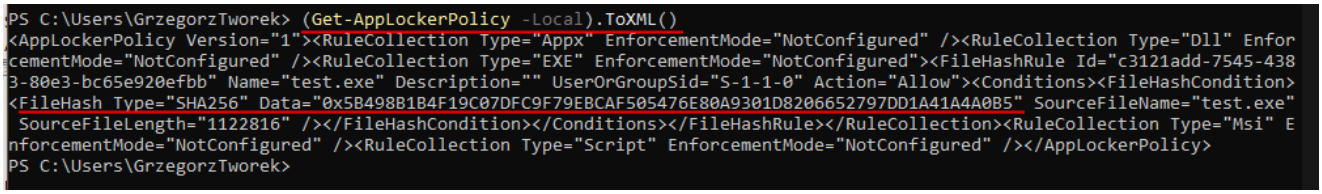
AppLocker bypass by hash caching misuse

gtworek.github.io/PSBits/applockercachebypass.html

If you need to know more about AppLocker - https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/applocker-overview

As we know, AppLocker may use 3 different types of rules for whitelisting executable files: `Path`, `Publisher`, and `Hash`. The following whitepaper covers Hash rules. Such rules can be created through the wizard displayed in *secpol.msc -> Application Control Policies -> AppLocker -> Executable Rules -> Create New Rule...*

The wizard calculates a hash of the file, and then stores it within AppLocker policy. GUI will not display the hash value, but it may be checked from the PowerShell by issuing a command `(Get-AppLockerPolicy -Local).ToXML()`

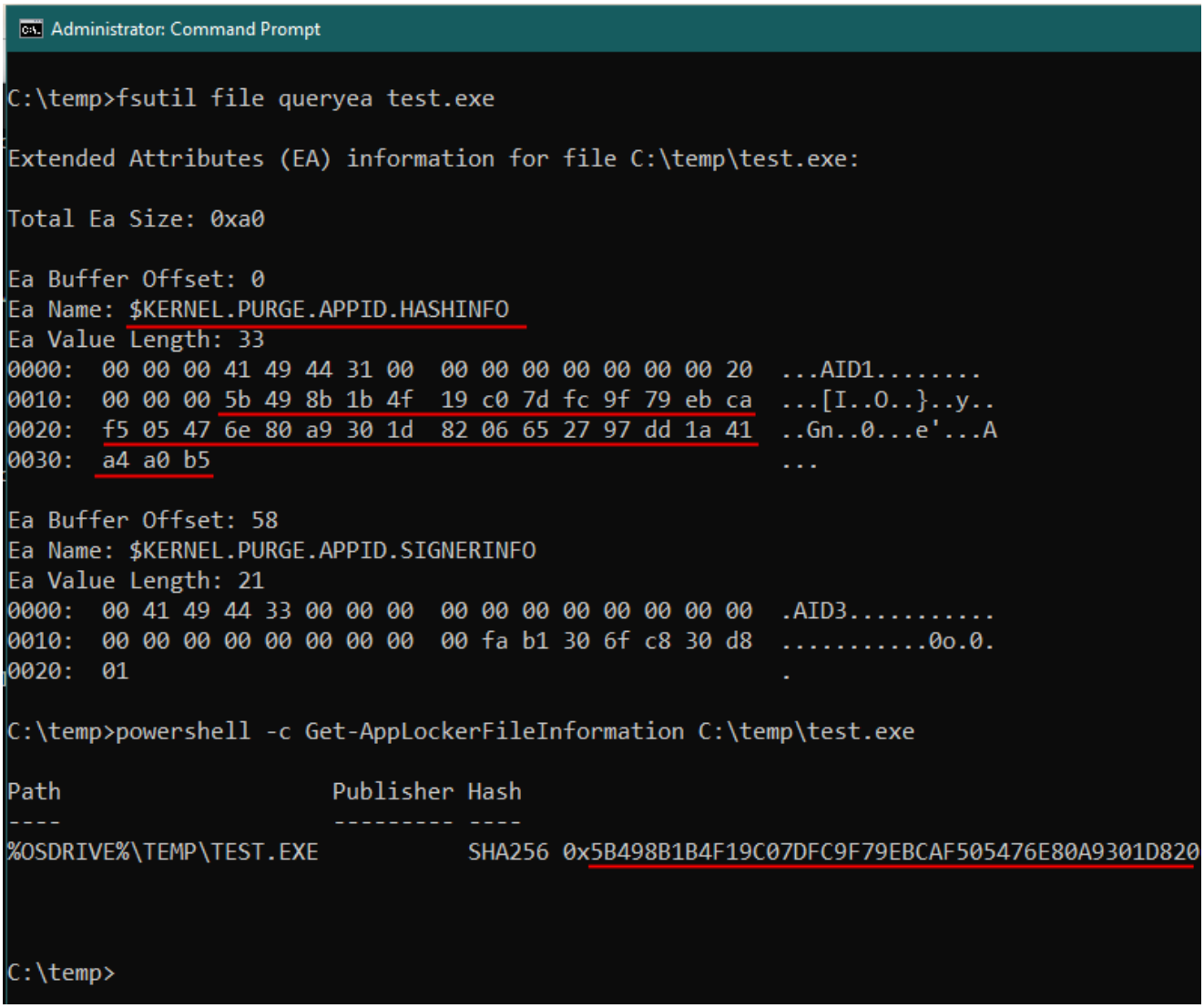


Despite naming the hash "SHA256", the value **IS NOT** the SHA256 of the file. Instead, the custom way of calculating the hash is used, and Microsoft does not officially publish the method of calculating it. At the first use of the executable file, its hash is calculated again (using the same algorithm) and if hashes match, the AppLocker rule is applied.

