VBA/VBScript VXeRs & RedTeam

By h0ffy // @JennyLab

Contents

1. Introduction

In this document, we will explore how to use VBA (Visual Basic for Applications) and VBScript to automate tasks and manipulate Word documents. These scripting languages are powerful tools for developers and IT professionals seeking to automate repetitive processes and increase efficiency in document management. Moreover, we will delve into scenarios related to Red Team and security testing, demonstrating how these technologies can be used both legitimately and, eventually, in threat-simulation contexts.

1.1. What is VBA?

VBA, or Visual Basic for Applications, is a programming language developed by Microsoft, widely used for automating tasks within the Microsoft Office application ecosystem. Designed to integrate natively with programs like Word, Excel, PowerPoint, and Outlook, VBA allows users to extend the capabilities of these tools by means of custom scripts that interact with their objects, methods, and properties. Its usefulness is not limited to administrative automation, but it also stands out in security evaluations and penetration tests thanks to its flexibility and deep access to system functions. VBA allows interaction with COM (Component Object Model) objects and with the Windows operating system, making it easier to manipulate Word documents and Excel spreadsheets, send automated emails from Outlook, and create presentations in PowerPoint. Moreover, it offers advanced capabilities to interact with OS APIs, perform WMI (Windows Management Instrumentation) queries, and execute external DLL library functions, making it a powerful tool for security evaluations in corporate environments.

A typical example of its use is creating macros in Excel to process large volumes of data automatically. However, as with VBScript, malicious actors have exploited this capability to insert code that performs harmful actions, such as downloading payloads or accessing sensitive data. Because of this, knowledge of VBA is valuable not only for legitimate task automation, but also for identifying and mitigating potential attack vectors in an IT security context.

Despite the risks, VBA continues to be a relevant tool in enterprise environments, especially in legacy systems where custom automation is still critical. Its capacity to work with external libraries and execute dynamic code makes it attractive for advanced simulations of malicious activities in Red Team and Pentesting scenarios.

Some fundamental concepts of VBA include the use of explicit variables with Dim, loop handling (For, While) and conditional structures (If...Then...Else), as well as advanced object and event manipulation. Moreover, VBA enables calling external library functions via statements such as Declare and PtrSafe. This makes it an indispensable tool for system administrators, security analysts, and Red Team teams who need to automate processes, evaluate security settings, and simulate attacks in Windows environments.

1.2. What is VBScript?

VBScript is a light version of Visual Basic specifically designed for creating scripts that simplify task automation in Windows systems. It is characterized by being simpler and less powerful than VBA (Visual Basic for Applications), yet it is very useful for implementing scripts aimed at relatively straightforward or moderately complex tasks in environments where advanced control or more robust tools are not required. VBScript is widely used in the field of system administration, as it enables direct interaction with the operating system and applications through WSH (Windows Script Host), a runtime environment that allows scripts with .vbs or .js extensions to be executed. In addition, VBScript can manage files, automate system configurations, and perform basic application control. Despite its usefulness, VBScript has fallen out of favor in recent years due to the shift toward more modern and secure technologies, as well as Microsoft's restrictions to mitigate vulnerabilities—since VBScript was once a frequent distribution vector for malware due to its ability to run on systems with few restrictions and its easy integration with browsers like Internet Explorer. Nevertheless, in certain legacy environments, VBScript still finds practical use for quick automation and lightweight maintenance scripts.

NOTE: One important detail to mention for those coming from a RedTeam background or small “kiddies” aspiring to become one. In the real world, where you face increasingly advanced security systems day by day, having tools and payloads in these languages at your disposal can determine the success or failure of an advanced attack on infrastructure.

NOTE: We assume at all times that the RedTeam-related audience already has an arsenal and intends to enhance it; if that is not the case... This might be a good time to start building one and get your first tips on payloads and attack methodologies. Future sections will be added to this document with ideas and techniques for developing frameworks and attacks, as well as using some development frameworks (although I can’t say "onpromise" because my personal tendency is to write them myself. I also hold a personal dislike toward anything I’ve programmed in the past, and such motives typically keep me away from those vile... I was once a kiddie #script_kiddies on Arrakis ;)

1.3. Document Objectives

Master the use of VBA for advanced Word document manipulation. Analyze practical examples integrating VBA and VBScript to solve specific tasks. Understand how to interact with other scripting languages from VBA. Apply best practices and consider key aspects when working with VBA and VBScript.

