProcessId(hWnd, &dwProcessId);
  if ( dwProcessId == GamePid && IsConsoleWindowClass(hWnd) && !g_hWnd )
    g_hWnd = hWnd;
  return TRUE;
}
```

במידה ונמצאה התאמה לאחת הבדיקות - המידע נרשם בדו"ח תחת המפתח "hacknames" עם התיאור "HACK WINDOW FOUND".

מציאת DLLs חשודים בתוך תהליך המשחק

כחלק ממערך ההגנה שלה, התוכנה מפעילה מנגנון ייעודי לזיהוי DLLs זדוניים בתוך המשחק המנסים לתפוס טרמפ על פונקצית הציור הרשמית כדי להציג מידע אסור על המסך (כמו ESP Overlay). על מנת לבצע זאת, התוכנה מבצעת Attach למשחק כ-Debugger ע"י שימוש בפונקציה DebugActiveProcess ושמה Hardware breakpoint בפונקציה HUD_Frame השייכת ל-client.dll, פונקציה זאת רצה בכל Frame. הגדרת ה-Hardware breakpoint מתבצעת באמצעות שימוש בפונקציה SetThreadContext. ברגע ש-Thread כלשהו קורא לפונקציה המנוטרת, התוכנה תקבל Debug Event, תשלוף את כתובת החזרה מראש המחסנית, היא תבדוק לאיזה DLL שייכת הכתובת כדי לבדוק האם הכתובת הזו נמצאת בתוך רשימת ה-DLLs המוחרגים:

```
hw.dll
v8.dll
engine.dll
client_save.dll
hwpatcher.dll
view_demo_helper.dll
vstdlibnewrevv.dll
libavformat-57.dll
```

```
nitro_api2.dll
steamptc.dll
gtlib.asi
steam_api.dll
```

אם הכתובת שייכת לאחד מהם, מתעלמים. במידה ולא, המידע על אותו DLL יישמר בדו"ח תחת המפתח "hack_modules".

מכיוון שהבדיקות כאן מבוססות על מידע שהגיע מה-PEB וגם מסריקת הזיכרון ע"י שימוש בפונקציה NtQueryVirtualMemory (חיפוש אחרי אזורי זיכרון מסוג MEM_IMAGE והצלבה מול חתימות MZ ו-PE) יש כמה דרכים לשבור את הזיהוי של התוכנה:

- ניתן להסתיר את הרפרנס ל-DLL בתוך ה-PEB ע"י שימוש בטכניקת Unlinking.
- השחתת חתימות ה-MZ ו-PE.
- הזרקת Shellcode - למשל שימוש בטכניקת Reflective DLL Injection - אבל כפי שתראו בהמשך התוכנה יודעת לזהות גם את זה.
- שינוי השם של ה-DLL לאחד השמות של ה-DLLs המוחרגים (הערה: זה דיי מחשיד שיש 2 DLLs עם אותו שם) או, שיטה יותר חשאית - פיצפוץ DLL מרשימת המוחרגים.

