Beyond good ol' Run key, Part 4

Last three articles about various startup/autostart methods covered a lot of different well- and less-known techniques for 'staying alive'. Many of them are actively used by malware and 'normal' software; some are just ideas that are worth describing because… luck favors a prepared mind. If you haven't read them previously, you can do so by visiting these links: Part 1, Part 2, Part 3.

In today's post I will cover some more techniques including hijacking of various debuggers and some more obscure ways of 'survival'. I think this is probably the lamest part of the series so far, because the techniques are old-school and amateurish, but luckily it is not the last one, so stay tuned for Part 5 😳

Hijacking debuggers

The list of debuggers one can replace on the system is as follows:

- Standalone Debugger (32- and 64- bit)

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\
CurrentVersion\AeDebug]
Debugger = PATH

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\
Windows NT\CurrentVersion\AeDebug]
Debugger = PATH
```

- .NET Debugger (32- and 64- bit)

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework]
DbgManagedDebugger = PATH

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\.NETFramework]
DbgManagedDebugger = PATH
```

- Script Debugger

```
[HKEY_CLASSES_ROOT\CLSID\
{834128A2-51F4-11D0-8F20-00805F2CD064}\LocalServer32]
@=PATH
```

Registering itself as a Script Debugger

The Windows Script Debugger (WSD) is a standalone tool that one can use to actively debug their scripts (e.g. vbs). Once installed, a developer can run one of the following commands:

- `cscript /x script.vbs`

or

- `wscript /x script.vbs`

to debug the script.

The name of the executable that is used as a debugger is stored inside the following key

- `[HKEY_CLASSES_ROOT\CLSID\{834128A2-51F4-11D0-8F20-00805F2CD064}\LocalServer32]`

and on a system where the WSD is installed may look like this:

- `[HKEY_CLASSES_ROOT\CLSID\{834128A2-51F4-11D0-8F20-00805F2CD064}\LocalServer32]`

```
@="C:\\Program Files\\Microsoft Script Debugger\\msscrdbg.exe"
```

One could replace the script debugger path to lead its own executable anytime the debugger is launched and this way making it kinda persistent on the system. Kinda, as the .exe will be executed only on rare occasion when the debugger is actually being installed (developer's or power user's machine).

Other issue is that the launching of the script debugger takes more steps than just looking up the value n the registry and launching the appropriate application.

When VBScript tries to find the debugger it talks to few COM components first (e.g. Process Debug Manager) so in order to make it work, one would need to also register these COM components (if you want to know more details, install WSD and see registry changes associated with the installation).

Hijacking Process Debug Manager

The alternative persistence mechanism could hijack one of these COM components that VBScript 'talks to' and replace its server path to point to a malicious file. The DLL does not even need to implement any COM functionality and it's enough for it to be a simple, loadable library. The Process Debug Manager that I mentioned earlier could do the trick here. Its CLSID's value on a system where WSD is installed is shown below:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\
{78A51822-51F4-11D0-8F20-00805F2CD064}\InprocServer32]
@="C:\\WINDOWS\\system32\\pdm.dll"
```

Pointing InprocServer32 to a malicious DLL would load anytime VBScript (or any other module) is 'consulting' ProcessDebugManager.

ServiceDll Hijack

Many entries under

- `[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services]`

can be hijacked by swapping their ServiceDll parameter to point to a malicious entry. e.g. the Remote Access Service registry entry

- `[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\RemoteAccess\Parameters]`

normally points to

- `%SystemRoot%\System32\mprdim.dll`

but it can be changed to point to a malicious component. There are many default services that could be hijacked this way.

Mapi32 Stub Library

Older versions of Outlook 2007 allowed to add extra functionality to Outlook by means of installing a custom version of mapi32.dll in the system directory as explained in this article.

The relevant Mapi32.dll Stub Registry Settings are provided in the Registry in the following location:

- `[HKEY_LOCAL_MACHINE\Software\Clients\Mail::(default)]`
  - `DLLPath`

    Full path to the Simple MAPI provider DLL.
  - `DLLPathEx`

    Full path to the MAPI provider DLL. Provider DLLs that support both Simple MAPI and MAPI must have both keys set.

Obviously, this mechanism is a perfect target for abuse.

Hijacking Client executables

The Registering Programs with Client Types article from Microsoft explains on 'how to register a program in the Windows registry as one of the following client types: browser, email, media playback, instant messaging, or virtual machine for Java.'. Looking at the registry entries associated with these registration we can find the following key:

- `[HKEY_LOCAL_MACHINE\Software\Clients\]`

Many applications listed under this key can be hijacked e.g. Mail program contain keys that point to executables::

- `[HKEY_LOCAL_MACHINE\SOFTWARE\Clients\Mail\Windows Mail\InstallInfo`
  - `HideIconsCommand`
  - `ReinstallCommand`
  - `ShowIconsCommand`

Windows 2000 Welcome

Installation of Windows 2000 always ends up with the "Getting Started with Windows 2000" window shown on the screen after the system restarts. User has an option to disable it, but the box is ticked ON by default.

The window shows up as a result of welcome.exe being executed from the following location:

- `C:\WINNT\Welcome.exe`

The flag that determines whether the welcome.exe is executed or not is stored in the following Registry location:

- `[HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\`

```
tips\Show: 0x00000001]
```

Replacing *welcome.exe* and ensuring the value of *Show* is equal to 1 will make the *C:\WINNT\Welcome.exe* execute every time system starts.

Well, not quite.

If the file is replaced, it will be 'magically' restored from the following location:

- `c:\WINNT\system32\dllcache\welcome.exe`

So, the malware needs to be copied into 2 locations, and… the Windows File Protection needs to be disabled as well 🙄

Thanks for reading and see you in the Part 5.

This entry was posted in **Anti-Forensics**, **Autostart (Persistence)**, **Compromise Detection**, **Forensic Analysis**, **Malware Analysis** by **adam**. Bookmark the **permalink**.