Persistence Techniques That Persist

cyberark.com/resources/threat-research-blog/persistence-techniques-that-persist

March 2, 2023

Abstract

Once threat actors gain a foothold on a system, they must implement techniques to maintain that access, even in the event of restarts, updates in credentials or any other type of change that might disrupt access. These techniques are collectively known as **persistence techniques**. In this blog post, we will focus on how malware can achieve persistence by abusing the Windows Registry. Specifically, we will focus on lesser-known techniques, many of which have been around since the days of Windows XP and are just as effective today on Windows 10 and 11.

Into Persistence

Today, most malware uses at least one persistence technique, even when a single run is all it takes to cause most of the damage (e.g., ransomware). Persistence, however, is most useful for stealthy campaigns meant to last a long time. In these situations, many infected machines will likely undergo a reboot at least once, terminating the malware process. Therefore, malware developers must add or implement a persistence mechanism to start again.

Several persistence mechanisms are often used legitimately. Most notable are Run Keys, Services and Scheduled Tasks. Because these methods have many legitimate uses, defenders can't blindly prevent all processes from using them. This is likely why malware developers favor these techniques, making them the most commonly used persistence techniques. However, because the methods are so well known, we can easily detect their use. This makes these techniques counterproductive in many situations since setting up the persistence can lead to the malware being discovered and removed. Because of this, malware will occasionally use less well-known techniques, hoping to avoid detection and allowing them to persist for longer.

Registry Terminology

Let's clarify some terminology used when discussing the registry. Firstly, we should mention that people often aren't very consistent with terminology, which can confuse things. In this blog post, we have tried to be consistent and only use the following terms: key, sub-key, value and set.

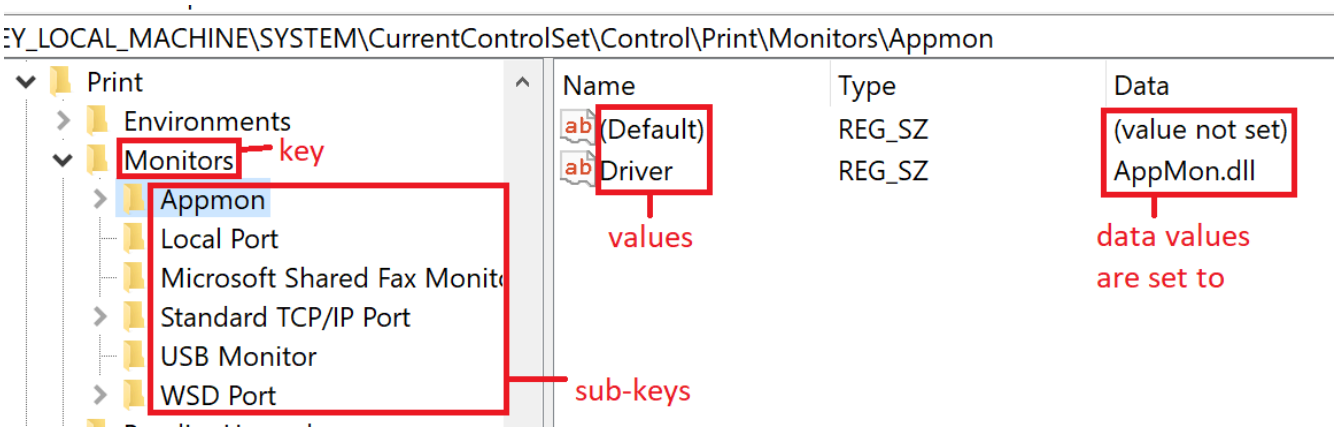


Figure 1. Registry terminology examples.

We can think of the registry like a file system. In this way, the keys and sub-keys would be the equivalent of folders and subfolders. In the same way, values are like files, and when a value is set to something, it's like having content in the file. To help illustrate this, let's take a look at the different kinds of lists that appear in the registry, which will be helpful later.

Lists tend to appear in the registry in one of three ways:

1. Lists of sub-keys (e.g., adding a port monitor), which would be like a list of all the subfolders in a specific folder.

