



To make Medium work, we log user data. By using Medium, you agree to our <u>Privacy Policy</u>, including cookie policy.



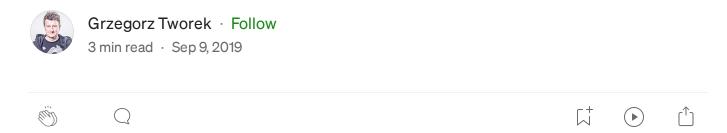
 \times

Sign in



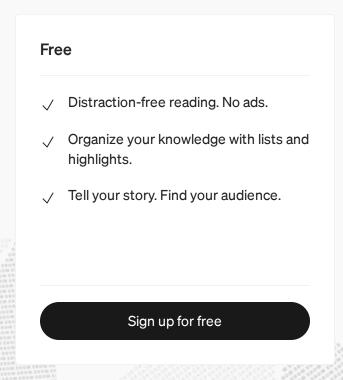
X

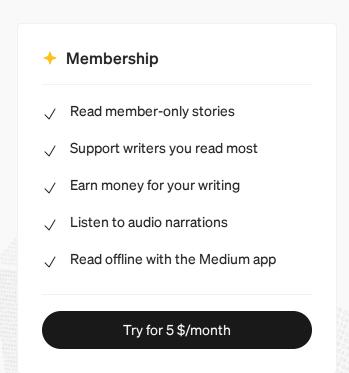
Using UEFI to inject executable files into BitLocker protected drives



To keep important things clear: BitLocker is a Windows-based full volume encryption solution. It encrypts every single sector of the volume, acting on the lowest possible layer — encrypting the data just before being written to the hardware and decrypting freshly it immediately after reading. BitLocker is considered relatively secure (including FIPS certification) and one of the main purposes is to protect data "at rest", when the Operating System cannot guard it. In practice it means, BitLocker is totally transparent for computer users, but if you try to play the data offline (mounting the disk drive to another machine, booting from USB stick etc.) — you realize everything is encrypted. The beauty of the solution is highly related to a method how BitLocker manages encryption keys, but we will not cover it here as totally irrelevant to the main topic. Just to wrap it up: when the OS is running it protects unauthorized users from manipulating critical data, and when the OS is not working — the data is encrypted an you cannot manipulate it too.

Medium





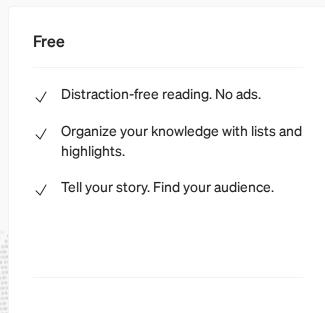
- Initializing OS parameters such as safe hoot etc
- To make Medium work, we log user data. By using Medium, you agree to our <u>Privacy Policy,</u> including cookie policy.
- Performing file operations postponed till reboot
- Logging drivers if enabled
- Initializing data related to Known Dlls
- Initializing pagefile(s)

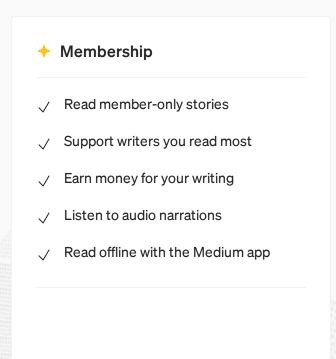
Etc.

