Malicious Documents Analysis Lab Guide

[GlobalSign] ACTION REQUIRED - Please Sign - Budget Approval.docm

Lab Solutions

1. Are there any macros present in the document?

Yes, the document contains VBA macros.

2. Identify the entrypoint of the macros.

The `AutoOpen` subroutine is automatically executed when the user clicks "*Enable Content*". This subroutine simply calls another function named `lhw409naw3`.

3. Consider the sequence of calls in function `lhw409naw3` where a variable is being repeatedly set. What type of value does this variable contain when the sequence is complete?

The malware reconstructs via concatenation a long Base64 string that has been split into four different functions.

4. How is the above-mentioned value decoded?

The malware passes the Base64 string to the function bHah394nh which decodes the Base64 encoding. The array of bytes returned from this function is then XOR-decoded with the key `119` (`0x77`).

5. Where is the final payload written to disk?

`%TEMP%\qapowengap.exe`

6. How could you extract the final payload?

The malware does not execute or delete the payload once it is dropped, so one method is to let the malware run and retrieve the payload from disk. Another method is to insert a VBA snippet to dump the binary data to another specified path once the payload is decoded.

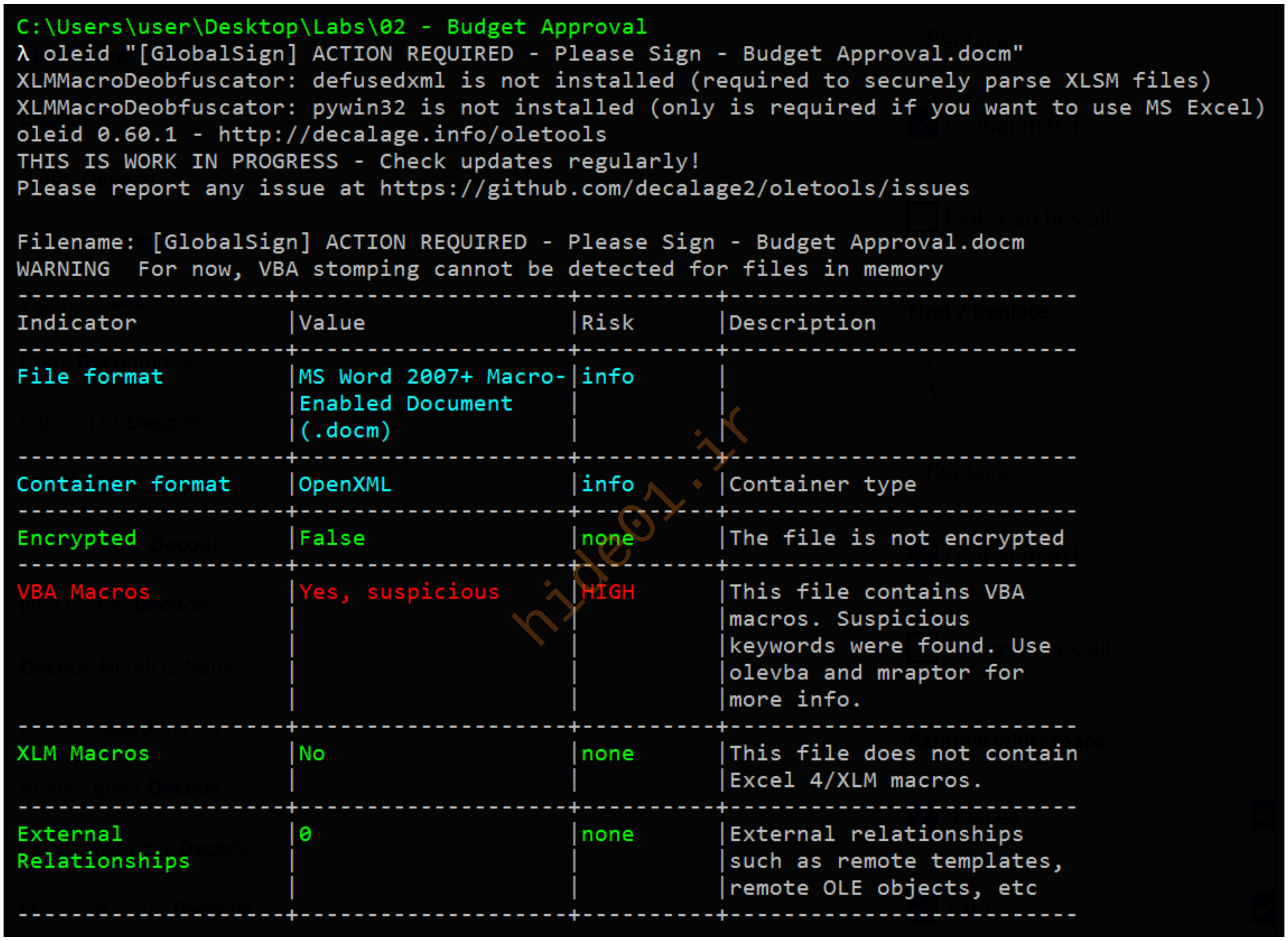
7. What is the final payload? Hint – it is safe to run, we promise.