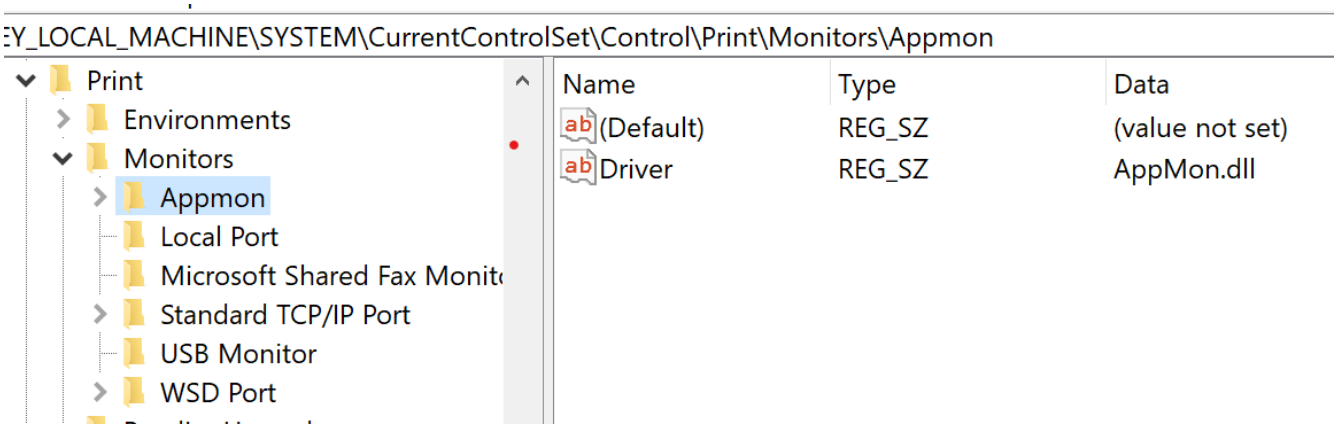


Figure 2. List of Monitors sub-keys, with each pointing to a different dll.

2. A single key with a list of values (e.g., Run Keys), which is like a list of all the files in a specific folder.

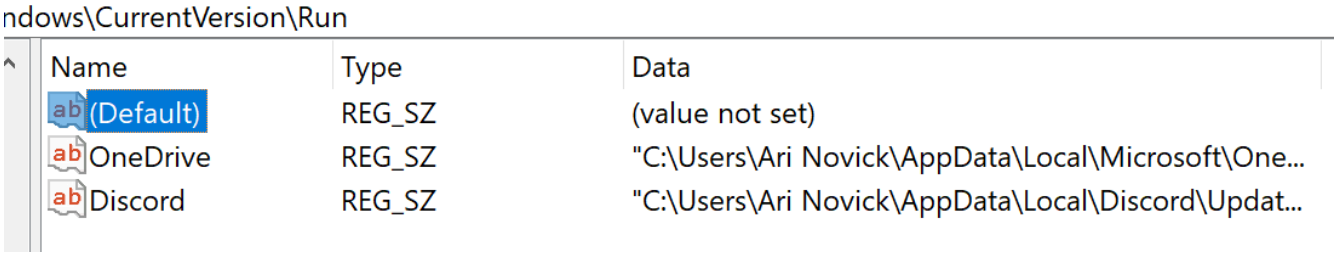


Figure 3. List of values in the Run Key.

3. A single multiline value in which each line is another item on the list (e.g., adding an authentication package to LSA), which is like a list of rows in the content of a specific file

