

Windows object permissions as a backdoor

 [grzegorztworek.medium.com/windows-object-permissions-as-a-backdoor-fa33eb7857c4](https://medium.com/windows-object-permissions-as-a-backdoor-fa33eb7857c4)

Grzegorz Tworek

February 27, 2021



Grzegorz Tworek

As the typical cyberattack kill chain follows the well-known schema, the response should follow it. This is also true in the “Local privilege escalation” scenarios, and it may be quite interesting in all cases, when the system needs to support unprivileged users, but such users reach sometimes for more than they should. Of course, such cases should be impossible, but they will occur, because admins make mistakes, and different bugs exist as well. Take a look at [CVE-2020-1048](#) and the attack based on single “*Add-PrinterPort*” cmdlet [described by Alex Ionescu](#), if you need an example how easy and unexpected such attacks are. If such incident happens, it must be processed its own way, especially during an investigation, and eradication phases. Investigators should focus not only on fixing the vulnerability, but also on identifying all “remains” an attacker left with regaining superpowers again in mind. There is a huge set of possibilities, an attacker can use for such purpose, such as:

- Adding his account to Administrators group,
- Creating new administrative account,
- Creating a Scheduled Task, providing a privileged access after the eradication,
- Installing specialized Windows Service,
- Installing DLLs, being automatically loaded by highly-privileged processes,
- Replacing legitimate files with malicious ones,
- Creating WMI event filters and consumers,
- Manipulating encryption keys needed for offline access,
- Etc.

Such backdoors make the administrative access possible, even if the attacker’s privileges are fixed, and the vulnerability itself is remediated. Some tools, like Sysinternals Autoruns, allow to take a quick look into most typical places, but such investigations are not very easy as all possible backdoor locations are nowhere listed as a complete set. What investigators do? They look for anomalies, like unusual settings, unusual files, etc. Fingers crossed, but there is still one thing left. With great exploiting potential, relatively easy to be planted within seconds, persistent, and effectively allowing an attacker to regain high privileges. It is object permission, also known as Access Control Lists or ACLs. Using ACLs as backdoors requires higher initial privileges, as the only reasonable scenario is transition from user to admin. But it is still quite a lot, and such escalation should not be possible without system’s owner approval. To quickly summarize ACL-based backdoors:

Pros:

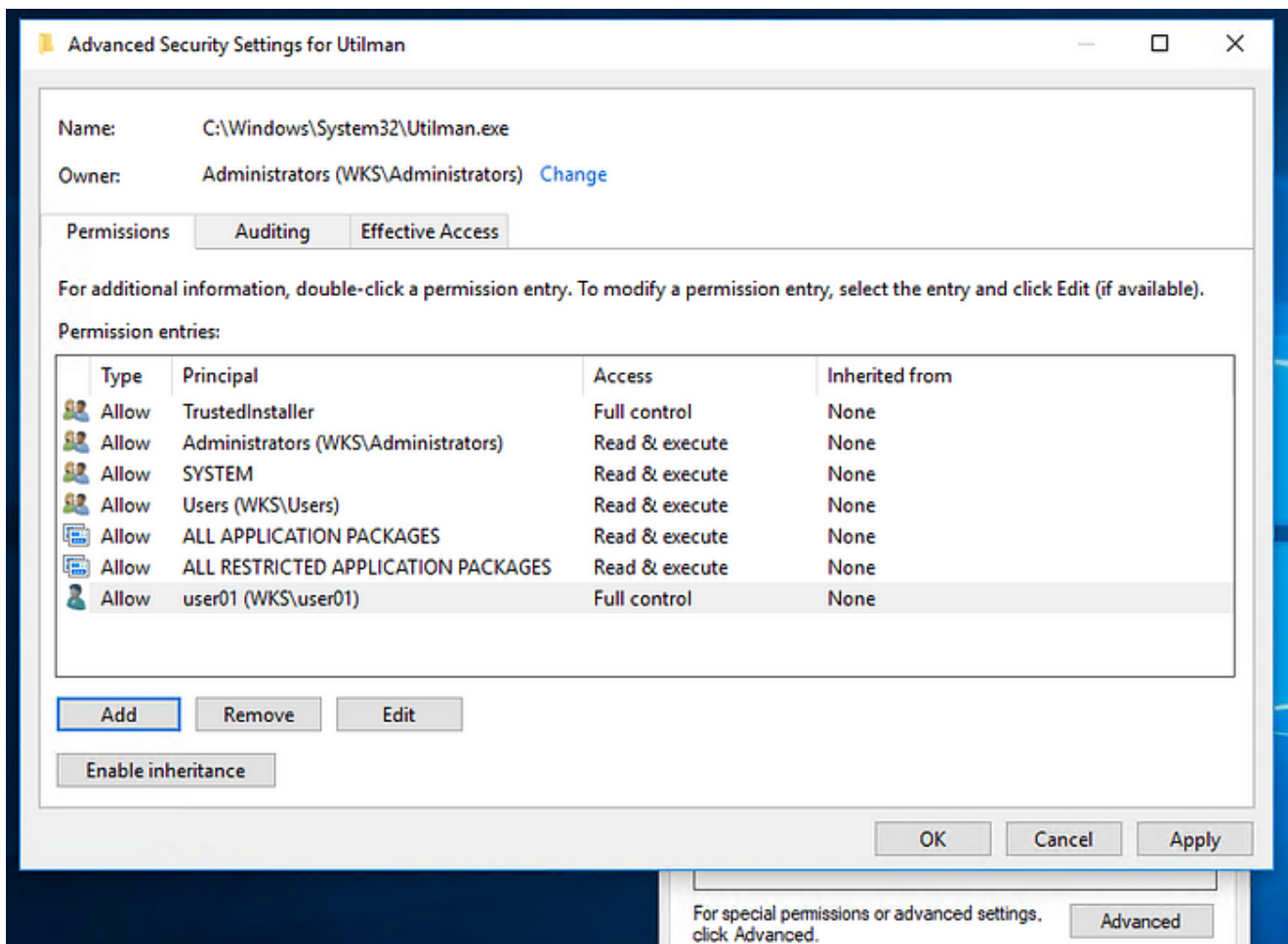
- Very quick to plant,
- No additional binaries/tools required, only built-in OS tools,
- Easy to plant remotely,
- Impossible to find, unless an admin looks specifically for them,
- Very rarely checked by scanning tools.