1.4. Document Structure

The document is organized into sections that address various aspects of using VBA and VBScript. Each section provides code examples along with clear, detailed explanations, intended to facilitate the reader’s understanding and practical application of the concepts.

1.5. Audience

The primary audience for this document includes RedTeam experts, developers, IT professionals, and tech enthusiasts interested in learning how to use VBA and VBScript to automate tasks and manipulate Word documents. No prior programming experience is required, but basic scripting and programming knowledge is recommended.

1.6. Prerequisites It is necessary to have a development environment compatible with VBA and VBScript, such as Microsoft Office or a text editor that supports Windows scripts. A Windows operating system is also required to run and test the scripts.

2. VBA

2.1. Using VBA as a Scripting Language for Documents

Let's do a common instrumentation as one would typically do in a Word document. To accomplish this in VBA, you can iterate through the contents of the Word document, searching for the strings and replace them with an image and text respectively ({{ **foto** }} and {{ **texto1** }})

Here is an example that can help you better understand how to do it:

```
Sub InsertarContenido()
    Dim doc As Document
    Dim rango As Range
    Dim imagenPath As String
    Dim texto As String

    ' Asume que el documento ya está abierto
    Set doc = ActiveDocument
    ' Ruta de la imagen que quieres insertar
    imagenPath = "C:\Users\Public\\Documents\jennylogo.jpg"
    ' Texto a insertar
    texto = "JennyL4b Is Sexy!"
    ' Buscar y reemplazar {{ foto }} por la imagen
    Set rango = doc.Content
    With rango.Find
        .ClearFormatting
        .Text = "{{ foto }}"
        If .Execute Then
            ' Insertar la imagen en el lugar donde se encontro {{ foto }}
            rango.InlineShapes.AddPicture FileName:=imagenPath
        End If
    End With

    ' Buscar y reemplazar {{ texto1 }} por el texto deseado
    Set rango = doc.Content
    With rango.Find
        .ClearFormatting
        .Text = "{{ texto1 }}"
        If .Execute Then
            ' Reemplazar {{ texto1 }} por el texto
            rango.Text = texto
        End If
    End With
End Sub
```

2.1.1. Code Explanation:

Variable Definition: doc refers to the active document. rango is the portion of the document where you search and replace text. imagenPath contains the path of the image to be inserted. texto is the text that replaces the {{ texto1 }} string.

2.1.2. Searching for {{ foto }}

Uses the Find object to look for the {{ foto }} string. When found: The specified image in imagenPath is inserted at that position.

2.2.3. Searching for {{ texto1 }}

Similarly, looks for {{ texto1 }} and replaces it with the text specified in the texto variable.

2.1.4. Considerations

NOTE: Make sure the Word document is open before running the code. The image path must be valid and accessible from the VBA code. This code assumes there is only one occurrence of {{ foto }} and {{ texto1 }}. If there are multiple, the code may need adjustments to iterate through all instances.

2.1.5. Basic Example for Understanding the Language

```
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Sub PausarEjecucion()
    MsgBox "La ejecución se pausará durante 5 segundos"
    Sleep 5000 ' Pausa la ejecución por 5000 milisegundos (5 segundos)
    MsgBox "La ejecución ha reanudado"
End Sub
```