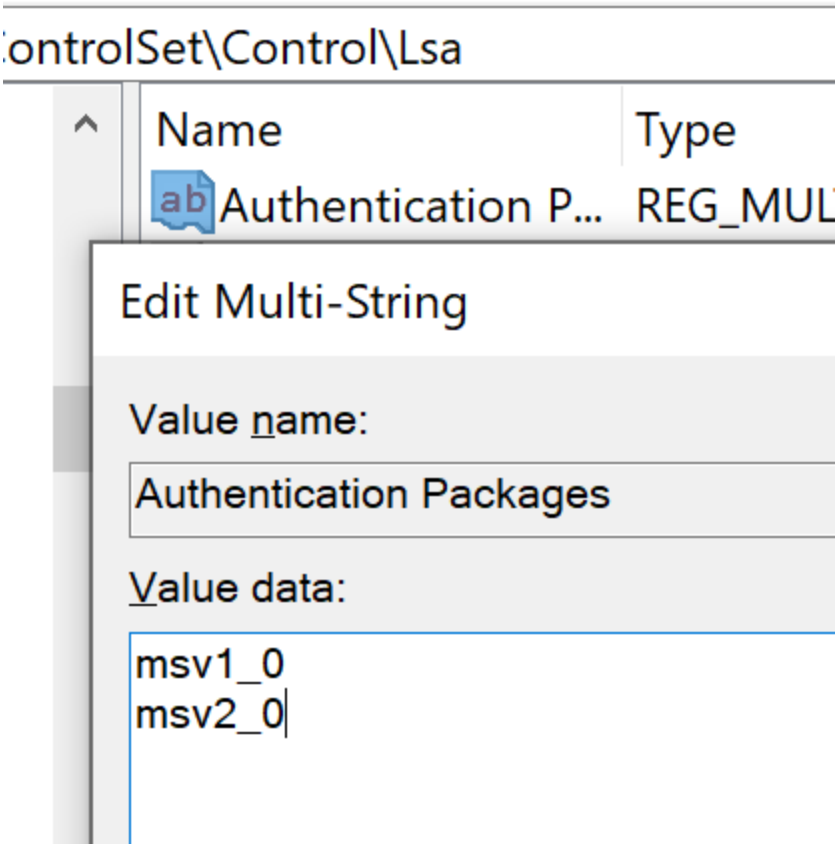


Figure 4. List of Authentication Packages to be used by Lsass.

Persistence in the Registry

There is an enormous range of persistence techniques that make use of the registry. Despite their variety, they all tend to follow the same basic steps:

1. The malware creates or modifies a key, setting a value in the key to one of two options:
    1. The file path of the malware.
    2. The CLSID/ProgID of the malware (only in cases where the malware is a COM server).
2. The malware process terminates, usually due to a reboot.
3. After each reboot, a process that is a part of the startup will check registry to determine if or what to load and will also load the newly created or modified value.
4. The process will then execute the malware as defined in the registry key.

The registry-based persistence techniques can be divided up as follows:

	Overriding Existing Key	Creating New Keys	Adding an Item to a List
Difficulty of finding "Unknown Techniques" with Procmon	Easy	Difficult	Easy
Difficulty of missing the change in the hijacked processes' behavior	Difficult	Easy	Easy

Overriding an existing key is a common approach for malware persistence. For example, there exists a key that UserInit uses to determine what shell to start after login. By default, this key is set to explorer.exe, but malware can modify it, setting itself up as a shell to be started. If we wanted to find new potential techniques like these, it's reasonably easy with the Sysinternals tool Procmon. For example, if a process executes a binary based on a path set in the registry, we could find it by filtering in Procmon for all the registry events related to the process where the Details field ends with ".exe." Once we find the key, overriding its value will result in the execution of a different binary, which attackers can use to run their malware. However, this approach has a significant disadvantage since overriding the key often results in abnormal behavior stemming from the process that accesses the modified key. As a result, this can potentially alert a user on an infected machine that something isn't right and ultimately result in the malware being discovered and removed.