It is an NFT simulator game.

Lab Walkthrough

1. Are there any macros present in the document?

Run *oleid* on the command line to detect macros.



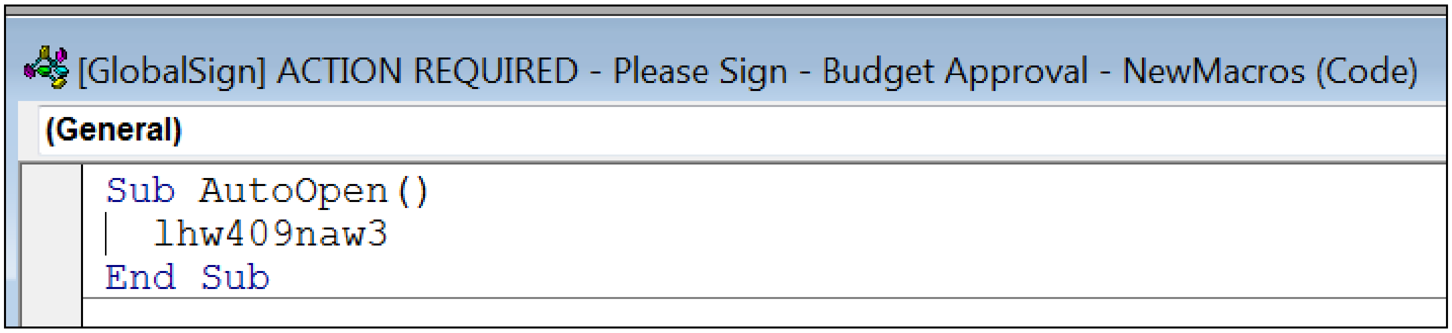
```
C:\Users\user\Desktop\Labs\02 - Budget Approval
λ oleid "[GlobalSign] ACTION REQUIRED - Please Sign - Budget Approval.docm"
XLMMacroDeobfuscator: defusedxml is not installed (required to securely parse XLSM files)
XLMMacroDeobfuscator: pywin32 is not installed (only is required if you want to use MS Excel)
oleid 0.60.1 - http://decalage.info/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

Filename: [GlobalSign] ACTION REQUIRED - Please Sign - Budget Approval.docm
WARNING  For now, VBA stomping cannot be detected for files in memory
--------------------+--------------------+----------+--------------------------
Indicator           |Value               |Risk      |Description
--------------------+--------------------+----------+--------------------------
File format         |MS Word 2007+ Macro-|info      |
                    |Enabled Document    |          |
                    |(.docm)             |          |
--------------------+--------------------+----------+--------------------------
Container format    |OpenXML             |info      |Container type
--------------------+--------------------+----------+--------------------------
Encrypted           |False               |none      |The file is not encrypted
--------------------+--------------------+----------+--------------------------
VBA Macros          |Yes, suspicious     |HIGH      |This file contains VBA
                    |                    |          |macros. Suspicious
                    |                    |          |keywords were found. Use
                    |                    |          |olevba and mraptor for
                    |                    |          |more info.
--------------------+--------------------+----------+--------------------------
XLM Macros          |No                  |none      |This file does not contain
                    |                    |          |Excel 4/XLM macros.
--------------------+--------------------+----------+--------------------------
External            |0                   |none      |External relationships
Relationships       |                    |          |such as remote templates,
                    |                    |          |remote OLE objects, etc
--------------------+--------------------+----------+--------------------------
```

Figure 1: oleid detects VBA macros

The text highlighted in red indicates the *oleid* has detected VBA macros.