2.1.6. VBA Code Example for Calling GetProcAddress and LoadLibrary

This example uses the GetProcAddress function to obtain the address of a function within a loaded DLL. We will use the kernel32.dll library as an example.

```
Declare PtrSafe Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA" (ByVal lpFileName As String) As Long
Declare PtrSafe Function GetProcAddress Lib "kernel32" (ByVal hModule As Long, ByVal lpProcName As String) As Long
Declare PtrSafe Function GetModuleHandle Lib "kernel32" Alias "GetModuleHandleA" (ByVal lpModuleName As String) As Long
Declare PtrSafe Function MessageBox Lib "user32.dll" Alias "MessageBoxA" (ByVal hwnd As Long, ByVal lpText As String, ByVal lpCaption As String, ByVal uType As Long) As Long

Sub CargarDLLyObtenerFuncion()
    Dim hModule As Long
    Dim pFunc As Long
    Dim resultado As Long

    ' Cargar la DLL
    hModule = LoadLibrary("kernel32.dll")
    If hModule = 0 Then
        MsgBox "Error al cargar la DLL."
        Exit Sub
    End If
    ' Obtener la dirección de la función GetTickCount
    pFunc = GetProcAddress(hModule, "GetTickCount")
    If pFunc = 0 Then
        MsgBox "Error al obtener la dirección de la función."
        Exit Sub
    End If

    ' Llamar a la funcion GetTickCount (usando la dirección de la función)
    resultado = CallFunction(pFunc)
    ' Mostrar el resultado (el número de milisegundos desde que el sistema arrancó)
    MsgBox "Resultado de GetTickCount: " & resultado
End Sub

Function CallFunction(ByVal pFunc As Long) As Long
    ' Esta función usa la dirección de la función obtenida
    ' Llamada indirecta a la función GetTickCount
    ' Esto sería una simplificación; en un entorno real, se usarían mecanismos como "CallWindowProc" o APIs adicionales para invocar dinámicamente las funciones
    CallFunction = pFunc ' Simulación, solo para ilustrar el uso
End Function
```

2.1.7. VBA Code Example for Calling GetProcAddress and LoadLibrary

This example uses the GetProcAddress function to obtain the address of a function within a loaded DLL. We will use the kernel32.dll library as an example.

```
pFunc = GetProcAddress(hModule, "GetTickCount")
hModule = GetModuleHandle("kernel32.dll")
```

2.1.8. GDI Calls

```
Declare PtrSafe Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA" (ByVal lpFileName As String) As Long
Declare PtrSafe Function GetProcAddress Lib "kernel32" (ByVal hModule As Long, ByVal lpProcName As String) As Long
Declare PtrSafe Function CreatePen Lib "gdi32.dll" (ByVal fnPenStyle As Long, ByVal nWidth As Long, ByVal crColor As Long) As Long

Sub CrearLapis()
    Dim hModule As Long
    Dim pCreatePen As Long
    Dim hPen As Long

    ' Cargar la librería gdi32.dll
    hModule = LoadLibrary("gdi32.dll")

    ' Obtener la dirección de la función CreatePen
    pCreatePen = GetProcAddress(hModule, "CreatePen")

    If pCreatePen <> 0 Then
        ' Crear un lápiz (solo como ejemplo, no dibuja en VBA directamente)
        hPen = CreatePen(0, 2, RGB(255, 0, 0)) ' Estilo sólido, grosor 2, color rojo
        MsgBox "Lápiz creado con éxito: " & hPen
    Else
        MsgBox "No se pudo cargar la función CreatePen."
    End If
End Sub
```

2.1.9. Calling ntdll.dll

To illustrate access to advanced system functions, this section presents a practical example that uses VBA to call the ntdll.dll library. This example shows how to dynamically load system libraries and call specific functions, such as **NtQuerySystemInformation**, which provides detailed information about the operating system's status.