למידע נוסף על הזרקות קוד:

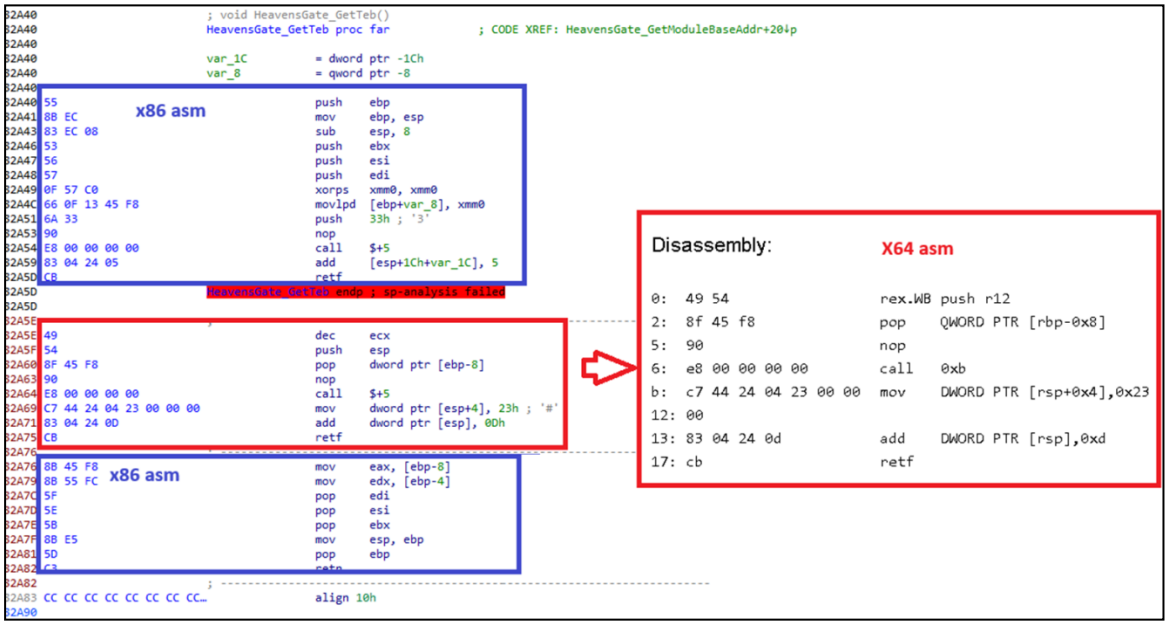
- https://www.digitalwhisper.co.il/files/Zines/0x98/DW152-2-CodeInjection.pdf
- https://www.cyberark.com/resources/threat-research-blog/masking-malicious-memory-artifacts-part-iii-bypassing-defensive-scanners
- https://www.huntandhackett.com/blog/concealed-code-execution-techniques-and-detection
- https://www.unknowncheats.me/forum/anti-cheat-bypass/321071-erasing-pe-header-idea.html

מציאת אזורי זיכרון חשודים - פתחו של גן עדן

התוכנה עוברת על הזיכרון של תהליך המשחק ומחפשת אחרי אזורי זיכרון המוגדרים RWX (קריאה, כתיבה, הרצת קוד) שאינם שייכים למודול מסוים (כלומר Type שווה ל MEM_PRIVATE). את המידע על הזיכרון של תהליך המשחק היא אוספת באמצעות שימוש בפונקציה NtQueryVirtualMemory. אממה, התוכנה קוראת לפונקציה בדרך חשאית משהו. התוכנה עושה שימוש בטכניקת Heaven's Gate.

למה דרך חשאית? כי גם אם אנחנו יודעים שהתוכנה עושה שימוש כלשהו בקריאת מערכת כלשהי - במידה ונשים Breakpoint על אותה קריאת מערכת - זאת לא בהכרח הקריאה שהיינו מעוניינים בה. מכיוון שמדובר בתהליך 32 ביט ה-Debugger ימקם את ה-Breakpoint בגרסה ה-32 ביט של Ntdll. בנוסף, לא כל

Debugger תומך (למשל שימוש ב-Breakpoint בגרסה 64 ביט של Ntdll, או למשל לבצע Single step במעבר בין 32 ל-64 ולהפך) בדיבאגינג של Wow64 - למשל x32dbg לא תומך בזה. בעוד ש-WinDbg כן.



בתמונה למעלה רואים דוגמה לפונקציה המשתמשת בטכניקת Heaven's Gate על מנת לשלוף את כתובת ה-TEB. נראה שאותו קטע קוד בצד ימין (מוקף בצבע אדום) זהה לקטע קוד הנמצא ב-GitHub.

למי שרוצה internals על Heaven's Gate:

- http://blog.rewolf.pl/blog/?p=102
- https://cloud.google.com/blog/topics/threat-intelligence/wow64-subsystem-internals-and-hooking-techniques/

לסיום, במידה ונמצאו קטעי זיכרון העומדים בתנאים שהוזכרו למעלה - המידע עליהם יישמר בדו"ח תחת המפתח "map_injections".