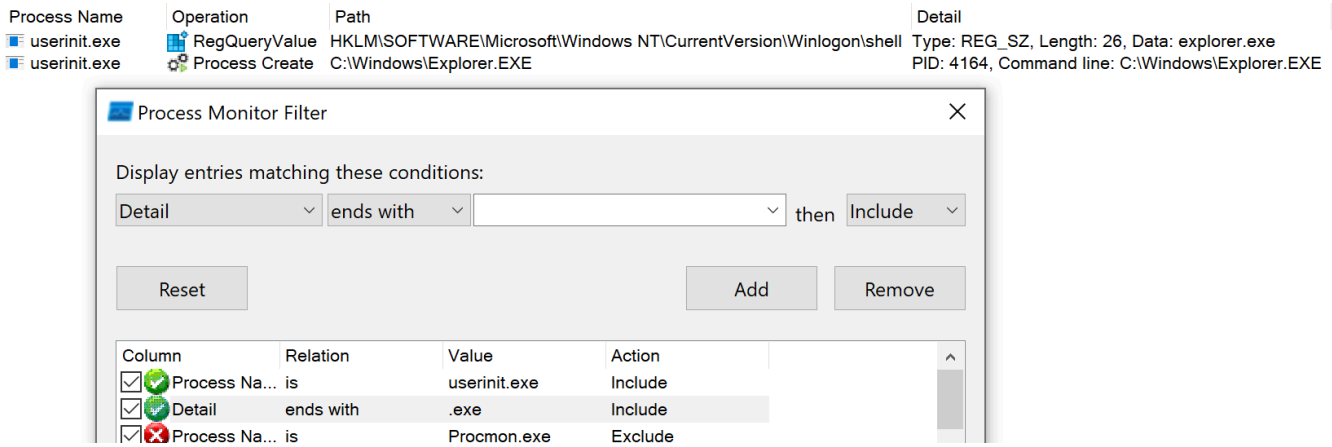


Figure 5. UserInit checks the shell value to determine which shell to open.

Creating a new key is an alternative to modifying an existing key. This requires finding a key that doesn't exist by default but is checked by a process. For example, Microsoft Office applications check for the existence of the Key "HKLM\Software\Microsoft\Office Test\Special\Perf," which doesn't exist in a standard Windows distribution. It's thought that Microsoft uses this key for testing or debugging purposes. A malicious DLL can create the key and set the default value to the path of the DLL. Once the value is set, every time Microsoft Office applications are opened, they load that DLL. This option has the advantage of not "breaking" the hijacked process, meaning a user might not notice any change on an infected machine. However, the main disadvantage of this method is that finding keys like these is more challenging than other approaches. We can use Procmon to see all the keys a process searches for that don't exist by default. But for each key, we need to check if it can be used for persistence, making finding new persistence techniques a bit more tedious.

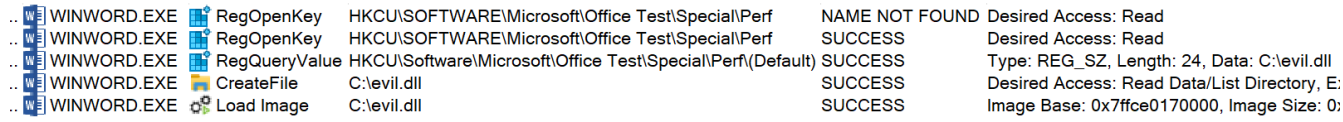


Figure 6. Word running before (first event) and after (the rest of the events) the creation of the key

Adding an item to a list (see Registry Terminology for list examples) is usually the most straightforward approach to finding a new technique. Using lists is beneficial because we don't have to replace any of the expected behavior of the process using the key. This means that a user is less likely to notice the unusual behavior, and as a result, malware using this method is less likely to be discovered. Additionally, finding new potential techniques is relatively easy with Procmon since we can filter for registry enumeration operations or multiline values.

The Forgotten Techniques

Now that we have a general understanding of how persistence in the registry can be achieved, it's time to look at specifics. The following examples are cases so rarely used that they never made it into the MITRE ATT&CK Matrix, even though some of them have been around since at least Windows XP. It's also worth noting that many of these techniques are known to AutoRuns. One final note before going into specifics: with a single exception, all of the examples listed below require elevated privileges. As a result, these techniques are somewhat more challenging to use since they need a method for the malware to gain elevated privileges before implementing their persistence.

Boot Verification Program:

As its name suggests, the Service Control Manager (SCM) manages the services on a machine. Among the SCM's responsibilities is keeping track of a configuration that describes which services need to be started during the boot process and the order in which to create them.

The SCM keeps track of the **current configuration** and the **Last Known Good (LKG) configuration**. When a change is made to the configuration, the current configuration is updated, but the LKG doesn't change. During the next boot, the SCM uses the order defined by the current configuration. If the boot succeeds, the LKG is updated as well. If the boot fails, the SCM will try starting the services a second time following the LKG instead.