Cons:

Cover only “user to admin” privilege escalation scenarios.

And how such backdoor can be planted in practice? There is a quick scenario:

- User is an admin for a moment,
- User launches cmd.exe using its temporary superpowers,
- User modifies ACLs of utilman.exe, taking its ownership and adding himself to the list



Modified ACL of the Utilman.exe utility

- User removes his privileges, and behaves politely,
- Admins fix the vulnerability,

- User logs back on as regular user,
- User overwrites the original utilman.exe with cmd.exe, for example using ""
- User presses Ctrl+Alt+Del, and then Win+U and has admin access again.

Admins and investigators should stop here for a moment and honestly answer the question: do the existing incident response procedure has any chance to detect such backdoor before it's exploited by rewriting the file? If user changes the file, the detection is relatively easy with system integrity checkers such as sfc.exe. But when it comes to the file permissions — typical procedures and tools are just blind.

```

Administrator: Command Prompt
C:\>icacls c:\Windows\System32\Utilman.exe
c:\Windows\System32\Utilman.exe NT SERVICE\TrustedInstaller:(F)
                                BUILTIN\Administrators:(RX)
                                NT AUTHORITY\SYSTEM:(RX)
                                BUILTIN\Users:(RX)
                                APPLICATION PACKAGE AUTHORITY\ALL APP
                                APPLICATION PACKAGE AUTHORITY\ALL RES
                                WKS\user01:(F)

Successfully processed 1 files; Failed processing 0 files

C:\>sfc /scannow

Beginning system scan. This process will take some time.

Beginning verification phase of system scan.
Verification 100% complete.

Windows Resource Protection did not find any integrity violations.

C:\>

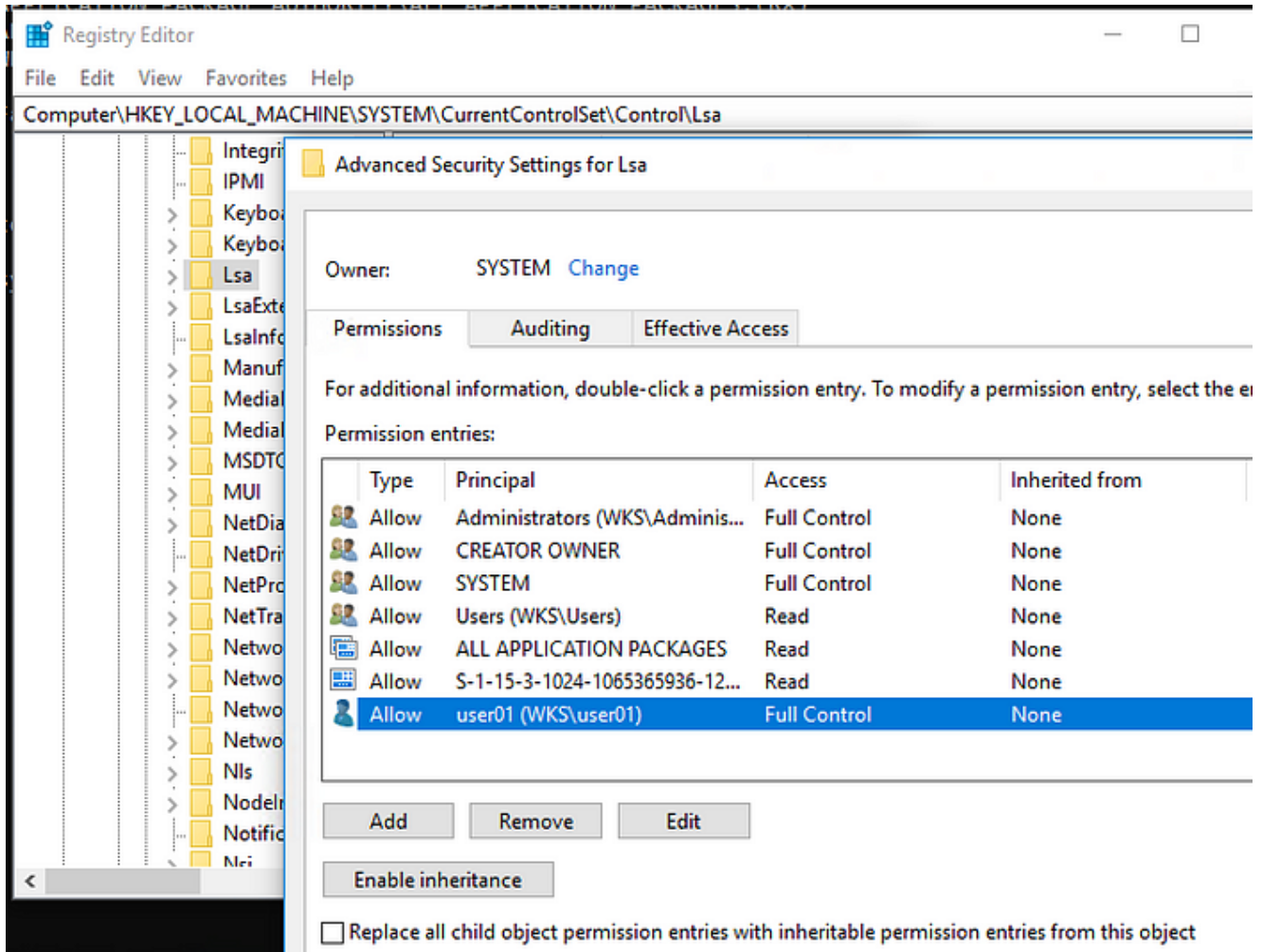
```

Utilman.exe seen as legitimate file, despite ACL modification

