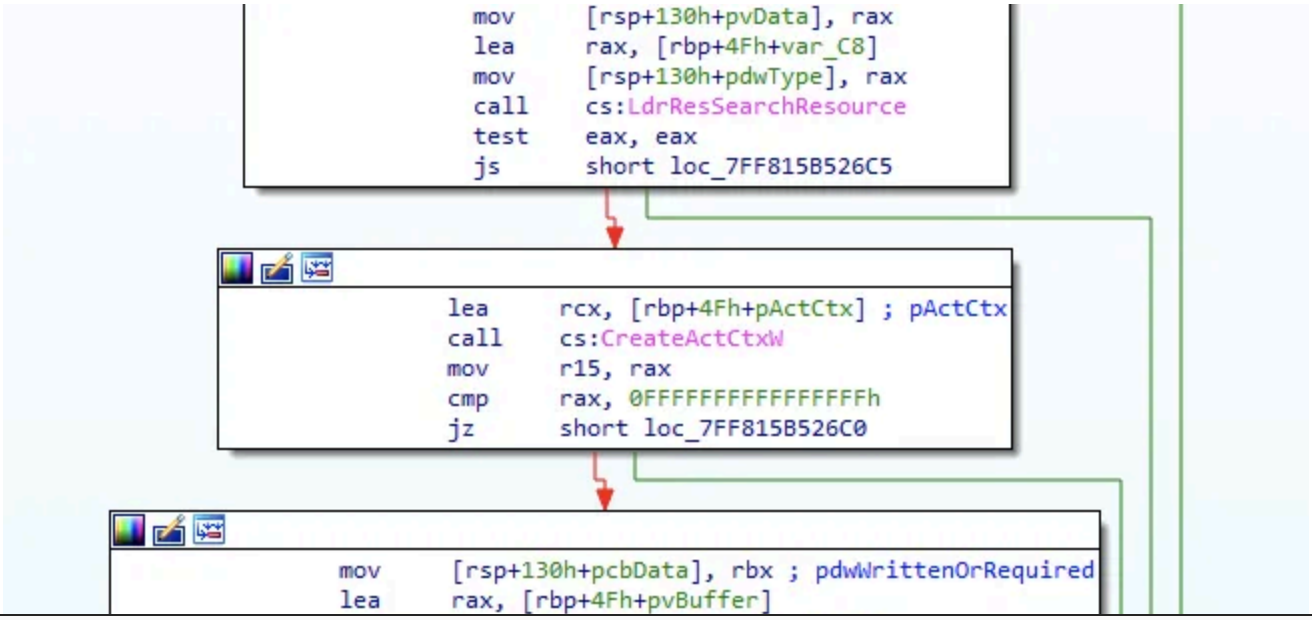
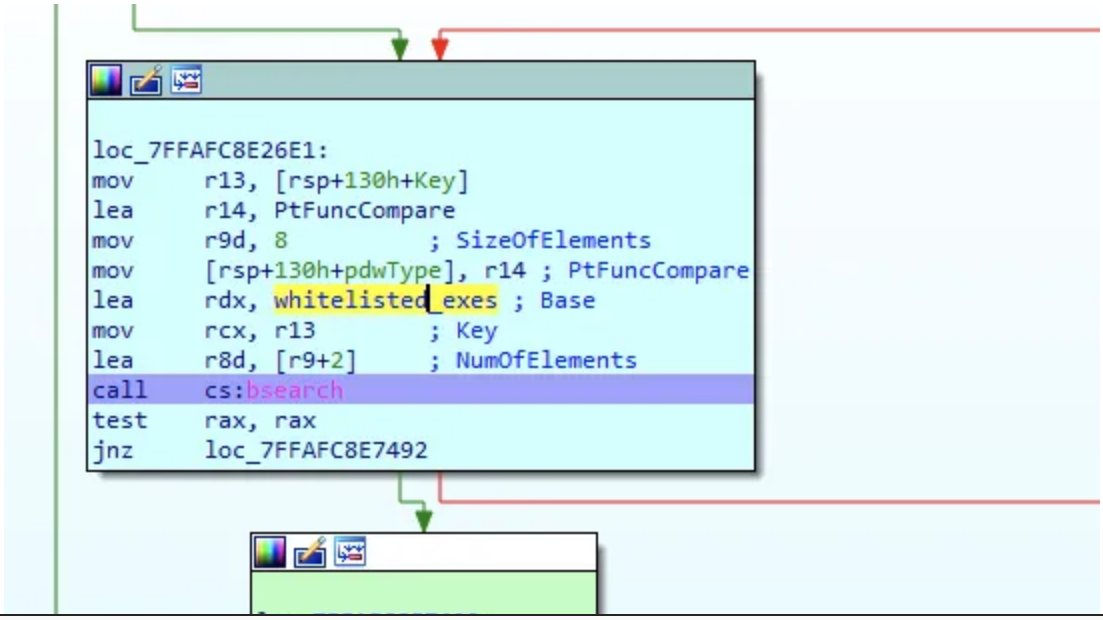
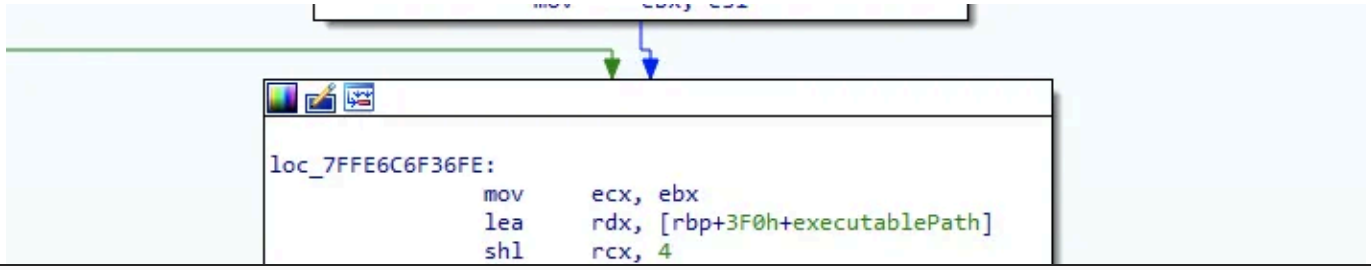
UAC Bypass by Mocking Trusted Directories