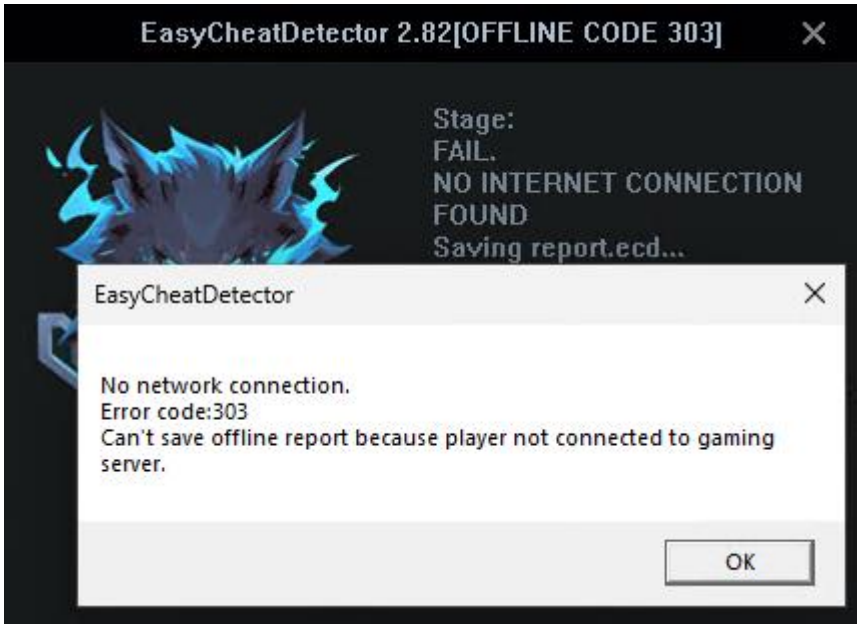
בדיקות נוספות

- בדיקה האם אחד ה-DLLs בתהליך מתחיל במחרוזת "Injected_" - בבדיקה הזאת יש לתוכנה באג, הבדיקה היא Case-sensitive והתוכנה משתמשת בשם של ה-DLL באותיות קטנות.
- מעבר על אזורי הזיכרון באמצעות הפונקציה NtQueryVirtualMemory, וחיפוש אחרי PE ו-MZ, לאחר מכן ניסיון שליפת שם המודול ע"י שימוש בפונקציה NtQueryVirtualMemory עם הפרמטר MemoryMappedFilenameInformation, במידה והיא לא מצליחה לשלוף את השם של המודול מסיבה כלשהי - דיווח.
- חיפוש אחרי הוקים ב opengl32.dll.

