Offensive WMI - Interacting with Windows Registry (Part 3)

0xinfection.github.io/posts/wmi-registry-part-3

September 12, 2021

2021-09-12

This is the third instalment of the “Offensive WMI” series (the 2nd is here), and this blog will focus on interacting with the Windows Registry. A useful thing to know before we start, MITRE ATT&CK classifies querying of registry values under T1012 and its modification under T1112.

Let’s dive in.

What is Windows Registry?

In simple terms, the registry is a database that stores configuration settings and options of the operating system: the kernel, device drivers, services, SAM, user interface and third party applications all make use of the registry. This makes the registry a very attractive resource for attackers.

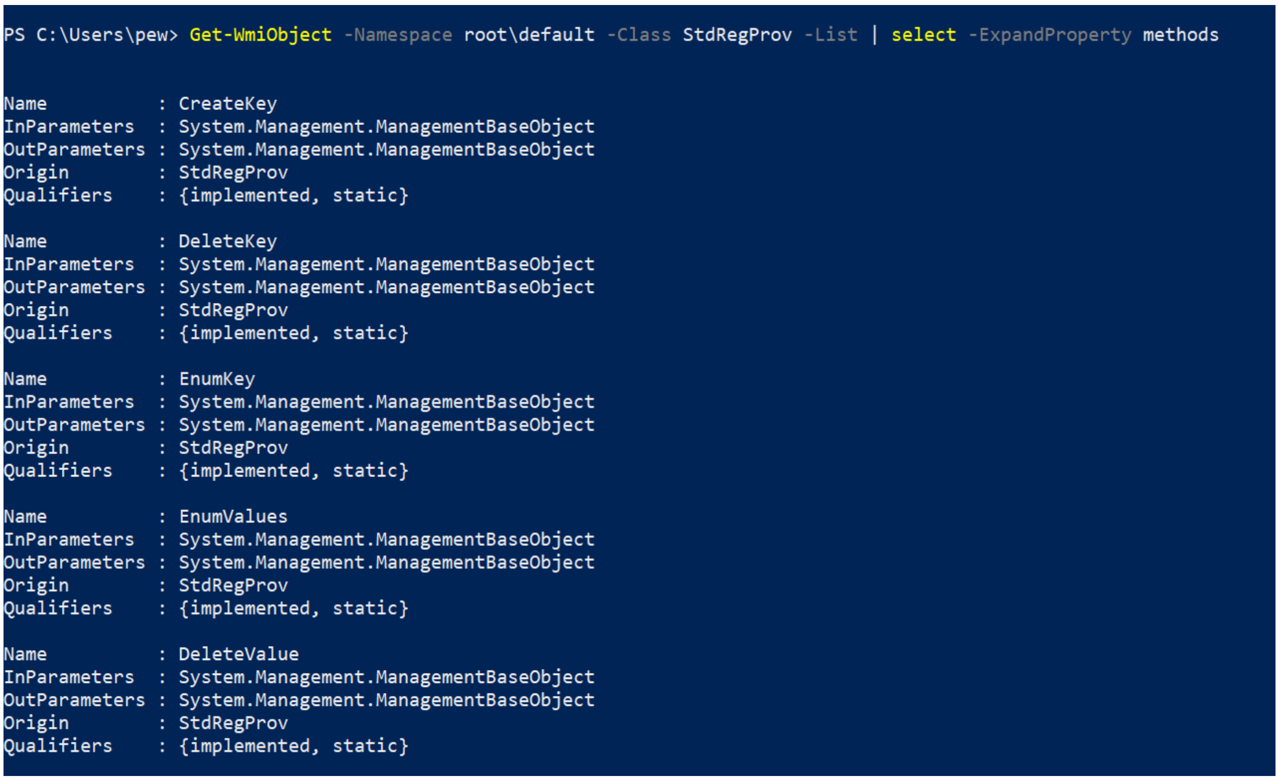
The registry consists of sections known as *hives*, e.g. `HKEY_LOCAL_MACHINE`, `HKEY_CURRENT_USER`, etc. Upon inspection of the registry in `regedit.exe`, they appear to be arranged in a similar fashion to a filesystem. Each hive has a number of keys. The keys can have multiple subkeys. A key or subkey acts as a store for values. A registry item consists of a name and value pair.

Registry & WMI

WMI provides a class called `StdRegProv` for interacting with the Windows Registry. With this in hand, we can do a variety of things – including retrieval, creation, deletion and modification of keys and values. An important point to note here is that we need to use the `root\DEFAULT` namespace for working with the registry.