By default, Winlogon notifies the SCM that a boot was successful after the first successful login. In some situations, we might prefer to define the successful boot differently (e.g., if a Windows Server needs to run a specific application, we might want to include a successful launch of the application as part of the successful boot). Microsoft allows users to define a custom boot verification program for those situations by creating the registry key "HKLM\System\CurrentControlSet\Control\BootVerificationProgram" and setting the value of ImagePath to the path of our boot verification program.

Another change that should be made when using this feature legitimately is the "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" key. Setting the ReportBootOk value to 0 in that key will prevent Winlogon from notifying the SCM about a successful boot and prevent conflicts with the boot verification program.
Malware with administrator privileges is able to modify this key and can set an arbitrary boot verification program that the SCM will start after every boot. We can see in the following image a snippet of Procmon where the SCM (the process named "services.exe") searches for the key and loads the executable set in that key.

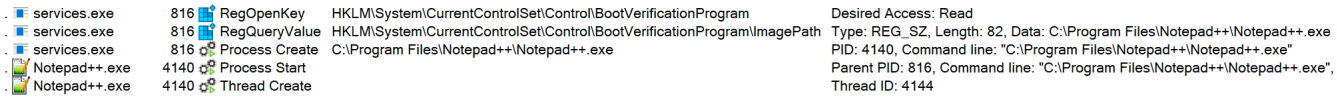


Figure 7. The SCM launching Notepad++ as the boot verification program.

Path Interception by App Path:

This method works when a process uses the ShellExecuteEx function and provides only the name of an executable and not the full path (e.g., when using the Windows run prompt to launch cmd). Windows has a set number of locations where it searches for the executable in these cases. The following keys are two locations checked:

“HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths” and “HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths.”

If either key has a sub-key with the executable name given to ShellExecuteEx, then the values of the sub-key are checked. If the value of Default is set to the path of an executable, then Windows will launch the executable from that path. This functionality can allow third-party developers to use ShellExecuteEx to launch their programs from wherever the key refers to rather than copying the programs into touchy locations like System32 to be part of the search order.

Only the path under HKLM requires administrator privileges to write. But since the path under HKCU is checked first, even if an admin sets a value under HKLM, malware without administrator privileges can still use this technique by creating a new value under HKCU. This is in some ways similar to versions of COM hijacking, where even if the CLSID is defined under HKLM, malware with low privileges can create a copy of the CLSID under HKCU and hijack the execution flow.

This method is somewhat different than the other examples here. For starters, we couldn’t find any examples of malware using this in the wild, and AutoRuns also doesn’t seem to recognize it as a persistence method. Additionally, this is the only method that can be implemented **without** elevated privileges.

Malware can use this functionality to override an existing app path so that when a user tries to launch a legitimate process with ShellExecuteEx, they end up running the malware. In the following image, we see what happens when the app path for chrome.exe has its value set to Notepad++.exe instead.

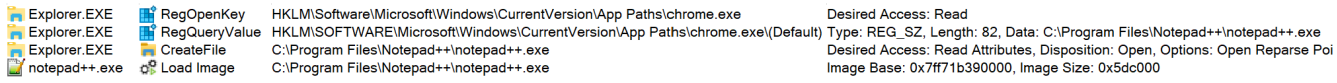


Figure 8. The explorer.exe process tries to run chrome, but the App Paths key redirects to Notepad++.

SMSS Configuration:

The Session Manager Sub-System (a.k.a. the Session Manager or smss) is the first process to launch during the boot process. It is responsible for many different tasks, including starting some other processes that run early in the boot process. To do this, it relies partly on values set in the “HKLM\SYSTEM\CurrentControlSet\Control\Session Manager” registry key. It uses several values to determine which processes to launch, including BootExecute, PlatformExecute, SetupExecute, and s0InitialCommand. Each value has its purpose and

legitimate uses. However, malware with administrator privileges can modify these keys, setting itself as the intended target and running among the first processes during boot.

