Abusing Windows Telemetry for Persistence

By Christopher Paschen in Application Security Assessment, Penetration Testing, Research, Security Testing & Analysis

Today we're going to talk about a persistence method that takes advantage of some of the wonderful telemetry that Microsoft has included in Windows versions for the last decade. The process outlined here affects Windows machines from 2008R2/Windows 7 through 2019/Windows 10.

As of this posting, this persistence technique requires local admin rights to install (requires the ability to write to HKLM) and is not visible in autoruns.

TLDR:

If you don't care about the how and why, here is what you need to know:

- Ensure your test/target machine has an active network connection
- Add a key of any name to **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\TelemetryController**
- Inside this new key, create a *Reg_SZ* value "Command" and set its data value to the **.exe** file you would like started
- Create DWORD keys for Maintenance, Nightly, Oobe, and set them all to one (only Nightly is required to be run once every 24 hours)
- Enjoy your persistence! It should run periodically from a Windows scheduled task
- You can test with **schtasks /run /tn "\Microsoft\Windows\Application Experience\Microsoft Compatibility Appraiser"** or by manually starting the task in the task scheduler gui.

As a visual example, these registry additions will start notepad.exe as system when the scheduled task is run

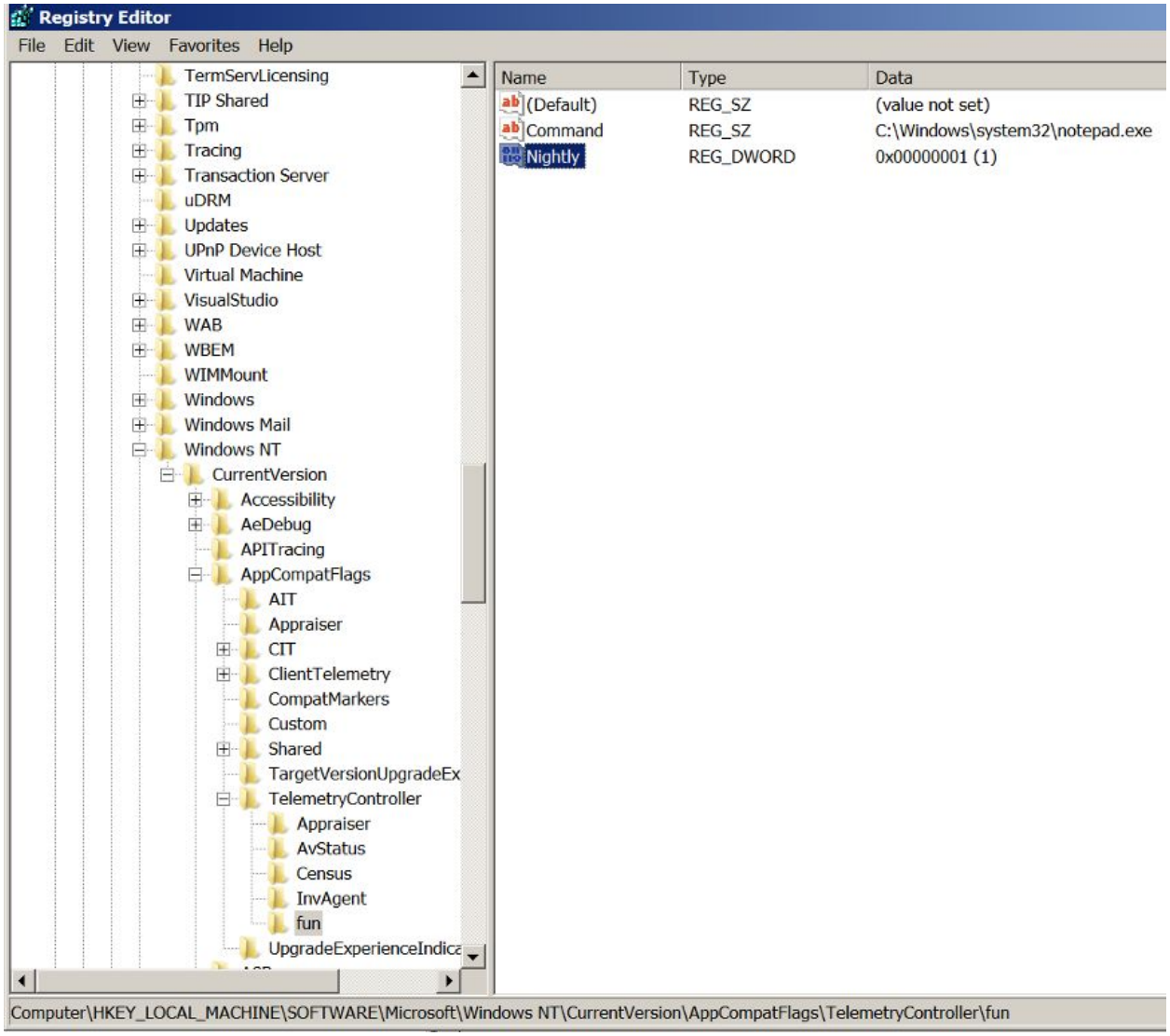


Figure 1 – Persistence Configured in RegEdit.exe

How and Why

So, let's dive into what's happening and why this is kicking your task off at the system level. First, **c:\windows\system32\CompatTelRunner.exe** appears to be a binary designed to run a variety of telemetry tasks. It in and of itself, the binary does not collect much data. CompatTelRunner appears to check some system statistics, ensuring a network is connected and then running a variety of commands to perform the actual telemetry collections. Think of this as a telemetry manager.