Let's start by exploring what methods are available to us:

```
Get-WmiObject -Namespace root\default -Class StdRegProv -List | select -
ExpandProperty methods
```



```
PS C:\Users\pew> Get-WmiObject -Namespace root\default -Class StdRegProv -List | select -ExpandProperty methods


Name          : CreateKey
InParameters  : System.Management.ManagementBaseObject
OutParameters : System.Management.ManagementBaseObject
Origin        : StdRegProv
Qualifiers    : {implemented, static}

Name          : DeleteKey
InParameters  : System.Management.ManagementBaseObject
OutParameters : System.Management.ManagementBaseObject
Origin        : StdRegProv
Qualifiers    : {implemented, static}

Name          : EnumKey
InParameters  : System.Management.ManagementBaseObject
OutParameters : System.Management.ManagementBaseObject
Origin        : StdRegProv
Qualifiers    : {implemented, static}

Name          : EnumValues
InParameters  : System.Management.ManagementBaseObject
OutParameters : System.Management.ManagementBaseObject
Origin        : StdRegProv
Qualifiers    : {implemented, static}

Name          : DeleteValue
InParameters  : System.Management.ManagementBaseObject
OutParameters : System.Management.ManagementBaseObject
Origin        : StdRegProv
Qualifiers    : {implemented, static}
```

From the output above, we can see methods like `CreateKey`, `DeleteKey`, `EnumKey`, `EnumValues`, `DeleteValues`, etc, for interacting with the Registry. Interesting.

Two important things to know before jumping in:

1. First, WMI uses constant numeric values to identify different hives in the registry. The table below lists the constants for accessing registry hives:

Variable	Value	Hive
$HKCR	2147483648	HKEY_CLASSES_ROOT
$HKCU	2147483649	HKEY_CURRENT_USER
$HKLM	2147483650	HKEY_LOCAL_MACHINE
$HKUS	2147483651	HKEY_USERS
$HKCC	2147483653	HKEY_CURRENT_CONFIG

2. And secondly, the registry has different data types, and each data type can be accessed using a particular method in WMI. The table below maps common data types to their methods:

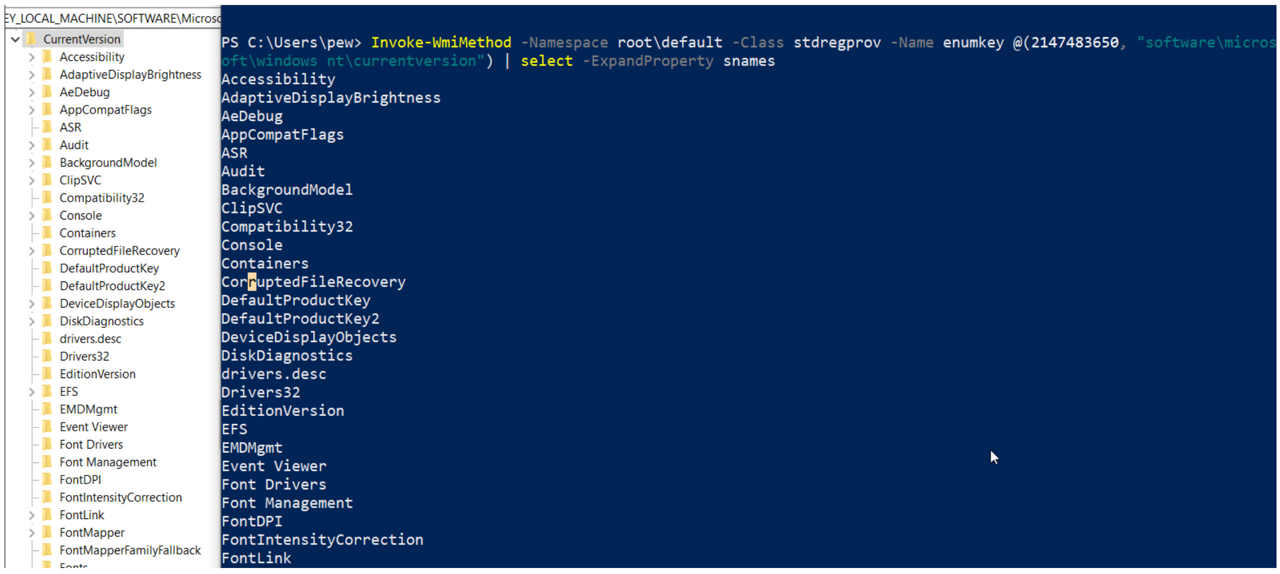
Method	Data Type	Type Value	Function
`GetStringValue`	REG_SZ	1	Returns a string.
`GetExpandedStringValue`	REG_EXPAND_SZ	2	Returns expanded references to env variables.
`GetBinaryValue`	REG_BINARY	3	Returns array of bytes.
`GetDWORDValue`	REG_DWORD	4	Returns a 32-bit number.
`GetMultiStringValue`	REG_MULTI_SZ	7	Returns multiple string values.
`GetQWORDValue`	REG_QWORD	11	Returns a 64-bit number.