2. Identify the entrypoint of the macros.

Open the document in Word and navigate to the Macro view. Observe the function `AutoOpen`. This executes automatically if/when macros are enabled. It calls the function `lhw409naw3`.


[GlobalSign] ACTION REQUIRED - Please Sign - Budget Approval - NewMacros (Code)
(General)

```
Sub AutoOpen()
    lhw409naw3
End Sub
```

Figure 2: AutoOpen routine

3. Consider the sequence of calls in function lhw409naw3 where a variable is being repeatedly set. What type of value does this variable contain when the sequence is complete?

Observe the calls where *awev09vn* is set, then concatenated several times.

```
Public Function lhw409naw3() As Variant
    Dim awev09vn As String
    Dim GHhn03nh() As Byte
    Dim ygAGWEnv2890hg As Long
    awev09vn = bn5ASDrgavwea()
    awev09vn = awev09vn & gAwefin3JIJG54()
    awev09vn = awev09vn & yFinvpawjoe5()
    awev09vn = awev09vn & pGgAWeoBjop06x()
```

Figure 3: awev09vn is set and modified

It is initially set as the result of a call to `bn5ASDrgavwea`. Observe that funcion. It is a very long sequence where a Base64 string is assembled. Each of the other functions continues this pattern, each time appending the new Base64 to the previously assembled portion. At the end of the sequence `awev09vn` contains a large Base64 string.

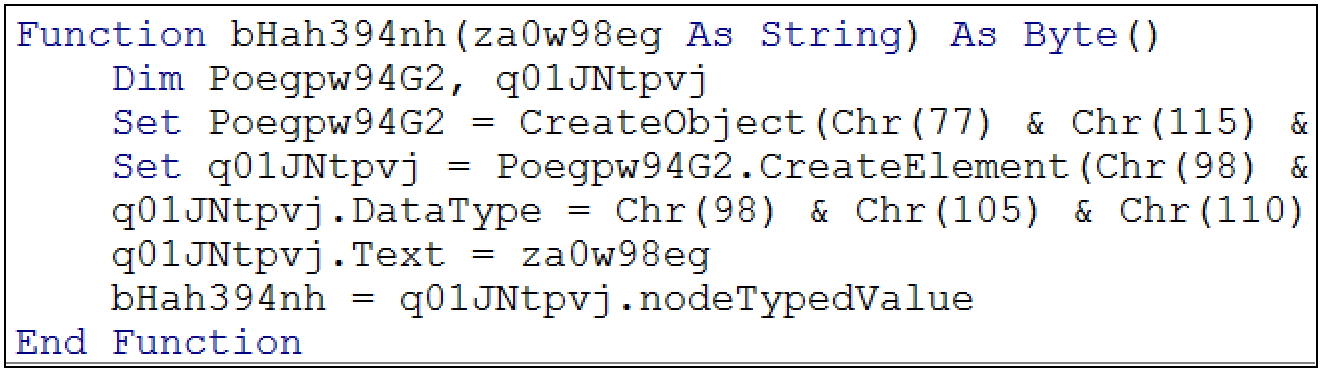
4. How is the above-mentioned value decoded?

Consider the next code sequence.

```
GHhn03nh = bHah394nh(awev09vn)
For ygAGWEnv2890hg = LBound(GHhn03nh) To UBound(GHhn03nh)
  GHhn03nh(ygAGWEnv2890hg) = GHhn03nh(ygAGWEnv2890hg) Xor 119
Next
```