It's worth noting, however, that any malware that uses this method will have to be specially written since it will be started in "no man's land" with — depending on when exactly they start — no KnownDLLs loaded into memory, no HKCU hive and even part of the HKLM hive missing. The following image shows what would happen to Notepad if it were set as the first line in the multiline string value BootExecute. Notepad starts up as the second program to run in the entire boot but then immediately crashes and closes since it's not written in such a way that it can run in these conditions.

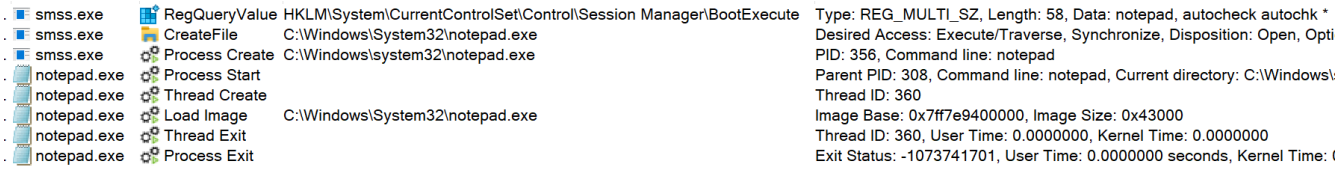


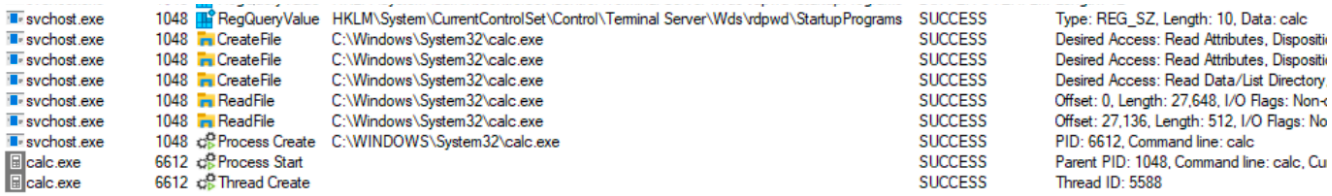
Figure 9. Smss running Notepad, followed by Notepad immediately crashing.

RDP Startup Program:

One of the convenient features of RDP in Windows is the ability to copy and paste between the host and client machines. To help facilitate this feature, Microsoft provides Windows machines with a program called rdpclip. Having rdpclip run all of the time would be a waste of resources, as it's only applicable when logged into a client with RDP. Due to this, Windows will only launch rdpclip when a user logs into the machine with RDP. When the login occurs, Windows will search for the key "HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server\Wds\rdpwd" in the registry.

If the key exists and has the StartupPrograms value set to an executable, Windows will launch the executable during login. By default, this key is set to rdpclip, which results in that process launching after an RDP login. This blog post offers some more details about rdpclip and a related vulnerability.

Malware with administrator privileges can change the StartupPrograms to itself, resulting in the malware being launched every time a user logs into the client via RDP but not during a regular login. The following snippet shows how the service responsible for running rdpclip will end up running calc instead when the key is modified.



svchost.exe	1048	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\Wds\rdpwd\StartupPrograms	SUCCESS	Type: REG_SZ, Length: 10, Data: calc
svchost.exe	1048	CreateFile	C:\Windows\System32\calc.exe	SUCCESS	Desired Access: Read Attributes, Dispositi
svchost.exe	1048	CreateFile	C:\Windows\System32\calc.exe	SUCCESS	Desired Access: Read Attributes, Dispositi
svchost.exe	1048	CreateFile	C:\Windows\System32\calc.exe	SUCCESS	Desired Access: Read Data/List Directory
svchost.exe	1048	ReadFile	C:\Windows\System32\calc.exe	SUCCESS	Offset: 0, Length: 27,648, I/O Flags: Non-
svchost.exe	1048	ReadFile	C:\Windows\System32\calc.exe	SUCCESS	Offset: 27,136, Length: 512, I/O Flags: No
svchost.exe	1048	Process Create	C:\WINDOWS\System32\calc.exe	SUCCESS	PID: 6612, Command line: calc
calc.exe	6612	Process Start		SUCCESS	Parent PID: 1048, Command line: calc, Cu
calc.exe	6612	Thread Create		SUCCESS	Thread ID: 5588

Figure 10. Service running calc instead of rdpclip.