Querying the registry

Enumerating keys

Now that we know the constants, let's try to enumerate the available subkeys under a well-known registry path `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion`. Putting together what we know so far, we can use this command to get all keys under the registry item:

```
Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name EnumKey
@(2147483650, "software\microsoft\windows nt\currentversion") | select -
ExpandProperty snames
```



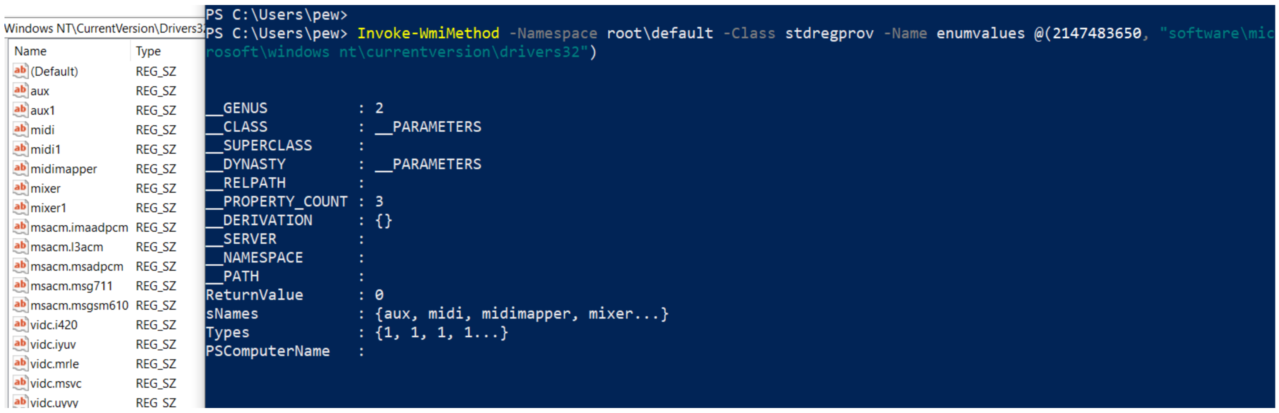
```
PS C:\Users\pew> Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name enumkey @(2147483650, "software\micros
oft\windows nt\currentversion") | select -ExpandProperty snames
Accessibility
AdaptiveDisplayBrightness
AeDebug
AppCompatFlags
ASR
Audit
BackgroundModel
ClipSVC
Compatibility32
Console
Containers
CorruptedFileRecovery
DefaultProductKey
DefaultProductKey2
DeviceDisplayObjects
DiskDiagnostics
drivers.desc
Drivers32
EditionVersion
EFS
EMDMgmt
Event Viewer
Font Drivers
Font Management
FontDPI
FontIntensityCorrection
FontLink
```

> **NOTE**: The same can be done with upper hierarchical registry paths as well. If you don't know the absolute path, you can explore the registry by simply replacing the path in the command above.
>
> e.g. – If you replace the path `software\microsoft\windows nt\currentversion\schedule` in the above command with just `software`, then the output will list all subkeys under the `HKEY_LOCAL_MACHINE\Software`. This is helpful when exploring unknown nested items in a registry.

Enumerating values

Now that we know how to list the keys available under the registry item, lets enumerate the values under the `Drivers32` key:

```
Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name enumvalues
@(2147483650, "software\microsoft\windows nt\currentversion\drivers32")
```



Windows NT\CurrentVersion\Drivers3

Name	Type
(Default)	REG_SZ
aux	REG_SZ
aux1	REG_SZ
midi	REG_SZ
midi1	REG_SZ
midimapper	REG_SZ
mixer	REG_SZ
mixer1	REG_SZ
msacm.imaadpcm	REG_SZ
msacm.l3acm	REG_SZ
msacm.msadpcm	REG_SZ
msacm.msg711	REG_SZ
msacm.msgsm610	REG_SZ
vidc.i420	REG_SZ
vidc.iyuv	REG_SZ
vidc.mrle	REG_SZ
vidc.msvc	REG_SZ
vidc.uyvy	REG_SZ

```
PS C:\Users\pew>
PS C:\Users\pew> Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name enumvalues @(2147483650, "software\mic
rosoft\windows nt\currentversion\drivers32")


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 3
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
sNames           : {aux, midi, midimapper, mixer...}
Types            : {1, 1, 1, 1...}
PSComputerName   :
```