Utilman.exe is a very simple example, but it works perfectly. Other possible locations for such scenarios include:

- Windows Operating System files, being used by highly privileged processes. Hundreds of EXE and DLL files actually. Some of them may be exploited easily, other may require dropping own binary file, but any approach based on comprehensive list and manual check is irrational.
- Program Files, and other non-windir locations, being used by highly privileged processes. Such locations are used typically by third-party software, and they exist on each Windows based machine. It may be slightly harder to pick ideal candidate, but at the same time, every single computer may be slightly different making the detection even harder.

- Registry permissions. Less known than file permissions, which makes them even more tempting to exploit. As registry settings determine the OS behavior, changes may disrupt security in multiple different ways. For example, the REG_MULTI_SZ value "" in the Control\Lsa determines which DLL is loaded by the lsass.exe. It allows to perform multiple sneaky attacks, but at the same time is a great backdoor, as the DLL added to the list by an attacker will be loaded at every boot, executing its code in the SYSTEM context.



Modified ACL of the registry key

Windows Service permissions. Even less known, however easy to display with "". Even if the result may be a bit intimidating at the first sight, it is perfectly precise, well documented and powerful. The only enigmatic part here is the way how it is displayed, as it is "" or SDDL. Under the hood, it is just an ACL, specifying who can start or stop the service, change its configuration etc. If an attacking user provides himself a possibility of changing the configuration, he owns the system forever. When he wants to re-gain admin superpowers, he can change the existing configuration to make Service Manager launch malicious executable, instead of legitimate one.