As the hash calculation is expensive in performance terms (regardless the algorithm), Microsoft decided to cache the calculated hash, and store it in a very special way together with the corresponding file. Such cache should meet couple of requirements:

- tightly bound with its file - met by using NTFS Extended Attributes (EA), as documented by Microsoft,
- protected from tampering/manipulation - met by naming EA with `$kernel` prefix. Such EAs can be modified/created only by Windows Kernel, which makes it protected from usermode processes,
- invalidated immediately if the executable file content changes - met by adding `purge` prefix, according to the documentation.

For the AppLocker hash cache, the full name `$kernel.purge.appid.hashinfo` is used. The complete list of EAs, including their contents may be displayed using the built-in command: `fsutil.exe file queryEA x:\path\file.exe`



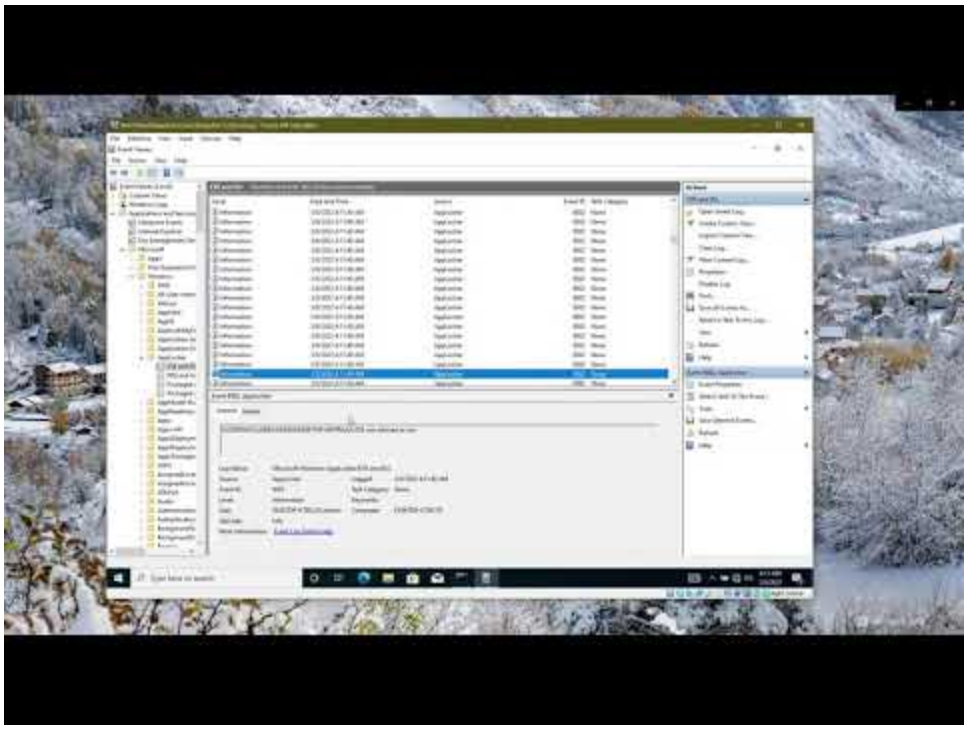
Effectively it means, such cache stays with a file, and when the file is manipulated, the old cache is deleted, and then re-calculated at the next run.

According to Microsoft, it's NTFS kernel driver responsible for the invalidation. If a malicious actor can change the file content without using the NTFS driver, it is possible to create a situation, where the cache does not reflect the real file being executed. It may be misused in the following scenario:

1. Administrator creates the AppLocker hash rule for a trusted file,
2. Malicious actor runs the file, to make sure its hash is calculated and then cached in the NTFS Extended Attribute,
3. Computer is turned off, and the hard disk content is manipulated, replacing the exe file with malicious code,
4. Computer is turned on, and the executable file may be run.

When AppLocker needs to determine if there is a hash rule allowing to run the file, it checks only the cache. As the cache remained intact, the file will be allowed to run, despite the changed content, and of course its real hash. The whole scenario can be seen on a short video,

demonstrating it on a VM and its virtual drive.



Even if the cache misuse can be see as a weakness, it does not open any new way of attacking the protected Windows OS. If a malicious actor can manipulate the file content without using NTFS driver, it may compromise Windows security in many other ways too.