Figure 4: Decoding sequence

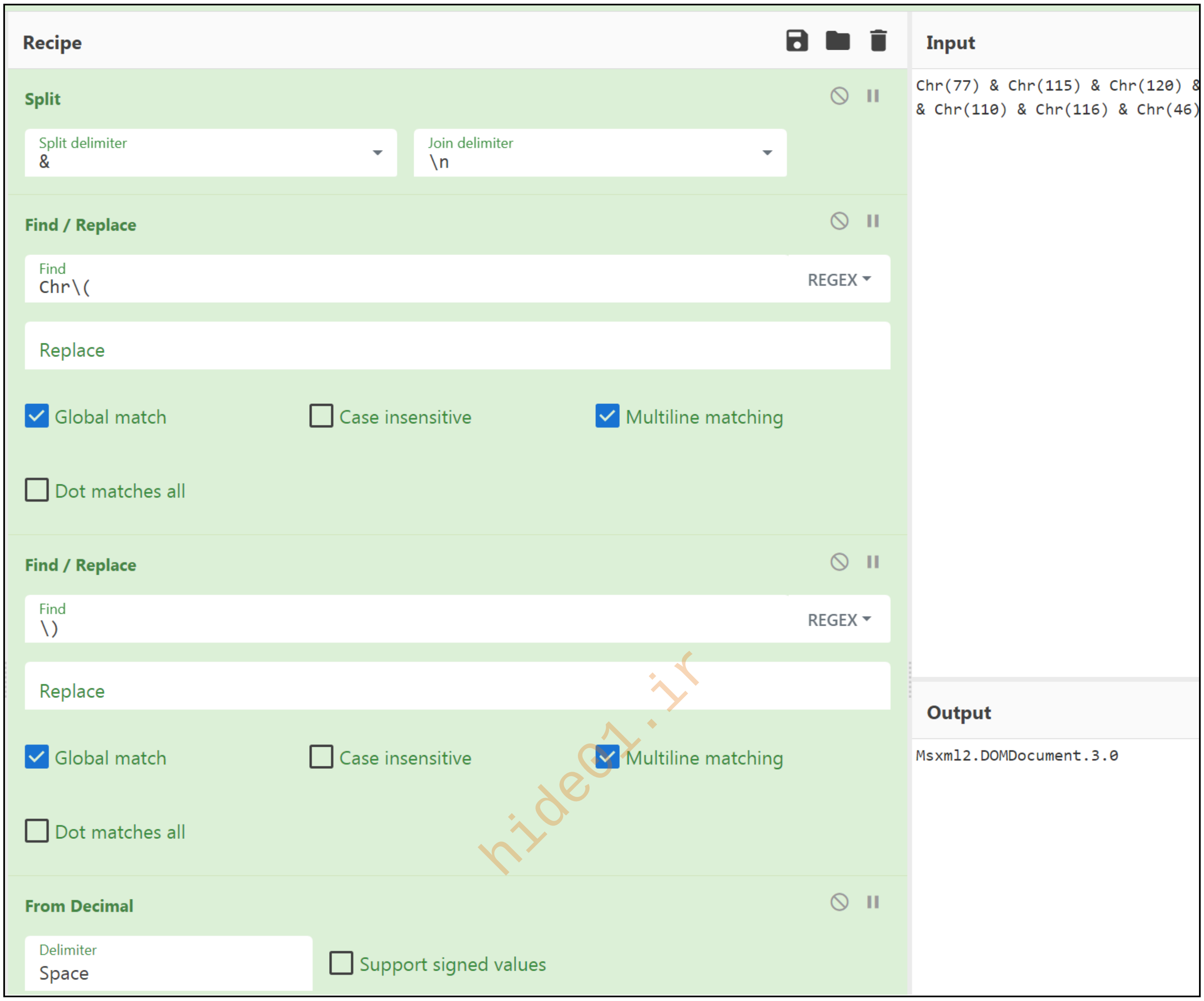
The function call to bHah394nh takes the Base64 string as an argument. Consider that function.



```
Function bHah394nh(za0w98eg As String) As Byte()
    Dim Poegpw94G2, q01JNtpvj
    Set Poegpw94G2 = CreateObject(Chr(77) & Chr(115) &
    Set q01JNtpvj = Poegpw94G2.CreateElement(Chr(98) &
    q01JNtpvj.DataType = Chr(98) & Chr(105) & Chr(110)
    q01JNtpvj.Text = za0w98eg
    bHah394nh = q01JNtpvj.nodeTypedValue
End Function
```

Figure 5: Base64 decoding routine (truncated)

The strings in this function are obfuscated. They are composed of individual characters that are assembled with the `Chr` function and concatenated with the & operator. To decode the strings, copy the `Chr` sequences into *CyberChef*. Use the *Split*, "*Find / Replace*", and "*From Decimal*" rules.