*ⓘ **IMPORTANT NOTE**: THERE ARE PROTECTIONS THAT LIMIT THIS; THIS TAKES PLACE IN A SPECIFIC CONTEXT THAT ENABLES THE EXECUTION OF THESE STEPS, AND LATER WE WILL DISCUSS THAT*

```
Declare PtrSafe Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA" (ByVal
lpFileName As String) As Long
Declare PtrSafe Function GetProcAddress Lib "kernel32" (ByVal hModule As Long,
ByVal lpProcName As String) As Long
Declare PtrSafe Function NtQuerySystemInformation Lib "ntdll.dll" (ByVal
SystemInformationClass As Long, ByVal SystemInformation As Long, ByVal
SystemInformationLength As Long, ByRef ReturnLength As Long) As Long

Sub ConsultarInformacionDelSistema()
    Dim hModule As Long
    Dim pNtQuerySystemInformation As Long
    Dim result As Long
    Dim buffer As Long
    Dim length As Long

    ' Cargar la libreria ntdll.dll
    hModule = LoadLibrary("ntdll.dll")

    ' Obtener la direccion de la función NtQuerySystemInformation
    pNtQuerySystemInformation = GetProcAddress(hModule,
"NtQuerySystemInformation")

    If pNtQuerySystemInformation <> 0 Then
        ' Llamar a NtQuerySystemInformation para obtener informacion del sistema
        result = NtQuerySystemInformation(0, buffer, 0, length)

        ' Mostrar el resultado (solo un ejemplo simple, puedes usar estructuras mas
complejas)
        MsgBox "Resultado de NtQuerySystemInformation: " & result
    Else
        MsgBox "No se pudo cargar la función NtQuerySystemInformation."
    End If
End Sub
```

2.1.10. Executing JavaScript from VBA

To run JavaScript from VBA, you can use the Internet Explorer JavaScript engine (in older Windows versions) via CreateObject("MSHTML.HTMLDocument"). Below is an example of how you might execute JavaScript in a browser object:

This technique is most commonly used when you need to interact with a web browser or manipulate the DOM of a page.

```
Dim IE As Object
Set IE = CreateObject("InternetExplorer.Application")
IE.Visible = False
IE.Navigate "about:blank"

' Ejecuta JavaScript
IE.document.parentWindow.execScript "alert('Hola desde VBA!')", "JavaScript"
```

2.1.11 Executing PowerShell from VBA

To run PowerShell from VBA, you can use WScript.Shell to run a PowerShell script.

This code executes a PowerShell command directly from VBA. You can use it to run PowerShell scripts or console commands within a VBA environment.

```
Dim objShell As Object
Set objShell = CreateObject("WScript.Shell")
objShell.Run "powershell -Command ""Write-Host 'Hello from PowerShell'"""
Set objShell = Nothing
```

2.1.12 Running Batch from VBA

You can also run Batch scripts from VBA in a way similar to how you would run PowerShell, using (WScript.Shell).

Here, the cmd.exe /C command runs a Batch script (or simply a console command).

```
Dim objShell As Object
Set objShell = CreateObject("WScript.Shell")
objShell.Run "cmd.exe /C echo Hello from Batch"
Set objShell = Nothing
```

2.1.13. Executing VBScript from VBA

VBA and VBScript are very similar languages, and you can directly invoke VBScript from VBA using **CreateObject("WScript.Shell")** to run a .vbs file or the VBScript code itself.

2.1.14. Executing VBScript from VBA: Loading .vbs

This code runs a .vbs file from VBA. You can use it to run VBScript stored in a .vbs file.

```
Dim objShell As Object
Set objShell = CreateObject("WScript.Shell")
objShell.Run "wscript.exe ""C:\ruta\al\script.vbs"""
Set objShell = Nothing
```

2.1.15. Executing VBScript from VBA: Embedded Code

This code embeds VBScript directly in VBA and runs it. You can use it to execute VBScript without needing external files.

```
Dim vbScript As String
vbScript = "Set objShell = CreateObject(""WScript.Shell"")" & vbCrLf & _
          "objShell.Popup ""Hello from VBScript!""" & vbCrLf & _
          "Set objShell = Nothing"
CreateObject("WScript.Shell").Run "wscript.exe """ & vbScript & """", 0, True
```

2.1.15. VBA Injection into an Open Word Document

This VBA code injects text and images into an open Word document. You can use this approach to personalize documents or templates with dynamic content.