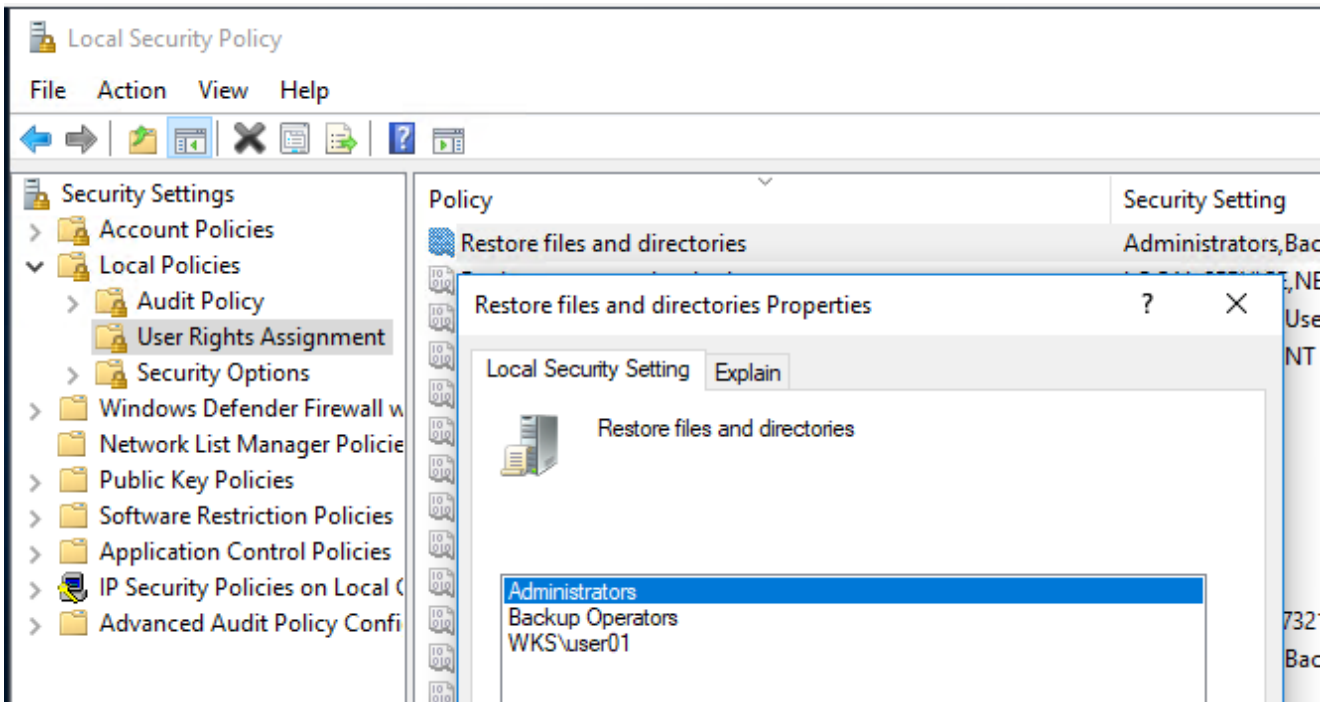
```

Administrator: Command Prompt
C:\>sc.exe sdshow spooler
D:(A;;CCLCSWLOCRRC;;;AU)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)(A;;CCLCSWRPWPDTLOCRRC;;;SY)
S:(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)
C:\>_

```

Spooler service permissions

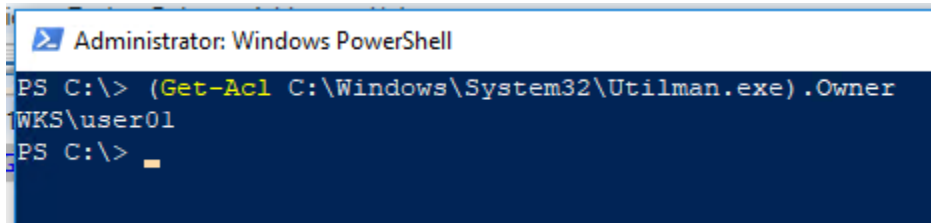
System privileges. System privileges are not object permissions, but they may be treated as permissions for the entire Operating System. Some privileges are relatively harmless, as changing the timezone or even shutting down the server will not make user the admin again, while . Especially , and may be tempting for an attacker, as both allow to gain admin privileges with couple of command lines and no third-party tools. More complicated, but still successful scenarios rely on SeAssignPrimaryTokenPrivilege, SeCreateTokenPrivilege, SeDebugPrivilege, SeLoadDriverPrivilege, SeManageVolumePrivilege and SeTcbPrivilege. Restore and TakeOwnership privileges are used to gain control over sensitive Operating System files, and the attack itself resembles the scenario with utilman.exe, as described above.