Recipe

Split

Split delimiter: &

Join delimiter: \n

Find / Replace

Find: Chr\(REGEX

Replace

[✓ Global match] Case insensitive [✓ Multiline matching]

Dot matches all

Find / Replace

Find: \) REGEX

Replace

[✓ Global match] Case insensitive [✓ Multiline matching]

Dot matches all

From Decimal

Delimiter: Space

Support signed values

Input

```
Chr(77) & Chr(115) & Chr(120) &
& Chr(110) & Chr(116) & Chr(46)
```

Output

```
Msxml2.DOMDocument.3.0
```

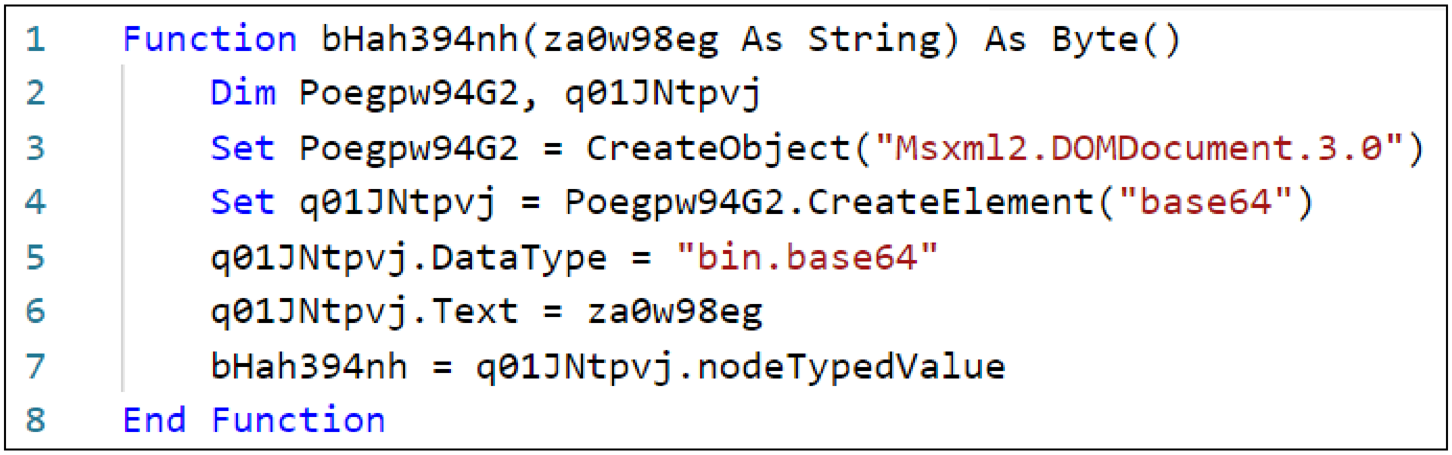
Figure 11: Decoding strings with CyberChef (input truncated)

In this example *Split* is used to separate the characters, "*Find / Replace*" is used to remove the function syntax and isolate the decimal value. "*From Decimal*" is used to decode the decimal to ASCII.

Paste the following path into your FLARE VM web browser to reproduce the recipe:

```
file:///C:/ProgramData/chocolatey/lib/cyberchef.flare/tools/cyberchef.html#recipe=Spl
it('%26%20','%5C%5Cn')Find_/_Replace(%7B'option':'Regex','string':'Chr%5C%5C('%7D,'',
true,false,true,false)Find_/_Replace(%7B'option':'Regex','string':'%5C%5C)'%7D,'',tru
e,false,true,false)From_Decimal('Space',false)
```

Repeat this process to decode all the strings in the function. Copy the function text into *VS Code* and replace the decoded strings. Set the *Language Mode* to *Visual Basic* to activate syntax highlighting (found at bottom-right of *VS Code* display).