Default Browser:

Changing the default browser might seem like an obvious idea. There is even a similar method for changing the default file association. Even though the concept seems simple, the execution is not. The key that associates a specific URL protocol with a process is: "HKCU\Software\Microsoft\Windows\Shell\Associations\UrlAssociation\<protocol>\UserChoice."

Each key needs two values: the progID and a unique hash value. The progID leads to the process that'll handle whatever URL is to be opened. The hash, however, is used for validation. It seems the hash is the result of a proprietary algorithm used by Microsoft that takes at least two inputs: the value of progID and some unique value leading to different hashes, even if the same progID is used. If malware change the progID without touching the hash, this results in an invalid hash, leading Windows to reset the default browser to Microsoft Edge.

Threat actors can try and reverse engineer the hash algorithm and implement it themselves (or try some already existing tools). Alternatively, if an attacker has gained access to the machine, they can also try and use Windows settings to set the malware as the default browser. However, this requires changing two more keys. The first is "HKLM\SOFTWARE\RegisteredApplications," where they need to add a value for their process pointing to yet another key in a location of their choice.

Figure 11. The registered application value for Wordpad defines the path for the capabilities of Wordpad.

This key represents the capabilities of the executable and, as such, must contain a sub-key called UrlAssociations to define the capability to handle URLs, with values of both HTTP and HTTPS set to the progID of their malware.

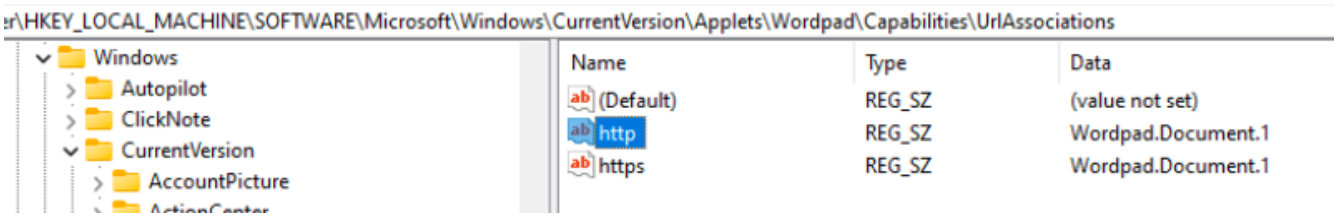


Figure 12. Added HTTP/HTTPS capabilities to Wordpad

By doing this, they can make the malware the default browser in Windows Settings.

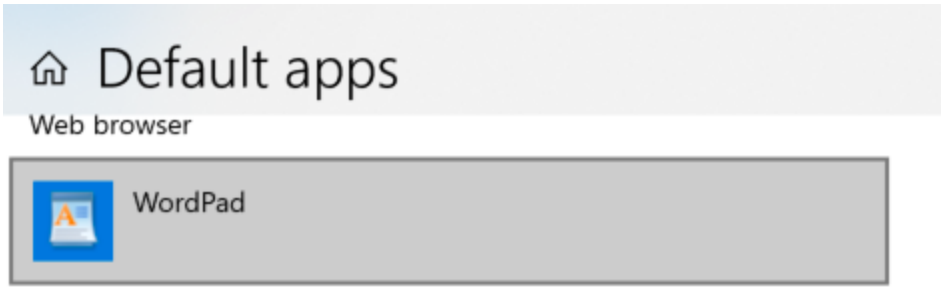


Figure 13. Added HTTP/HTTPS capabilities to Wordpad.

Logon UI Credential Providers:

Logon UI credential providers are COM objects used to collect credentials during the login process. Windows offers multiple ways to authenticate (passwords, smart cards, biometrics, etc.), and the Logon UI must provide a user with the correct prompt to request credentials. Logon UI queries all the credential providers during authentication to find the appropriate one. Once received, the user can choose a provider and supply it with credentials. Using *COM hijacking* to replace an existing credential provider, malware can abuse these mechanisms. However, another way to gain persistence here, assuming the malware is running with administrator privileges, is registering as an additional Credential Provider to be loaded by the Logon UI.