SeRestorePrivilege assigned to an unprivileged user

Active Directory. Even if out of scope (it is not a Windows object, and it cannot be called a local privilege escalation), Active Directory is mentioned here to remind, that it is vulnerable to the same type of permission-based attacks.

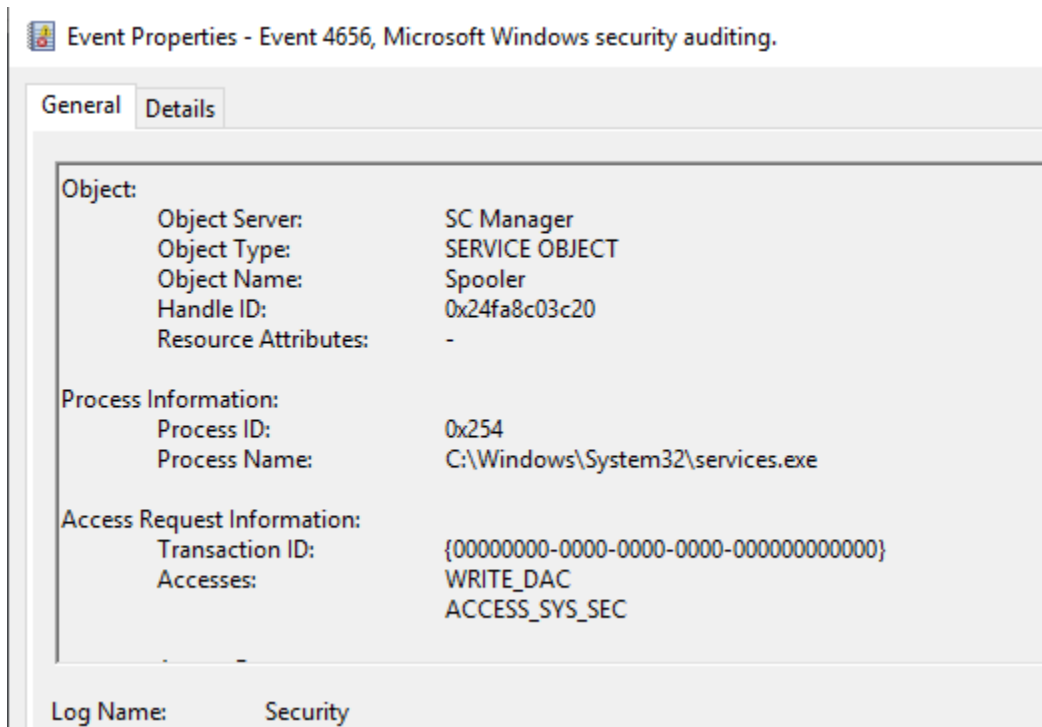
On the top of permissions allowing an attacker to manipulate objects, there is additional layer some could call “metapermissions”. It is a permission to manipulate existing permissions, or to take the ownership of the object, leading effectively to the permission manipulation as well. Sometimes “metapermissions” may be visible on the ACL (like WD, or WO in the SDDL for a service), and in other cases — not. For example, the powerful *icac/s.exe* utility says nothing about the ownership of a file, and the owner can manipulate permissions to grant himself a write access. At the same time, PowerShell displays ownership with ease.



```
Administrator: Windows PowerShell
PS C:\> (Get-Acl C:\Windows\System32\Utilman.exe).Owner
WKS\user01
PS C:\> █
```

Displaying file owner using PowerShell

From the defenders’ perspective, two questions are most important: how to get notified when important object permissions change, and how to investigate and fix permissions after the attack. Theoretically, permission change for files, registry, and services can be audited and stored in the event log.



Audit trace for permission change

In practice, collecting audit events is not enough, and two additional solutions should be implemented:

- Log centralization — to prevent log manipulation by an unwanted admin, and to prevent old events overwriting if defenders analyze the case long time after the attack;
- Analytics — to avoid manual review of millions of entries, and to provide some alerting.