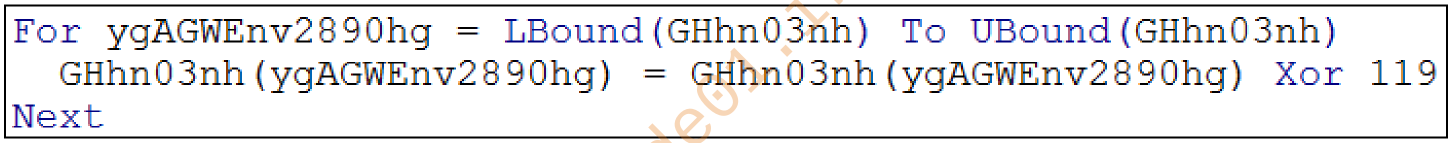


```
Function bHah394nh(za0w98eg As String) As Byte()
    Dim Poegpw94G2, q01JNtpvj
    Set Poegpw94G2 = CreateObject("Msxml2.DOMDocument.3.0")
    Set q01JNtpvj = Poegpw94G2.CreateElement("base64")
    q01JNtpvj.DataType = "bin.base64"
    q01JNtpvj.Text = za0w98eg
    bHah394nh = q01JNtpvj.nodeTypedValue
End Function
```

Figure 10: Function with strings decoded and syntax highlighting

Without scrutinizing each line, the strings suggest that this function performs Base64 decoding. This makes sense considering the incoming argument is a Base64 string. Recall that in Visual Basic the return value is declared by setting the value of the function name. In this case bHah394nh is set to the decoded text which means the function returns the decoded Base64 data.

In the next sequence the decoded Base64 text is modified further.



```
For ygAGWEnv2890hg = LBound(GHhn03nh) To UBound(GHhn03nh)
  GHhn03nh(ygAGWEnv2890hg) = GHhn03nh(ygAGWEnv2890hg) Xor 119
Next
```

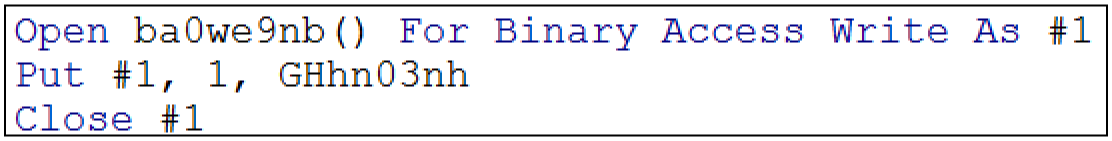
Figure 12: Further decoding

In this loop, `GHhn03nh` contains the Base64 string described in question 3. `ygAGWEnv2890hg` is the current index, starting at the beginning (`LBound`). On each iteration of the loop the current value is XORed with the decimal value `119` (`0x77`).

In total the data is Base64 decoded and XORed with `0x77`.

5. Where is the final payload written to disk?

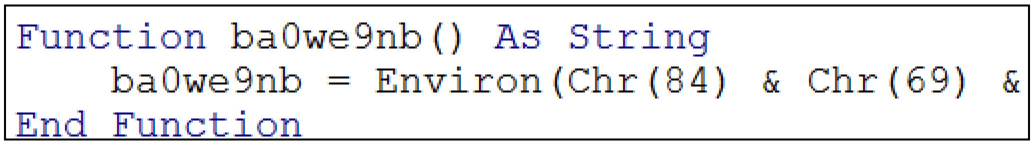
Consider the next code sequence.



```
Open ba0we9nb() For Binary Access Write As #1
Put #1, 1, GHhn03nh
Close #1
```

Figure 13: Open and write unknown file

Consider the function called here to get the filename, `ba0we9nb`.



```
Function ba0we9nb() As String
    ba0we9nb = Environ(Chr(84) & Chr(69) &
End Function
```

Figure 14: Encoded filename (truncated)

Decode the string using the previous *CyberChef* recipe. The value is `TEMP\qapowengap.exe`. This value is returned and used as the argument for `Open`. Then the contents of `GHhn03nh` are written to the file. This is the final payload.

6. How could you extract the final payload?

There are several viable strategies. One option is to manually decode the Base64, but that involves copying many lines of code which can be challenging. The easier alternative is to use dynamic analysis. Simply run the code and navigate to `%TEMP%\qapowengap.exe`.

7. What is the final payload? Hint – it is safe to run, we promise.

Run the payload and observe the behavior. If you have experience with reverse engineering, try opening it in IDA or Ghidra. It is a game for running an NFT business.