This code assumes that the Word document is already open and contains placeholder markers such as {{texto1}} and {{foto}}, which will be replaced by specific text and images.

```
Sub InyectarContenidoEnWord()
    Dim doc As Document
    Dim rango As Range
    Dim texto As String
    Dim imagenPath As String
    ' Asume que el documento ya está abierto
    Set doc = ActiveDocument
    ' Texto a insertar
    texto = "JennyLab Is Pwn3r"
    ' Ruta de la imagen que quieres insertar
    imagenPath = "C:\Users\JennyLab\SexyPics\imagen.jpg"
    ' Inyectar texto
    Set rango = doc.Content
    rango.Find.Text = "{{texto1}}"
    If rango.Find.Execute Then
        rango.Text = texto
    End If
    ' Inyectar imagen
    Set rango = doc.Content
    rango.Find.Text = "{{foto}}"
    If rango.Find.Execute Then
        rango.InlineShapes.AddPicture FileName:=imagenPath
    End If
End Sub
```

2.1.16. Summary

- **JavaScript:** You can use the Internet Explorer engine to run JavaScript in VBA.
- **PowerShell:** Runs through WScript.Shell by launching the powers hell command.
- **Batch:** Similar to PowerShell, but using the cmd.exe command.
- **VBScript:** You can run .vbs files or embed VBScript directly in VBA.

These methods allow you to execute scripts or commands from other languages within VBA, which may be useful for integrations or more complex tasks that require an additional language.

3. VBS / VBScript / Visual Basic Script

VBScript or Visual Basic Script is a lightweight scripting language developed by Microsoft, designed to automate tasks within the Windows ecosystem. Its close integration with the operating system and applications like Microsoft Office makes it an interesting tool for automation, application interaction, and in some cases, for security evaluations and penetration testing. VBScript allows interaction with OS components and applications through COM (Component Object Model) objects. For example, you can use it to automate tasks in Word or Excel documents, send emails via Outlook, or perform advanced operations such as directory scanning, registry reading, and file manipulation. This makes it appealing for Red Team and Pentesting scenarios, where its ability to interact with system resources can be leveraged to assess security configurations or simulate malicious activities.

A typical example is creating a Word object to modify a document. This kind of interaction not only allows you to automate legitimate administrative tasks, but can also be exploited by malicious actors to insert macros or scripts that perform undesirable actions. In addition, thanks to its compatibility with WMI (Windows Management Instrumentation), VBScript can be used to collect system information, such as hardware and software details, which makes it a useful tool for reconnaissance.

Although VBScript was officially discontinued by Microsoft in browsers like Internet Explorer, it remains available on Windows systems, keeping it relevant for certain contexts, particularly in legacy environments. It is important to know its syntax and capabilities not only for automation and administration but also to identify potential attack vectors and strengthen defenses.

Some fundamental VBScript concepts include using explicit declarations (Dim, Set), handling conditional structures (If...Then...Else) and loops (For, While), as well as manipulating objects via their methods and properties. This language is an excellent starting point for anyone seeking to understand how scripts interact with the operating system and applications in a Windows environment, providing a solid foundation for Red Team and Pentesting activities.

3.1. Injecting Text and Images into an Open Document

This technique is an effective and versatile strategy in Red Team operations. The ability to inject content into an open Word document using VBScript offers multiple applications—from customizing documents or templates with dynamic content to more advanced targets related to security testing and attack simulations. As with VBA, VBScript can directly interact with a Word object to modify its content. This approach allows you to inject text, images, or other dynamic elements at runtime, providing a powerful tool for automated tasks. One of the key advantages of this technique is the ability to generate dynamic documents tailored to specific needs. For example, in a legitimate context, you can automate the creation of personalized reports, letters with customer data, or any kind of business document that needs to be generated in bulk. However, from a security testing perspective, this same capability can be exploited to create malicious documents intended to deceive users or analysis systems. A practical example is managing custom templates containing dynamic content, which allows generating seemingly authentic documents but designed to evade security controls or execute payloads.

The ability to inject content in real time is also helpful for simulating scenarios in which an attacker might modify a legitimate document to insert malicious content. This includes not only text but also images, hidden macros, or even links to external resources. By directly interacting with an open Word document, VBScript gives granular control over the file's content and layout, making it an ideal tool for operations requiring a high level of customization.