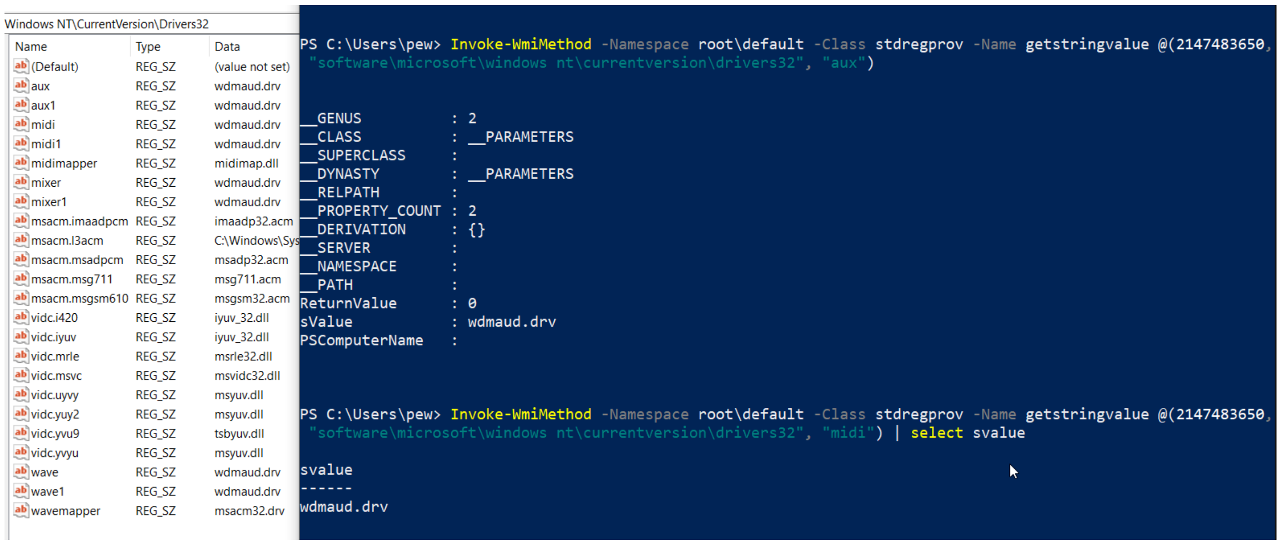
As we can see, the output contains the subkey names under `sNames` and the associated data type under `Types` property. Of course, we can use Powershell's `select -ExpandProperty` switch to have an extended view of the values of the properties returned in the output.

Reading values

Now let us try to read the values of the subkeys. For our example, we'll be reading the values of the `Drivers32` subkey (which defines the Windows NT DLLs for applications). Several malware variants have been observed making use of this key (see Riern Trojan Family) in the past.

The following command reads the value of the subkeys `aux` and `midi` under the `Drivers32` key. Please note that the method name passed to the cmdlet (via the `-Name` switch) will vary depending upon the registry data type (see the datatype table above).

```
Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name GetStringValue
@(2147483650, "software\microsoft\windows nt\currentversion\drivers32", "aux")
Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name GetStringValue
@(2147483650, "software\microsoft\windows nt\currentversion\drivers32", "midi") |
select svalue
```


Windows NT\CurrentVersion\Drivers32

Name	Type	Data
(Default)	REG_SZ	(value not set)
aux	REG_SZ	wdmaud.drv
aux1	REG_SZ	wdmaud.drv
midi	REG_SZ	wdmaud.drv
midi1	REG_SZ	wdmaud.drv
midimapper	REG_SZ	midimap.dll
mixer	REG_SZ	wdmaud.drv
mixer1	REG_SZ	wdmaud.drv
msacm.imaadpcm	REG_SZ	imaadp32.acm
msacm.l3acm	REG_SZ	C:\Windows\Sys
msacm.msadpcm	REG_SZ	msadp32.acm
msacm.msg711	REG_SZ	msg711.acm
msacm.msgsm610	REG_SZ	msgsm32.acm
vidc.i420	REG_SZ	iyuv_32.dll
vidc.iyuv	REG_SZ	iyuv_32.dll
vidc.mrle	REG_SZ	msrle32.dll
vidc.msvc	REG_SZ	msvidc32.dll
vidc.uyvy	REG_SZ	msyuv.dll
vidc.yuy2	REG_SZ	msyuv.dll
vidc.yvu9	REG_SZ	tsbyuv.dll
vidc.yvyu	REG_SZ	msyuv.dll
wave	REG_SZ	wdmaud.drv
wave1	REG_SZ	wdmaud.drv
wavemapper	REG_SZ	msacm32.drv

```
PS C:\Users\pew> Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name getstringvalue @(2147483650,
 "software\microsoft\windows nt\currentversion\drivers32", "aux")


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 2
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
sValue           : wdmaud.drv
PSComputerName   :



PS C:\Users\pew> Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name getstringvalue @(2147483650,
 "software\microsoft\windows nt\currentversion\drivers32", "midi") | select svalue

svalue
------
wdmaud.drv
```