It appears this binary was created to be easily extensible, and to that end, it relies on the registry to instruct on which commands to run. The problem is, it will run any arbitrary command without restriction of location or type. At first glance, this may appear to be a serious security problem, but I don't think that's the case. If you have already gained administrative access to a

system, as is required to execute this process, then you already have a wide variety of options to exploit.

When **CompatTelRunner.exe** runs (current version as of May 2020), it first checks that a few conditions pass before continuing its telemetry quest.

One of these conditions must be met:

- System is Windows 10/Server 2019
- System is a client version of Windows
- **HKEY_LOCAL_MACHINE \Software\Microsoft\Windows\CurrentVersion\Policies\DataCollection\Commercial DataOptIn** is a DWORD not equal to zero

Interestingly, these checks were added at some point after the release of server 2016. Prior to the update of **CompatTelRunner.exe**, these checks were not performed, and this executable would run the commands in the registry key regardless of Windows version.

After this check (or lack thereof), the presence or absence of command line parameters will decide what run mode the program is operating in. There are three run modes corresponding to a few conditionals.

If command line parameters that specify a DLL/function are provided, **CompatTelRunner.exe** validates them against an approved list. This causes **CompatTelRunner.exe** to start the DLL provider and exit. If a DLL/function name is not provided, the program continues on to identifying the run mode.

Run mode two (OOBE) is entered if **HKEY_LOCAL_MACHINE \SOFTWARE\Microsoft\WindowsNT\CurrentVersion\AppCompatFlags\TelemetryController\Oobe** exists and -maintenance was not provided. This key is deleted after it is checked.

If *-maintenance* was provided, then we validate that we can run and enter run mode zero. Validation consists of **HKEY_LOCAL_MACHINE \SOFTWARE\Microsoft\WindowsNT\CurrentVersion\AppCompatFlags\TelemetryController\TestAllowRun** being set to REG_DWORD that is not zero or passing a system state validation.