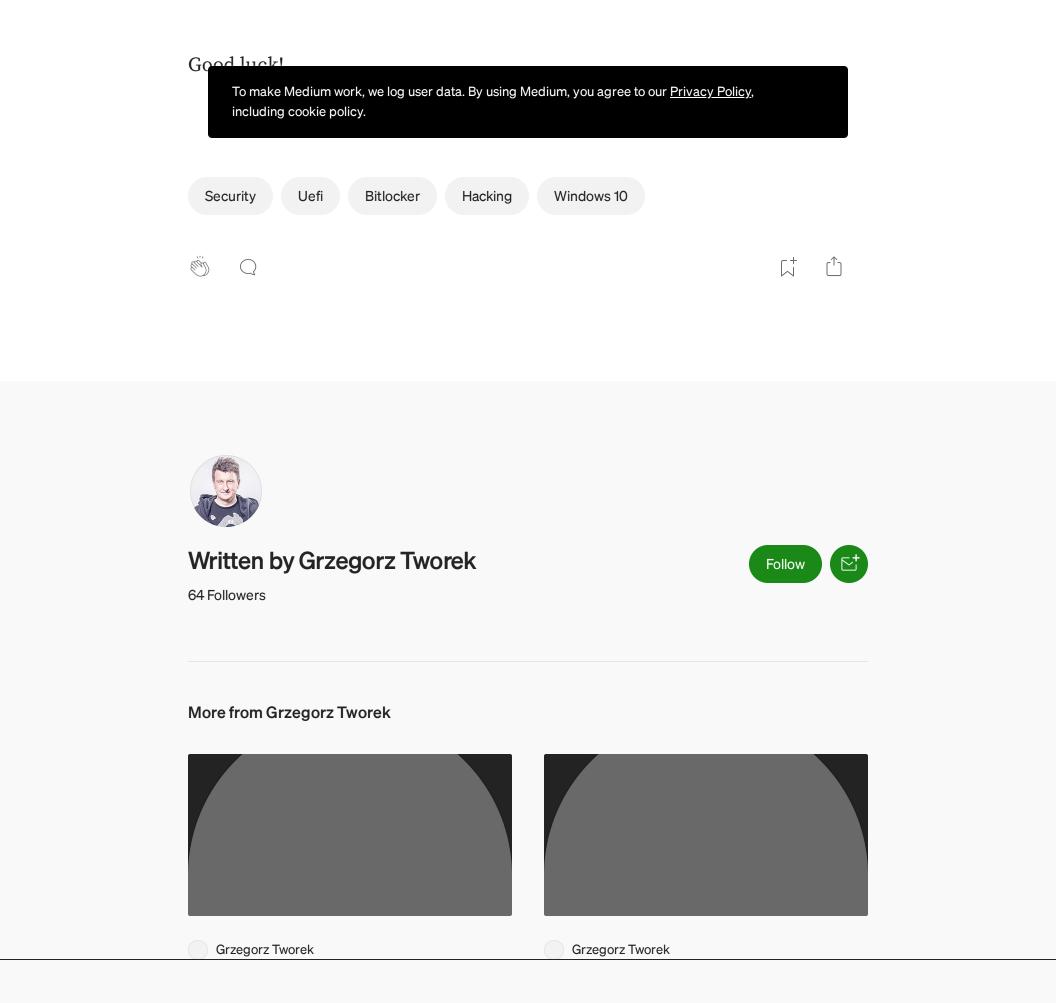
One of the steps to be performed relies on a NtQuerySystemInformation()
function (depreciated by Microsoft) with a 0x85 as a parameter. This parameter is not documented but according to the information provided within PDB symbol files, it may be interpreted as SystemPlatformBinaryInformation. NtQuerySystemInformation() scans UEFI tables stored within hardware memory looking for a piece of data with properly constructed headers. If such pattern ("WPBT", length, revision and a checksum) is found, the structure is passed to the smss.exe. And here the magic begins.

- Smss.exe stores the piece of UEFI memory within a file called wpbbin.exe.
- Smss.exe takes execution parameters (command line) from the same UEFI block.
- The wpbbin.exe is checked for integrity with <u>IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY</u>.
- The wnhhin exe is executed