> **TIP**: Here is a good cheatsheet of juicy locations in registry that can be useful for an attacker. You might want to try exploring them. :)

Modifying the registry

We now know about reading key and value pairs from the registry using WMI. These didn't require administrative privileges so far, however – creation, deletion and updating the keys and values *may* require elevated privileges.

Let's try to create new keys and subkeys. But before that, we need to check whether we have access to a specific registry item. Once again there are constants defining the access levels to the keys. The following table summarizes the permissions with associated constants:

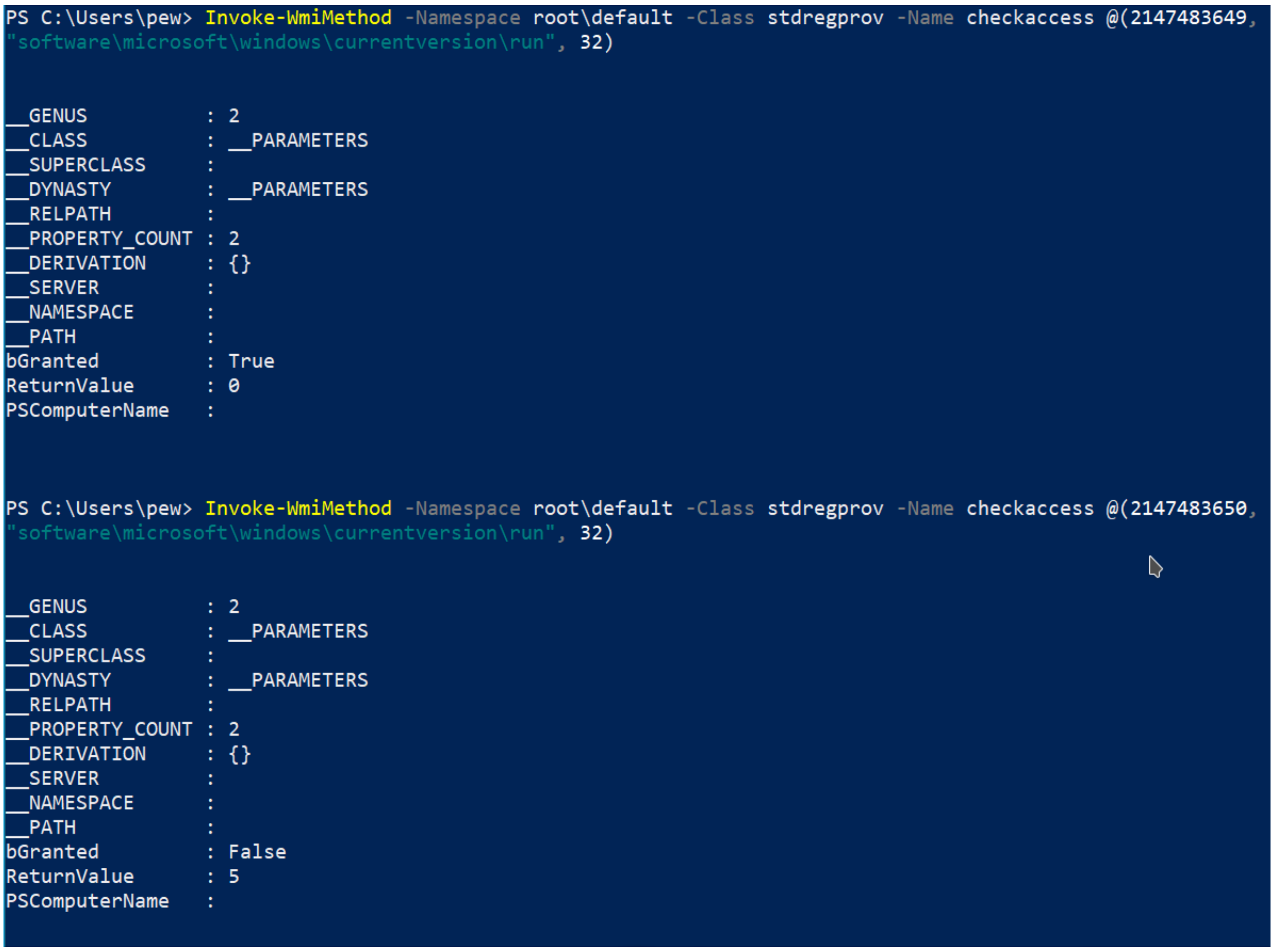
Method	Value	Function
`KEY_QUERY_VALUE`	1	Query the values of a registry key
`KEY_SET_VALUE`	2	Create, delete, or set a registry value
`KEY_CREATE_SUB_KEY`	4	Create a subkey of a registry key
`KEY_ENUMERATE_SUB_KEYS`	8	Enumerate the subkeys of a registry key
`KEY_NOTIFY`	16	Change notifications for a registry key or for subkeys of a registry key
`KEY_CREATE`	32	Create a registry key
`DELETE`	65536	Delete a registry key
`READ_CONTROL`	131072	Combines the STANDARD_RIGHTS_READ, KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS and KEY_NOTIFY values