When a user that is part of the Administrators group wants to execute a process that requires elevation, the UAC prompt is presented to confirm process elevation to the user. This UAC prompt however, is not popped for ALL administrative executables on Windows. There are a few exceptions that will “auto elevate” the target executable with no UAC prompt being thrown thus bypassing UAC (to many’s surprise!). This select group of trusted executables have additional security checks done to them to ensure they are in fact trusted so this feature is not abused by malicious actors. This approach has been used in previous UAC bypasses and will be the heart of this bypass method as well. There are a few loopholes we need to bypass for





The last auto elevating requirement is that the target executable resides in a "trusted directory," such as "C:\Windows\System32". Figure 3 shows AIS doing this check on a path requesting elevation, in this case one of the paths its considering "trusted" is "C:\Windows\System32".



3. Executing from a trusted directory (“C:\Windows\System32”) .

Appinfo.dll (AIS) will use *RtlPrefixUnicodeString* API to see if the target executable path begins with “C:\Windows\System32\” for one of the trusted directory checks. This is pretty bullet proof check considering its comparing against the canonicalized path name of the target executable requesting elevation. So for bypassing this check, I construct a directory called “C:\Windows \” (notice trailing space after “Windows”). This won’t pass the *RtlPrefixUnicodeString* check of course, and I’ll also mention that this is

Figure 5 — Directory deletion requests silently fail and unable to rename directory to remove trailing space.

When this awkward path is sent to AIS for an elevation request, the path is passed to *GetLongPathNameW*, which converts it back to "C:\Windows\System32\winSAT.exe" (space removed). Perfect! This is now the string that trusted directory checks are performed against (using *RtlPrefixUnicodeString)* for the rest of the routine. The beauty is that after the trusted directory check is done with this converted path string, it is then freed, and rest of checks (and final elevated execution request) are done with the original executable path name (with the trailing space). This allows all other checks to pass and results in appinfo.dll spawning my winSAT.exe copy as auto elevated (since it is both properly signed and whitelisted for

Figure 7

Github to POC:

https://github.com/tenable/poc/tree/master/Microsoft/Windows/UACBypass