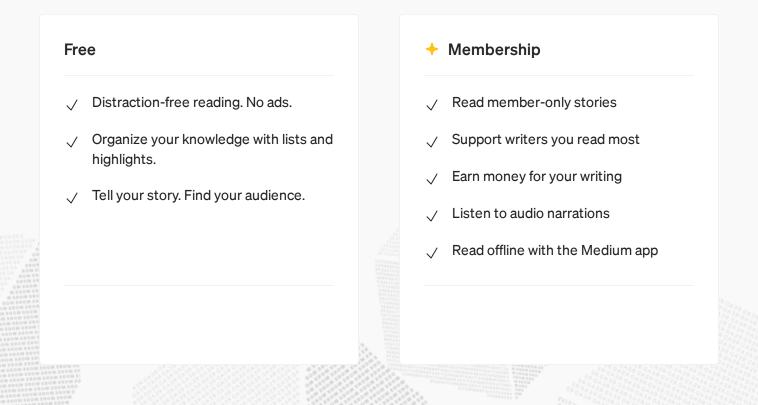
Medium







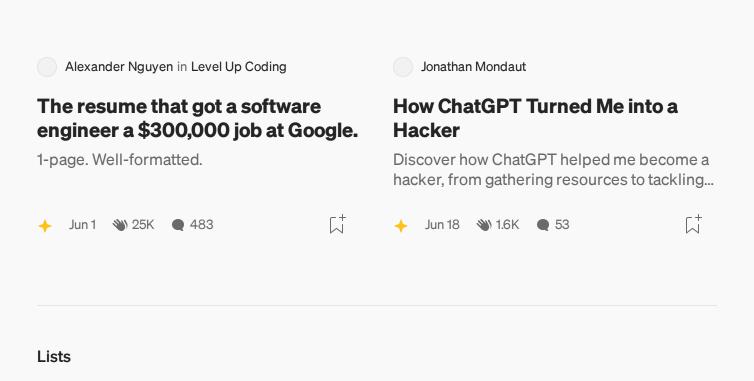
Medium



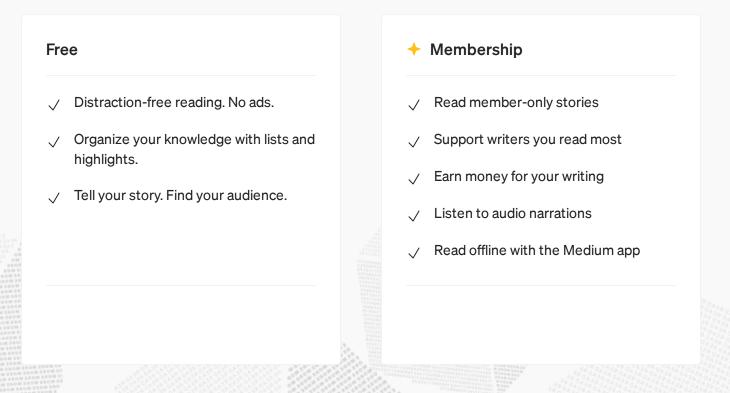
To make Medium work, we log user data. By using Medium, you agree to our <u>Privacy Policy</u>, including cookie policy.

See all from Grzegorz Tworek

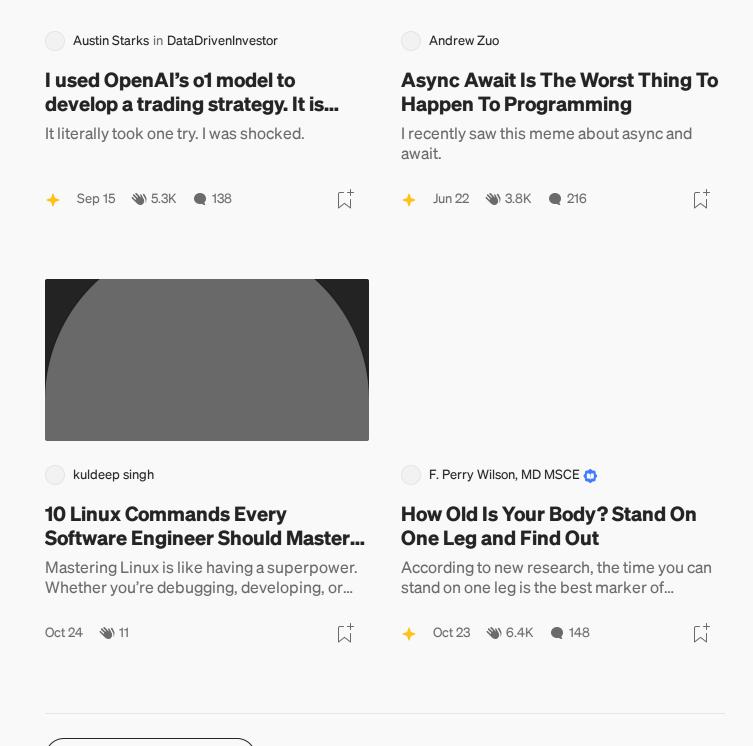
Recommended from Medium



Medium



To make Medium work, we log user data. By using Medium, you agree to our <u>Privacy Policy</u>, including cookie policy.



Medium