Method	Value	Function
`WRITE_DAC`	262144	Modify the DACL in the object's security descriptor
`WRITE_OWNER`	524288	Change the owner in the object's security descriptor

Checking permissions of a key

For our example, we'll pick the `Run` key under the hive `HKEY_CURRENT_USER` first, then the `HKEY_LOCAL_MACHINE`. Here's how to do it:

```
Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name CheckAccess
@(2147483649, "software\microsoft\windows\currentversion\run", 32)
Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name CheckAccess
@(2147483650, "software\microsoft\windows\currentversion\run", 32)
```



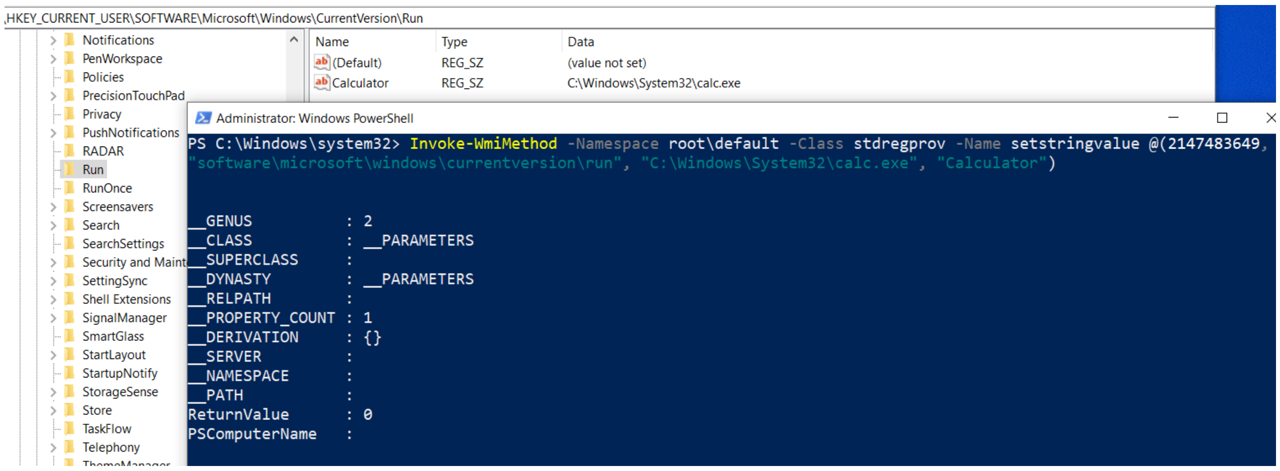
```
PS C:\Users\pew> Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name checkaccess @(2147483649,
"software\microsoft\windows\currentversion\run", 32)


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 2
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
bGranted         : True
ReturnValue      : 0
PSComputerName   :


PS C:\Users\pew> Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name checkaccess @(2147483650,
"software\microsoft\windows\currentversion\run", 32)


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 2
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
bGranted         : False
ReturnValue      : 5
PSComputerName   :
```

The `bGranted` property in the output tells us whether we have access to the specific item in the registry. From the above example, we can clearly see that our user currently has access to the `Run` key under `HKEY_CURRENT_USER` but not `HKEY_LOCAL_MACHINE`.