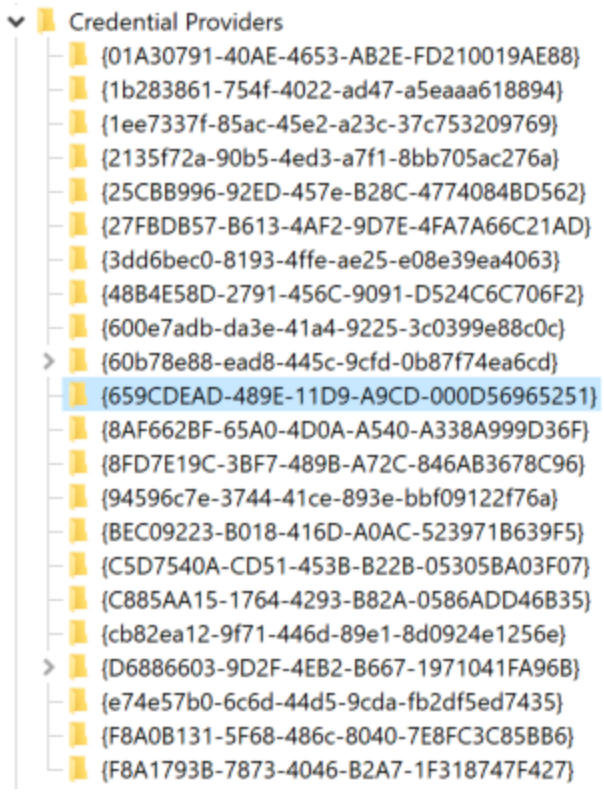


Figure 14. The BitsProxy CLSID was added as a credential provider.

Figure 15. The BitsProxy CLSID was added as a credential provider.

Mitigation

So, what can we do to protect ourselves?

In general, finding a “one size fits all” solution for such a wide variety of potential techniques is difficult. Some of the known persistence techniques also have legitimate uses, complicating things further. But things aren’t entirely hopeless here.

For starters, we can monitor events of writing values to the registry. Specifically, we should monitor for values ending with potentially dangerous file extensions such as “.exe” or “.dll” (this also works for the techniques that use COM objects since the COM servers still need to be registered under HKCR). Security solutions can perform this monitoring in many ways, such as WMI, API Hooking and Kernel Callbacks. Any event like that can be a potential persistence technique. Users can also manually check via the Startup tab in the Task Manager (or in msconfig for older Windows distributions) to see if any suspicious programs make use of the techniques checked by the Task Manager.

While prevention isn't always easy, we can assemble a database of known techniques and restrict write access to the keys that are used in those techniques. This can be done using an access control solution that allows for a granular policy-based access control for registry keys, such as CyberArk Endpoint Privilege Manager.

But provided that administrative access to a machine allows for such a broad opportunity to change system settings, the most effective way to counter the whole class of persistence techniques is to remove local admin rights from the users completely and replace them with conditional policy-based elevation for both regular users and administrators. This, again, is done best with a specialized tool such as CyberArk Endpoint Privilege Manager.

Conclusion

We only examined methods that use the registry, yet we've already observed an extensive range of potential ways to gain persistence. Unfortunately, there is a decent chance that the methods discussed here are just the tip of the iceberg when it comes to obscure persistence techniques used in the registry.

As previously mentioned, Procmon offers some functionality to help us find such methods. We tried using Procmon's Boot Logging capabilities to get all the events during the boot process on several machines, for both Windows 10 and Windows 11. For each PML file generated by Procmon, we filtered for registry events with the result "NAME NOT FOUND." This would narrow things to the potential keys that can be used for persistence only during the boot process and creating keys that don't exist by default. Much to my surprise, on all the machines I checked, even after removing the duplicates, I consistently got over 50,000 events. While the vast majority of these likely can't be abused, such a large number strongly indicates that there are still many more potential methods. For this reason, even infrequent logins from an administrator introduces a significant risk of enabling malware to gain persistence on the system. That's why local admin rights should be removed from all users in an organization.