Downlevel Driver Enabler

by Jason A. Donenfeld (zx2c4)

"Downlevel Driver Enabler" enables the use of Windows 10 PnP signed drivers on Windows 7 and 8.1.

Last year, Microsoft announced that they would no longer provide intermediate certificates for Authenticode-signed kernel drivers after June 1, 2021. This prompted widespread panic, as it effectively meant that it would be now impossible for many drivers to issue updates — for reliability, security, or otherwise — on Windows 7 and 8.1. OSR ran a blog series on this, culminating in a final post in April indicating that after much haggling with Microsoft PMs, the prospects of any change to the policy is hopeless, and the situation is a "lost cause". R.I.P. Windows 7 and 8.1 driver updates? Not at all, and that is what this repository presents.

Driver signing background

Before Windows 10 1607, there were two ways of signing drivers: Authenticode signatures, in which you pay a CA for the ability to sign your own drivers, or Windows Hardware Compatibility Publisher signatures, in which you either run your driver through a battery of hardware tests, called WHLK (which OSR points out is impossible for most driver types), and submit the results of those tests to Microsoft, or more recently, simply ask Microsoft for an "attestation signature", which amounts to more or less the same thing without the testing headache. At some point Microsoft was going to require WHLK testing for Windows Server, but eventually gave up on that, so now attestation signatures are fine for both Windows 10 Client and Windows 10 Server (2016/2019). But attestation is only for Windows 10, which means if you want a Windows Hardware Compatibility Publisher signature on Windows 7 and 8.1, you must go through the testing that may well not be available for your driver.

As an aside, it turns out you can actually use an old Authenticode certificate basically indefinitely — beyond the June 1, 2021 expiration date — by timestamping it using a bogus timestamping server, and then adding the bogus CA used to generate those timestamp signatures to the system's trust store. Evidently the requirements of timestamp CAs are less stringent than those of code signing CAs. While there's arguably a "safe" way of doing that, (ab)using expired or intermediately expired Authenticode certificates seems to go against the spirit of the requirements, and so it seems a bit too dirty for production. One could imagine getting a certificate consequently blacklisted with tricks like that.

So, with WHLK not available for many drivers, and Authenticode no longer viable after June 1, 2021, it would seem the only way forward for driver updates of any kind is on Windows 10, using attestation signatures, and just giving up entirely on trying to ship security or reliability updates to Windows 7 and 8.1. That laziness is appealing, but also not viable for real world systems that still require the old operating systems.

It turns out that Windows 7 and 8.1 will load drivers that have been signed using the Windows 10 attestation service, but only if they are non-PnP (i.e. do not use an `.inf` and `.cat` file). That means Windows 7 and 8.1 developers of non-PnP drivers can simply transition to the Windows 10 attestation service after June 1, 2021 and all will be well. But PnP drivers — extremely common — are still left out in the cold. The distinction between the two driver types, however, provides a hint.

Driver signature verification

A first inclination upon learning that non-PnP drivers can load but PnP drivers cannot might be that one could just write a little non-PnP rootkit driver to fiddle around with whatever needs fiddling with, enabling the PnP driver to load subsequently. That, again, seems unfortunately too dirty for production, and a bit intellectually lazy too. Instead it is more interesting to understand the *actual* difference between the non-PnP case and PnP case.

The kernel verifies drivers when they are being loaded, in order to make sure that untrusted code is not loaded into the most trusted part of the OS. To this end, the loader is concerned primarily with the signature on the `.sys` driver code itself, rather than any supporting userspace files around it. So, the signature verifier — implemented in `ci.dll` — looks at the signature in the `.sys` and makes sure that it chains up to a valid root in a valid way. In our case here, the relevant chaining is that it ends in a particular Microsoft certificate related to the Windows Hardware Compatibility Publisher with proper EKUs. If all checks out, then the driver loads. It is very simple. For this reason, Windows 10 attestation works on both Windows 10 and Windows 7 and 8.1. The kernel's verifier cares that a driver is trusted by Microsoft, since the relevant security boundary here involves trust, rather than which particular operating system it has been "certified" to run smoothly on. And if you think about it, that makes sense: the kernel is trying to enforce signatures as a means of security, in order to have a trusted boundary. The policy it cares about is a simple security one, rather than anything fancier or more pedantic about certifications or WHLK test suites or anything like that. This is a real, important security boundary.

The userspace PnP driver store is a bit more complicated. Here, it not only cares about the signature of the `.sys` driver code itself, but also all of the other supporting userspace files, such as the `.inf` file and other programs the `.inf` file might instruct the OS to install. These supporting files are listed in a `.cat` file, and this `.cat` file is signed with the same type of signature as the `.sys` driver code file. But the `.cat` file also has some additional fields, the most relevant of which is the `OSAttr` field, which lists the version of Windows with which the driver has been certified or attested to work. The userspace PnP driver installer, `drvinst.exe`, cares about this, and will return `ERROR_SIGNATURE_OSATTRIBUTE_MISMATCH` (0xE0000244) if `OSAttr` lists a different Windows version. This is *not* a security check. It is a boring policy check, and one that is not even uniformly applied, as the kernel's verifier does not care about it, hence the case of non-PnP drivers without `OSAttr` checks. And seeing that certification for Windows 7 and 8.1 is not even possible now, it is no longer even a *sensible* policy check. And, to repeat again, this is very much *not* a security check. It might now be described as an *outdated* or *obsolete* policy check.