Creating registry entries

Now that we know that we have write access to the registry key `Run` under `HKEY_CURRENT_USER`, we'll add our favourite calculator app to the registry item. This will cause a calculator to pop up every time the system boots up, a very common technique seen in malwares to gain persistence.

```
Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name SetStringValue
@(2147483649, "software\microsoft\windows\currentversion\run",
"C:\Windows\System32\calc.exe", "Calculator")
```



```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
Calculator	REG_SZ	C:\Windows\System32\calc.exe

```
PS C:\Windows\system32> Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name setstringvalue @(2147483649,
"software\microsoft\windows\currentversion\run", "C:\Windows\System32\calc.exe", "Calculator")


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
PSComputerName   :
```

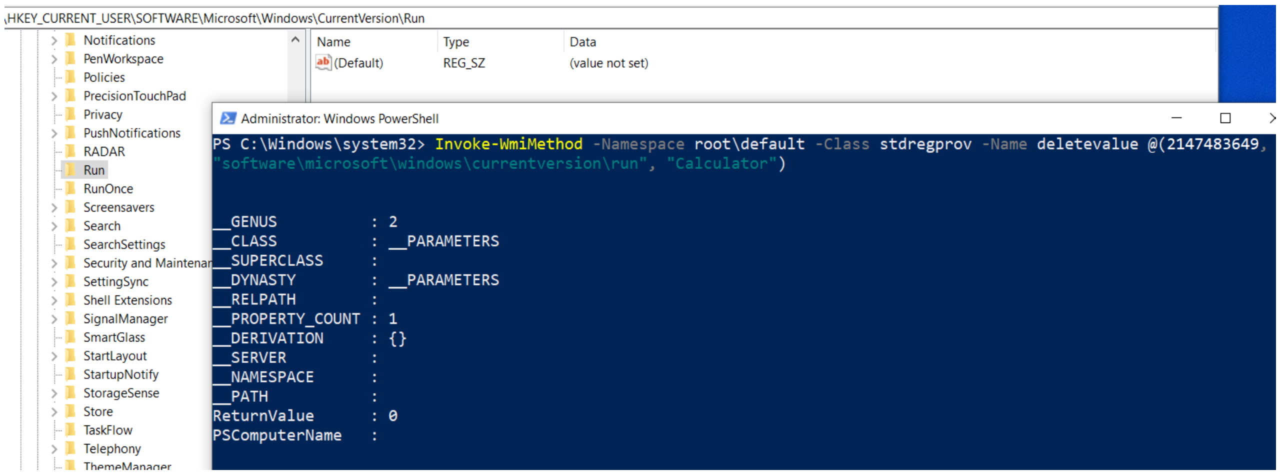
Boom, our calculator app has achieved persistence. :D

> **NOTE**: An existing subkey under a registry key can also be updated using the same above.

Deleting registry entries

To delete a registry subkey we don't need the value:

```
Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name DeleteValue
@(2147483649, "software\microsoft\windows\currentversion\run", "Calculator")
```



```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

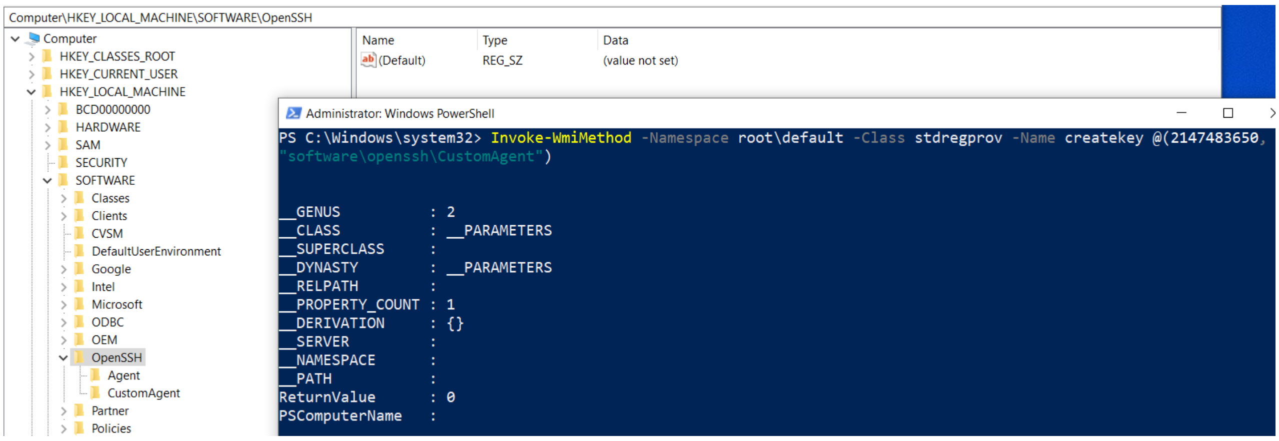
Name	Type	Data
(Default)	REG_SZ	(value not set)

```
PS C:\Windows\system32> Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name deletevalue @(2147483649,
"software\microsoft\windows\currentversion\run", "Calculator")


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
PSComputerName   :
```