Moreover, this technique can be extended to interact with other Office components, which broadens its usefulness in diverse environments and scenarios. For example, it is possible to combine it with scripts to automatically send these generated documents via email or store them in shared locations, facilitating massive or controlled distribution. Its integration with the Office object model makes VBScript a powerful and flexible option for both legitimate tasks and threat simulations in a controlled environment, making it indispensable for Red Team teams and security professionals interested in understanding and mitigating risks linked to automation in Office environments.

```
Sub InyectarContenidoEnWord()
    Dim doc As Document
    Dim rango As Range
    Dim texto As String
    Dim imagenPath As String

    ' Asume que el documento ya está abierto
    Set doc = ActiveDocument

    ' Texto a insertar
    texto = "JennyLab h4x0rs!!."

    ' Ruta de la imagen que quieres insertar
    imagenPath = "C:\ruta\a\tu\imagen.jpg"

    ' Inyectar texto
    Set rango = doc.Content
    rango.Find.Text = "{{texto1}}"
    If rango.Find.Execute Then
        rango.Text = texto
    End If

    ' Inyectar imagen
    Set rango = doc.Content
    rango.Find.Text = "{{foto}}"
    If rango.Find.Execute Then
        rango.InlineShapes.AddPicture FileName:=imagenPath
    End If
End Sub
```

3.1.1. What about the Code?

This VBScript allows you to inject both text and images into an open Word document, using the COM object model to interact directly with the file. It is a powerful and versatile technique, applicable in multiple scenarios ranging from legitimate task automation to security simulations in Red Team operations.

3.1.2. Variable Definition

The first step in the script is to declare and define the necessary variables: a reference to the Word document object, the search range used to locate placeholders, the text values to be inserted, and the path of the image to be included in the document. These variables keep the script reusable and flexible, easily adapting to different needs or templates.

3.1.3. Text Injection

The script looks for a specific string in the document that acts as a placeholder, in this case {{texto1}}. Once found, this string is replaced with the text specified in the variables. This approach is ideal for templates that require dynamic customization, such as creating contracts, reports, or letters. Additionally, it provides precise control over the content's location, ensuring the final result fits the document's design.

3.1.4. Image Injection

Similarly, the script looks for a placeholder string for the image, for example {{foto}}. Once located, the script inserts the specified image from the given path at that position. The script leverages Word's object model capabilities to automatically adjust the size and position of the image, ensuring seamless integration into the document layout. This is particularly useful for generating visually appealing documents or automatically inserting logos, digital signatures, graphics, or other visual elements.

3.1.5. Customization and Improvements

This approach can be expanded to handle multiple types of dynamic content, including tables, charts, or even links to external resources. You can also implement loops to process multiple documents automatically, enabling you to work with large data sets or customize multiple documents in a single process—making it valuable for both administrative tasks and security exercises.

Red Team and Pentesting Use Cases: In a security testing context, these types of scripts can be used to generate decoy documents containing malicious payloads or to assess detection systems' ability to identify suspicious modifications in legitimate documents. The integration of dynamic content enhances these documents' credibility, making them more effective in social engineering or phishing test simulations.

By combining automation and customization, leveraging Word's object model to create documents that meet specific needs (be they administrative or security-related), VBScript stands out as a powerful and versatile tool with practical applications across various environments. Below are some examples of how VBScript can be used to interact with Word documents.

NOTE: The following example code shows the main object on which all content injection into an open Word document will be based:

Example:

```
CreateObject("Word.Application") 'Creates an instance of the Word application within the VBScript environment.
CreateObject("{000209FF-0000-0000-C000-000000000046}") 'Creates an instance of the Word application.
```

- **Open the document:** The Word document is opened with Documents.Open and assigned to a variable for manipulation.

```
Set doc = objWord.Documents.Open("C:\ruta\al\documento.docx")
```

- **Find and replace:** Similar to the VBA example, you use the Find object to look for specific markers in the document and then insert text or images. Insert text: You can use the InsertAfter method to add text at the end of the document.

```
doc.Content.InsertAfter "Hello kitty!! al final del documento."
```

3.1.6. Considerations

Both methods require that the Word file be either already open or opened from the script.