The validation will check these conditions:

- “Power Saver” must not be active, if it is on, the program will always fail validation
- Program will pass validation if the machine is plugged in
- If the program has failed validation four previous times, we will pass if our battery status

is unknown, if the battery is greater than 5%, or if we are charging

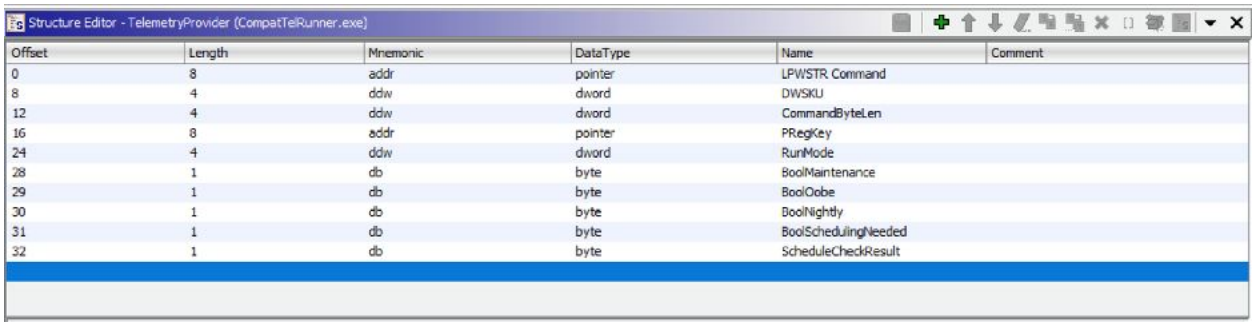
Once validation passes, the program will reset the registry key **RunsBlocked** to zero. If validation does not pass, the program will increment **RunsBlocked** by one.

If no command line parameters are passed, then **CompatTelRunner.exe** will enter run mode one (Nightly)

After the run mode is identified, some checks are performed against the scheduled task. We will then enter the area we care about, internally called RunTelemetry.

There are some additional checks run if our runmode is zero. If these checks pass, or if our run mode is not zero, the program will open **HKEY_LOCAL_MACHINE \SOFTWARE\Microsoft\WindowsNT\CurrentVersion\AppCompatFlags\TelemetryController.**

All subkeys are enumerated under this folder and used in an Initialize call to populate a structure. From the disassembly, this is what I found used in the structure.



Offset	Length	Mnemonic	DataType	Name	Comment
0	8	addr	pointer	LPWSTR Command	
8	4	ddw	dword	DWSKU	
12	4	ddw	dword	CommandByteLen	
16	8	addr	pointer	PRegKey	
24	4	ddw	dword	RunMode	
28	1	db	byte	BoolMaintenance	
29	1	db	byte	BoolOobe	
30	1	db	byte	BoolNightly	
31	1	db	byte	BoolSchedulingNeeded	
32	1	db	byte	ScheduleCheckResult	

Figure 2 – Structure Definition

These Fields are populated from the registry key:

- Command populates the *LPWSTR* Command/CommandByteLen
- Maintenance sets **BoolMaintenance**
- Nightly sets **BoolNightly**
- Oobe sets **BoolOobe**
- Sku sets the *DWORD* **DWSKU**
- SchedulingNeeded sets **BoolSchedulingNeeded**

Scheduling will perform additional registry reads, though for purpose of this post, they are not relevant.

Now we're at the fun part! The command specified is loaded into a buffer like so:

```
char command[520] = {0};
StringCchCatW(command, 260, L"%ls %ls%hs", this->CommandStr, L"-cv", <Some
Random looking string>);
```

Based on the run mode/scheduling used, *-oobe* or *-fullsync* may be added to the command line. Ultimately, this is passed as the second parameter to **CreateProcessW**, which is equivalent to running it as a shell command.

It's worth noting that **CompatTelRunner.exe** will block until the process it starts returns, so keep that in mind if you are adding programs to this key.

Why Does This Matter?

Attackers are always looking to find new and interesting ways to accomplish the same goal on target networks. To date, I have not seen this method of persisting on a machine documented. For blue teams, this points out a new registry location to add to Sysmon or other monitoring tools. For red teams, it outlines a new persistence mechanic that can be used on client networks. It also serves as a fairly trivial Admin > System elevation.

Currently, I would not call this a security bug. Admin level privileges are required to modify the keys affected. Mostly, this shows me that there are still many surfaces in Windows worth investigating, and just when you thought every vulnerability had been revealed, there is still more to discover