Creating new keys

In few cases, we *might* need to create keys under the main tree hierarchy. Let's say we want to create a new key called `CustomAgent` under the `HKEY_LOCAL_MACHINE\Software\OpenSSH` registry item. The process looks extremely simple:

```
Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name CreateKey
@(2147483650, "software\openssh\CustomAgent")
```



```
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\OpenSSH
```

```
Computer
  HKEY_CLASSES_ROOT
  HKEY_CURRENT_USER
  HKEY_LOCAL_MACHINE
    BCD00000000
    HARDWARE
    SAM
    SECURITY
    SOFTWARE
      Classes
      Clients
      CVSM
      DefaultUserEnvironment
      Google
      Intel
      Microsoft
      ODBC
      OEM
      OpenSSH
        Agent
        CustomAgent
      Partner
      Policies
```

Name	Type	Data
(Default)	REG_SZ	(value not set)

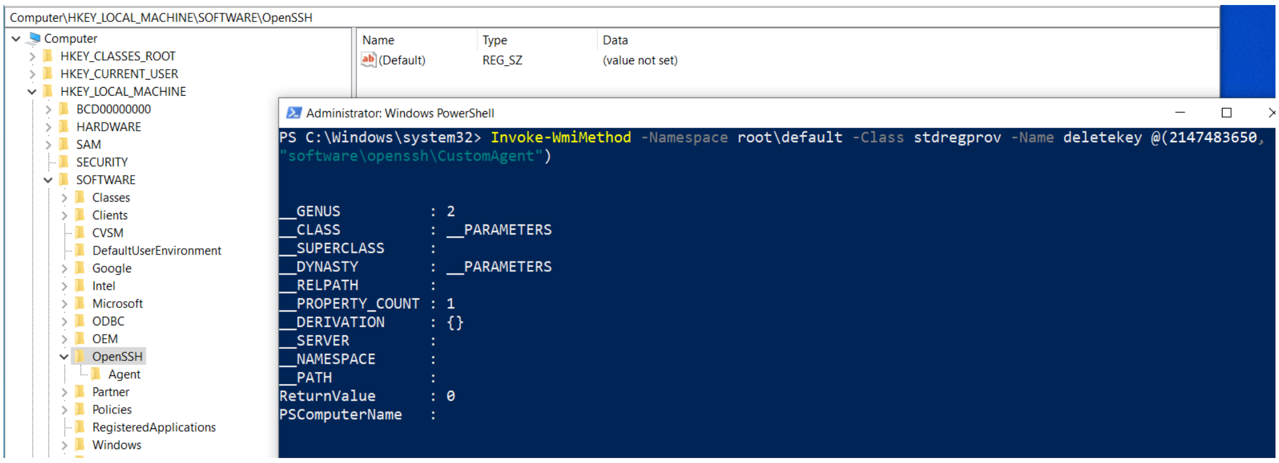
Administrator: Windows PowerShell

```
PS C:\Windows\system32> Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name createkey @(2147483650,
"software\openssh\CustomAgent")


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
PSComputerName   :
```

Deleting keys

Deleting the key is equally simple:

```
Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name DeleteKey
@(2147483650, "software\openssh\CustomAgent")
```



```
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\OpenSSH
```

```
Computer
  HKEY_CLASSES_ROOT
  HKEY_CURRENT_USER
  HKEY_LOCAL_MACHINE
    BCD00000000
    HARDWARE
    SAM
    SECURITY
    SOFTWARE
      Classes
      Clients
      CVSM
      DefaultUserEnvironment
      Google
      Intel
      Microsoft
      ODBC
      OEM
      OpenSSH
        Agent
      Partner
      Policies
      RegisteredApplications
      Windows
```

Name	Type	Data
(Default)	REG_SZ	(value not set)

Administrator: Windows PowerShell

```
PS C:\Windows\system32> Invoke-WmiMethod -Namespace root\default -Class stdregprov -Name deletekey @(2147483650,
"software\openssh\CustomAgent")


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
PSComputerName   :
```

Tools of the Trade

Conclusion

The registry is a treasure trove for attackers when it comes to gathering useful data. In addition, the registry can also be used to store payloads, serving as an ideal fileless attack vector and persistence mechanism. In a later part of the series, we'll take a look at how to create our entire C2 infra using just WMI and the registry. Now that we are done with the basics, in our next blog we'll start with basic reconnaissance with WMI.

That's it for now friend. Cheerio! \o/