Whether you use VBScript or VBA depends on preference, but VBA is generally more powerful because it offers a smoother integration with Microsoft applications. In the case of VBScript, if you want to modify documents automatically on a machine without user interaction, you need to ensure that Word is installed and configured correctly on that system. If your goal is to inject content from an external source into a document without opening it (for instance, from another file or application), this is also possible, but it depends on interop libraries or COM interfaces to interact with the document without opening it in Word. This type of integration is especially useful when working with templates or automated processes where certain parts of the document may change dynamically, such as in reports or specialized document generators.

3.2. Red Team and Pentesting Uses

In security testing contexts, these scripts can be utilized to generate decoy documents containing malicious payloads or to evaluate how well detection systems identify suspicious changes in legitimate documents. The integration of dynamic content increases these documents' credibility, making them more effective in social engineering or phishing simulations.

By combining automation and customization, and leveraging the Word object model to create documents that satisfy specific needs—whether for administrative tasks or security evaluations—VBScript emerges as a powerful and versatile tool with practical uses in various environments. Below are some examples of how VBScript can interact with Word documents, for instance, showing the main example on which all content injection in an open Word document will be based.

💡 **NOTE:** This could also be applied to other OLE objects, COM, or the Windows shell. From a Red Team perspective, that is quite interesting for generating payloads. Investigate, 🛠️, and pwn3d.

Code:

```
CreateObject("Excel.Application") 'Creates an instance of the Excel application in the VBScript environment.
CreateObject("PowerPoint.Application") 'Creates an instance of the PowerPoint application in the VBScript environment.
CreateObject("Outlook.Application") 'Creates an instance of the Outlook application in the VBScript environment.
CreateObject("Scripting.FileSystemObject") 'Creates an instance of the file system for file manipulation.
CreateObject("WScript.Shell") 'Creates an instance of the Windows Shell environment.
CreateObject("InternetExplorer.Application") 'Creates an instance of the Internet Explorer browser.
CreateObject("ADODB.Connection") 'Creates an instance for managing database connections.
CreateObject("MSXML2.XMLHTTP") 'Creates an instance for making HTTP requests.
CreateObject("SAPI.SpVoice") 'Creates an instance of the Windows Speech API for text-to-speech.
CreateObject("Shell.Application") 'Creates an instance for accessing advanced file system operations.
```

💡 **NOTE:** It is important to mention that you can make these calls through PowerShell. You can even interact to work with VBScript or VBA alongside PowerShell or .NET.

ℹ️ **NOTE:** For now, the document "PowerShell Language for VXerS or RedTeam" has not been added. If it ultimately isn't included, this information will be part of this document.

4. Additional CLSID Example

```
CreateObject("{000209FF-0000-0000-C000-000000000046}") 'Word application
CreateObject("{00024500-0000-0000-C000-000000000046}") 'Excel app
CreateObject("{91493441-5A91-11CF-8700-00AA0060263B}") 'PowerPoint app
CreateObject("{0006F03A-0000-0000-C000-000000000046}") 'Outlook application
CreateObject("{0D43FE01-F093-11CF-8940-00A0C9054228}") 'FileSystemObject
CreateObject("{72C24DD5-D70A-438B-8A42-98424B88AFB8}") 'Windows Shell
CreateObject("{0002DF01-0000-0000-C000-000000000046}") 'Internet Explorer
CreateObject("{00000514-0000-0010-8000-00AA006D2EA4}") 'ADODB.Connection
CreateObject("{F5078F32-C551-11D3-89B9-0000F81FE221}") 'MSXML2.XMLHTTP
CreateObject("{96749377-3391-11D2-9EE3-00C04F797396}") 'SAPI.SpVoice
CreateObject("{13709620-C279-11CE-A49E-444553540000}") 'Shell.Application
```

MSOffice Example:

```
' Example of more CLSIDs that can be useful in different scenarios
CreateObject("{00020906-0000-0000-C000-000000000046}") ' Word.Document
CreateObject("{0002450F-0000-0000-C000-000000000046}") ' Excel.Sheet
```

In this list, you can integrate CLSIDs of other applications or custom COM libraries. As you explore corporate environments, you may discover more specific CLSIDs for controlling different Windows components or third-party software.