Many articles on similar topics would now attempt to dazzle you with colorful screenshots of IDA Pro, indicating the impenetrably byzantine nature of the following reverse engineering work. In reality, though, the analysis here is not overly fancy: the PnP driver installer — `drvinst.exe` — calls into `setupapi.dll`, which eventually finds its way to `VerifyFile`, which in turn calls `WinVerifyTrust(DRIVER_ACTION_VERIFY)` in `wintrust.dll`. If that function returns `ERROR_APP_WRONG_OS` (0x0000047F), then `VerifyFile` returns

`ERROR_SIGNATURE_OSATTRIBUTE_MISMATCH` (0xE0000244) to its caller. Looking at `wintrust.dll`'s `WinVerifyTrust`, there is a dynamic function dispatch based on the GUID argument, which eventually leads to a call to `DriverFinalPolicy`, which in turn uses `CryptCATGetCatAttrInfo` and `CryptCATGetAttrInfo` to read the `OSAttr` field, and then sees if it matches the running OS using `_CheckVersionAttributeNEW`, returning a boolean. If it returns true, `DriverFinalPolicy` returns `ERROR_SUCCESS` (0x0); if not, it returns `ERROR_APP_WRONG_OS` (0x0000047F).

So naturally one starts to consider different ways of injecting into system services or patching binaries on disk or corrupting the file system cache or any of the usual techniques for such things, to turn either `ERROR_SIGNATURE_OSATTRIBUTE_MISMATCH` or `ERROR_APP_WRONG_OS` into an `ERROR_SUCCESS`. But fortunately, no such dirty technique is required. The `wintrust.dll` framework already gives us everything we need for such modifications, without having to resort to the dark arts.

When we call `WinVerifyTrust(DRIVER_ACTION_VERIFY)`, the `DRIVER_ACTION_VERIFY` constant is actually a GUID. `wintrust.dll`, in `_CheckRegisteredProviders` and `GetRegProvider`, then looks in `HKLM\SOFTWARE\Microsoft\Cryptography\Providers\Trust\{function name}\{that g` at two values, `$DLL` and `$Function`. If `$DLL` is not `wintrust.dll`, it calls `LoadLibraryW` on it (not `LoadLibraryExW`! yikes, but unrelated), and then it calls `GetProcAddress` on `$Function`. Finally it calls the resolved function.

Thus, all we have to do is implement our own `DriverFinalPolicy` function that calls the original one in `wintrust.dll`, and converts a return value of `ERROR_APP_WRONG_OS` (0x0000047F) into `ERROR_SUCCESS` (0x0). And presto, we are done, and Windows 10 drivers can load successfully on Windows 7 and 8.1. We do this *without* having to break any real security barriers or do anything dirty. Rather, we use the nice dynamic dispatch facilities already available in the OS to remove a now-antiquated OS version policy check. In some sense, Microsoft foresaw the need for pluggable policy years in advance.

Usage

So, with the above in mind, the actual implementation is trivial. Compile the ~20 line `shim.c` file in this repository into a `shim.dll`, and then set the

```
HKLM\SOFTWARE\Microsoft\Cryptography\Providers\Trust\FinalPolicy\{F750E6C3-38EE-1
```

registry key to the location of your `shim.dll`. When you are done, set the key back to its original value. (It is not recommended to leave the registry key pointing to your `shim.dll` or to install your `shim.dll` into `system32`, as multiple parties doing that will inevitably lead to the "dll hell" of yore.)

A driver installation at the command line can be easily simplified to:

```
> reg add HKLM\SOFTWARE\Microsoft\Cryptography\Providers\Trust\FinalPolicy\{F750E
> pnputil -i -a mydriver.inf
> reg add HKLM\SOFTWARE\Microsoft\Cryptography\Providers\Trust\FinalPolicy\{F750E
```

There is one caveat to consider, which is that the registry is a *shared resource*, and so multiple installers all using this method at once is going to lead to issues. Therefore, when doing this, take a mutex in a private namespace (so as to mitigate the trivial unprivileged DoS). So, by convention, let us do:

- Boundary descriptor: `L"DownlevelDriverEnabler"`
- Boundary descriptor SID: `WinLocalSystemSid` or `WinBuiltinAdministratorsSid`
- Private namespace: `L"DownlevelDriverEnabler"` with security attributes `O:SYD:P(A;;GA;;;SY)(A;;GA;;;BA)S:(ML;;NWNRNX;;;HI)` or `O:BAD:P(A;;GA;;;SY)(A;;GA;;;BA)S:(ML;;NWNRNX;;;HI)`
- Mutex name: `L"DownlevelDriverEnabler\\ShimInProgress"`

Take that mutex while shimming, and release it after the key has been restored to `WINTRUST.DLL`. If we all follow those rules, there will be safe and reliable support for driver updates on Windows 7 and 8.1. Hopefully this turns a rather hopeless situation into a productive one.

Addendum

Looking at things a bit closer, it appears as though the userspace PnP verifier checks for Authenticode signatures using the generic Authenticode check -- `WINTRUST_ACTION_GENERIC_VERIFY_V2`. This check is the normal Authenticode check that still remains valid for software in general, not just for kernel drivers. That means it is possible to receive Windows 10 attested `.sys.` and `.cat` files, and then simply *re-sign* the `.cat` file with a ordinary software Authenticode certificate. The still-valid software Authenticode certificate will enable PnP installation verifier to proceed, and the correct Microsoft signature on the `.sys` will allow the kernel to load it. In very brief tests, this appears to be the case, though it does warrant a bit more testing, as setupapi still aborts with `CERT_E_UNTRUSTEDROOT` (0x800B0109), despite letting the copy proceed, which on some configurations could wind up being fatal. In general this might require a bit more surgery than the above, but for others it could also prove a useful strategy.