If the attack already happened, the best recommendation is to re-install the machine. If machine was compromised, it will remain compromised, despite all efforts to fix it. Even if audit trail is well preserved, it should not be taken as trusted, as local administrators have multiple possibilities of manipulating it, including disabling auditing entirely during the attack. One of the most interesting methods of disabling audit relies on setting the “” Registry value. Such value makes auditing subsystem assume it is Windows installation, and not the regular operation, so auditing is pointless and just stops.

Of course, before the reinstallation, it makes sense to create an image of the compromised machine, and perform the detailed investigation. Investigators looking for permission-based attack, should focus on:

Analyzing ACLs for files and directories. As manual review is pointless (fresh Windows installation may contain over 100k files), the only reasonable way relies on automation. PowerShell cmdlet exposes SDDL property of file objects. It is possible to look for anomalies, broken inheritance, unusual owners etc. The best result may be achieved when comparing the compromised state to the healthy one, taken from baseline, similar machine or a trusted backup. returns owners of filesystem objects as well, both as dedicated property, and as a part of the SDDL string.

```
dir C:\windows\System32\*.exe | Get-Acl | Select-Object Path, SDDL | ogv
```

Path	Sddl
%SystemRoot%\System32\Usoclient.exe	O:S-1-5-80-956008885-3418522649-1831038044-1853292631-2271478464G:S-1-5-80-9!
%SystemRoot%\System32\Utilman.exe	O:S-1-5-21-3955961102-3330231408-1053378208-1002G:S-1-5-80-956008885-3418522!
%SystemRoot%\System32\VaultCmd.exe	O:S-1-5-80-956008885-3418522649-1831038044-1853292631-2271478464G:S-1-5-80-9!
%SystemRoot%\System32\wds.exe	O:S-1-5-80-956008885-3418522649-1831038044-1853292631-2271478464G:S-1-5-80-9!

Displaying owners of multiple files using PowerShell

Analyzing ACLs for Registry keys. The same approach and set of scripts used for filesystem may be used, as PowerShell exposes both filesystem, and registry as PSDrive object.

```
Administrator: Windows PowerShell
PS C:\> (Get-Acl -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\Lsa').SDDL
O:SYG:SYD:PAI (A;CIIO;KA;;;CO) (A;CI;KA;;;SY) (A;CI;KA;;;BA) (A;CI;KR;;;BU) (A;CI;KA;;;S-1-5-21-3955961
102-3330231408-1053378208-1002) (A;CI;KR;;;AC) (A;CI;KR;;;S-1-15-3-1024-1065365936-1281604716-351173
8428-1654721687-432734479-3232135806-4053264122-3456934681)
PS C:\>
```

Getting registry permissions using PowerShell

Analyzing Services ACLs. PowerShell may help with automation, but reading permissions is slightly more complex, as requires GetSecurityDescriptor method exposed by an investigated object, and does not have it. One of the workarounds relies on .

```
Select Administrator: Windows PowerShell
PS C:\> ([wmiclass]"win32_SecurityDescriptorHelper").Win32SDtoSDDL(((Get-WmiObject Win32_Service
-Filter "name='spooler'" -EnableAllPrivileges).GetSecurityDescriptor()).Descriptor).SDDL
O:SYG:SYD: (A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;AU) (A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA) (A;;CCLCSWRPWE
DTLOCRRC;;;SY) S: (AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)
PS C:\>
```

Getting service permissions with PowerShell and WMI

Analyzing system privileges. As there is no official offline way to read privileges database, and no PowerShell or even .Net implementation, privileges are the hardest area for automated analysis. The evidence should be collected while system is still alive, and it may use with parameter, or quite complex PowerShell wrapper for API function.

```
Administrator: Command Prompt
C:\>secedit /export /areas USER_RIGHTS /cfg output.txt
The task has completed successfully.
See log %windir%\security\logs\secedit.log for details in f...
output - Notepad
File Edit Format View Help
SeRestorePrivilege = user01,*S-1-5-32-544,*S-1-5-32-551
SeShutdownPrivilege = *S-1-5-32-544,*S-1-5-32-545,*S-1-5-32-551
SeTakeOwnershipPrivilege = *S-1-5-32-544
```

Dump of privileges using secedit

As described above, object permissions can be used as a very special type of backdoor. Its functionality is limited to “admin for a moment” scenarios, but such situations happen as well. In practice, it means that incident response and investigation plans should include object permissions too. This is especially important when it comes to Windows Privileges, as (using

official interfaces) these can be collected only while Operating System is still running. The most effective analysis way is the automated comparison against the healthy system, including both permissions, and the ownership of file, registry and service objects.