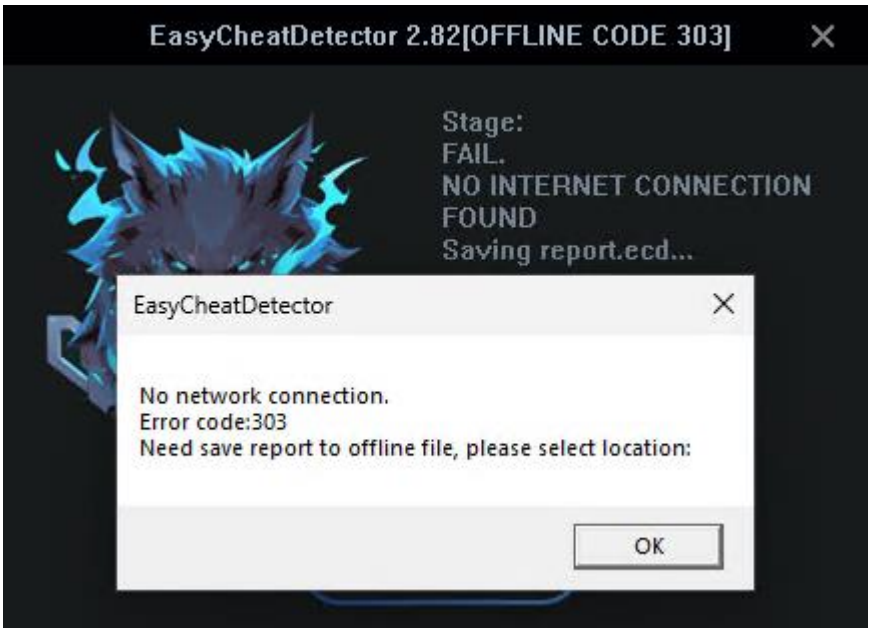
הדו"ח המלא

דו"ח מקומי

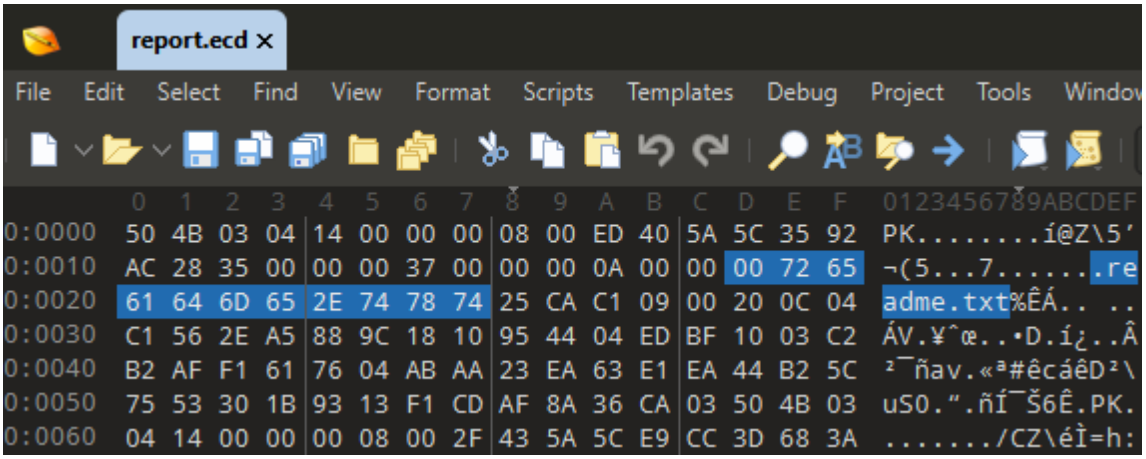
כזכור, במידה ואין אינטרנט אז הדו"ח של הסריקה אמור (כל עוד לא התוכנה לא זיהתה Sandbox או Debugger בבדיקות השונות) להישמר מקומית:



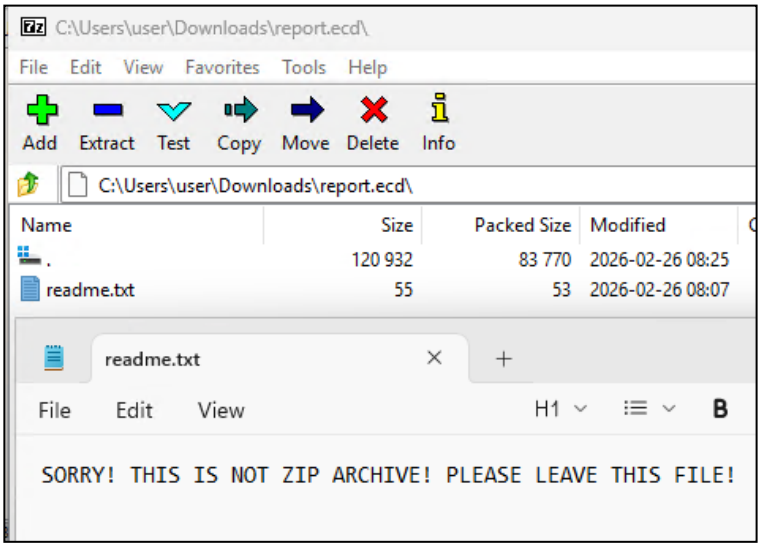
נראה שהתוכנה מסרבת לשמור את הדו"ח אם השחקן לא מחובר לסרבר מסוים. בעיה. מה נעשה? נתחבר לשרת מקומי (Localhost, עם בוטים) וככה עקפנו את הבדיקה הזאת:



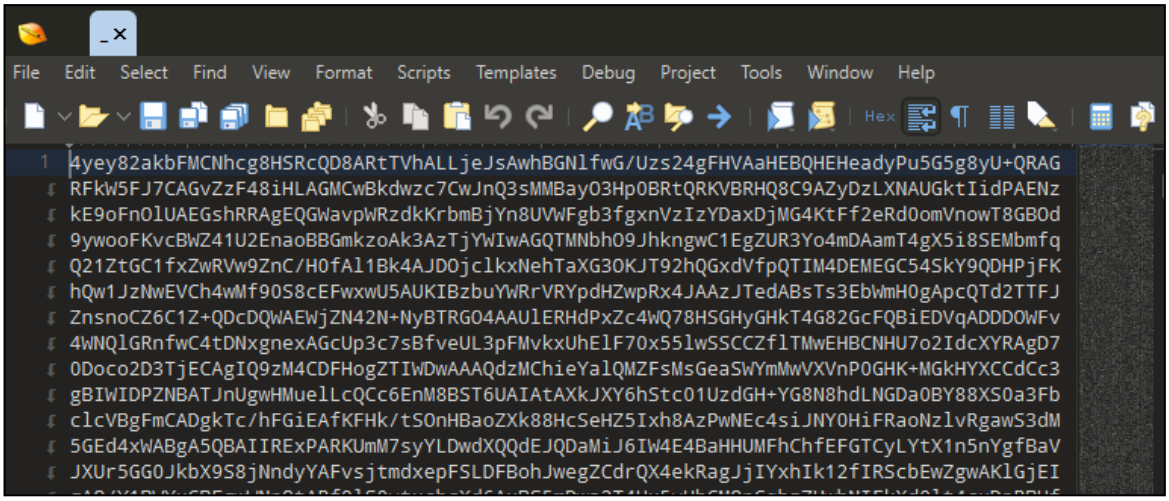
לאחר שנבחר מיקום, נראה קובץ בשם report עם סיומת ecd, מה זה הקובץ הזה?



נראה כמו קובץ Zip, בנוסף ניתן לראות שיש שם את המחרוזת readme.txt. נחלץ את מה שיש ב-Zip, נתחיל בבדיקה מה יש בקובץ readme.txt:



הקובץ הבא הוא ".":



נראה שהמידע מקודד באמצעות Base64, אך ניסיון לפענח את המידע לא מביא פלט קריא. לאחר קצת מחקר סטטי ודינמי הצלחתי לפענח את המידע וכתבתי סקריפט שמקבל קובץ report.ecd ומפענח את המידע, ניתן לראות כאן דוגמה לדו"ח המלא.

דו"ח מקוון

במידה ויש אינטרנט במכונה, הדו"ח של הסריקה אמור להישלח באופן מקוון לאותו שרת שראינו בחלק של "תקשורת ראשונית עם השרת", הבקשה נראית כך:

```
POST /ska/ecd.php?method=report HTTP/1.1
Accept:: */*
Cache-Control: no-cache
Connection: close
Content-Type: application/x-www-form-urlencoded
Host: fungun.top
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.5) Gecko/20100101 Firefox/45.0

data=...
```

המידע מוצפן/מקודד באותו אופן שהוא נשמר מקומית.

במידה והבקשה תקינה (וגם אם לא שלחנו יותר מדי בקשות בזמן קצר), נקבל תשובה כזאת:

```
{
"success": true,
"report_id": 123
}
```

סיכום

במאמר זה סקרנו יחד את תוכנת האנטי צ'יט Easy Cheat Detector וראינו שהמטרה שלה היא לא לחסום פעולות באופן אקטיבי, אלא רק לזהות ולדווח על צ'יטים. במהלך הניתוח ראינו איך התוכנה מנסה לגלות אם מריצים אותה ב-Debugger או בסביבה וירטואלית, ואיך היא אוספת מידע כמו מזהי חומרה (HWID), דרייברים, ורשימת תהליכים במערכת. בנוסף, החלק העיקרי והמעניין שסקרנו הוא השיטות שהתוכנה בודקת האם השחקן משתמש בצי'טים.

בסוף התהליך התוכנה אורזת ומצפינה את כל המידע הזה ושולחת אותו לשרת שלה. השורה התחתונה שראינו יחד היא שלמרות כל הבדיקות, עדיין אפשר לעקוף אותה יחסית בקלות.

על המחבר

דור נאמני (Dor00tkit), קורא אדוק של Digital Whisper עוד מגיל צעיר, תמיד נמשכתי לעולם ה-Low Level והנדסה לאחור בעיקר מהמאמרים שפורסמו פה. יש משהו מיוחד בלחזור למקום שליווה אותי בתחילת הדרך. עבורי, כתיבת המאמר הזה היא סוג של סגירת מעגל והזדמנות קטנה להחזיר למקום שהשפיע עליי כל כך הרבה.

תודות

לבורא עולם, למשפחה שלי, למייסדים והעורכים של Digital Whisper, ניקס וקהילת האקינג IL. כסיף, רז(יאל), ירדן שפיר, ליאור קשת, d4d, Amirag, elicn, zvikam, Dvd848, b0rlord, Sub, reaction, megabeets, Antartic, Zerith , Rainbow_Dash, omerk2511, Improvement, J_Cobra, CescFabregas, Syst3m ShuTd0wn, P, iTK98 ,Alex Ionescu, XenoKovah - OpenSecurityTraining2, LiveOverflow, Gynvael Coldwind, Sinaei, Daax, ergot86, Satoshi Tanda, Pavel Yosifovich, Michael Maltsev. מ.ב, ה.ג, א.ד, ק.ד, ע.פ, א.י, ע.מ, ט.ח, א.ר, ע.ז, ר.ק, ב.ה, א.מ, ע.ד, י.ש, ת.ב, ב.ש, ח.פ, ש.ג, ז.ר.

מקורות מידע

- https://www.digitalwhisper.co.il/files/Zines/0x04/DW4-3-Anti-Anti-Debugging.pdf
- https://www.digitalwhisper.co.il/files/Zines/0x58/DW88-3-AntiReversing.pdf
- https://www.digitalwhisper.co.il/files/Zines/0x5A/DW90-3-PE.pdf
- https://www.digitalwhisper.co.il/files/Zines/0x10/DW16-2-SignatureDetectionBypass.pdf
- https://www.unknowncheats.me/