<u>Criteria for designating "critical software."</u>

The specification of “critical” as defined in the EO is likely the result of a flawed attempt by people who do not know what they are doing to create a criteria that is perhaps less than helpful in a technical sense, but sensible from a naive perspective. So NIST is left with the problem of designating “software that performs functions critical to trust”. However, no Federal guidance on what constitutes “trust” is apparent. Trust, generally, is often described as the extent to which one is willing to be harmed by failures of the subject being ‘trusted’, and the question then arises of ‘trust for what’? So the ‘what’ of the consequences associated with failures (in the software in this case) is then a necessary component of the ‘trust for what’ framework.

An obvious and sensible approach taken over a long time with moderate success is to start with the consequences, and match surety with consequence, so that higher surety is required for higher negative consequences. More accurately, potentially serious negative consequences, because as yet unrealized (potential) consequences are the things we are protecting against prior to their realization (left of boom). The answer to “Who could have anticipated that?” better be “us” (more accurately U.S.) and thus the question of trust is not associated with the software alone, but rather on its use.

A reasonable conclusion is then that the designation ‘critical software’ applies to any software, the failure of which, can produce potentially serious negative consequences. The supply chain issue is that the production of consequences may be direct or indirect, and recursively indirect to the origin.

In addition, software does not operate in isolation, and thus, the context of the operating environment must come into play when it comes to determining the relevance of software to consequences. The combination of other hardware, software, real-world physical circumstances, and the threat and operating environment must be taken into account when understanding what constitutes criticality.

There is also the issue of time frames. Educating people is critical to the long-term future of society, particularly in a technology-rich environment, and in the context of long-term potentials ranging from next week to the time frame when the Sun explodes. In other cases, consequences to society may happen in minutes, as in the time frames for nuclear war and global annihilation. Biological effects such as the current pandemic take years to unfold on a global basis. Software is involved in all of these things.

Of course the consequence determines the criticality, which the EO does not particularly spell out. The so-called critical infrastructures might be taken as the context for NIST to consider, but this may also be so limiting as to ignore larger consequences not identified in the current official list, which by the way changes with time. Thus what is critical today may not be critical tomorrow, and vice versa.

Then we come to the list provided by the EO, which identifies “level of privilege or access required to function, integration, dependencies, direct access to networking and computing resources, performance of a function critical to trust, and potential for harm if compromised.” Government agencies must follow executive orders, and thus these must be taken into explicit account.

- Potential for harm if compromised is addressed above.
- Dependencies are addressed above.
- Integration, to the extent it is a useful term, is addressed above.
- Level of privilege or access required to function: In essence, the EO specified here that the principle of least privilege should be applied, or perhaps more clearly, that whether it is applied or not, the privileges granted should be associated with trust as defined. The ‘required to function’ is a side effect of not only the mechanisms required (least privilege) but the implementation selection of function. The EO doesn’t state that the function or privilege should or can be limited, only that if that’s how it works, the criticality should be determined on that basis. As such, this is also covered above (the consequences of failures).

- Direct access to networking and computing resources: Again, the EO seems to identify this as special, but no software actually has direct access to anything. It is all interpreted by the physical mechanisms. A reasonable interpretation would have to do with the direct and indirect ability of the software to observe, suppress, and induce signals in the networking media and operations in the computational arena. While the EO does not address storage, it is a reasonable 3rd component to be included, and in the larger context of cybernetic systems, it is also reasonable to consider the cybernetic methodology of sensors, actuators, communication, and control. The basic concept here is direct and indirect effects, and regardless of how direct, the ability to effect is the apparently critical issue. The implication seems to be that direct induction of consequences is more important than indirect induction of consequences, but whether you are killed by the stoppage of blood getting to your brain (direct cause of brain death) or the stoppage of your heart that indirectly causes the stoppage, you are still dead. It is a common mistake to assume or conclude that the surety of the direct mechanisms of failure is somehow more important than the surety of the indirect mechanisms that induce those direct mechanisms to take place, and that is one of the reasons we see so many failures today.
- Performance of a function critical to trust: is a bit tricky in this context. "critical software" is somehow to be defined in terms of a function "critical" to trust. But this is of course recursive. In essence, this appears to be covered by the combination of direct and indirect and the notion of consequence.

So a proposed definition is:

> Critical software:= Any software, the failure of which, can potentially, on its own or in combination with other components in an operating environment, either directly or indirectly, and over relevant time frames, result in serious negative consequences.

Finally, it should be noted that the notion of 'software' vs. 'firmware' vs. 'data' vs, 'wetware' vs. 'hardware' is problematic in that some of these names may be a distinction without a difference (or perhaps a difference without a distinction) while others may produce the same consequences, and all typically work in concert in actual systems. Thus the indirect issue applies across all of these areas, for example, in that excessive trust in wetware (a person's mental processes) may produce data that is interpreted by software in a hardware mechanism to create firmware put in a hardware system that fails to be of adequate surety to prevent a serious negative consequence.

Clarification would perhaps be to say "any thing' rather than 'any software'.

Critical:= Anything, the failure of which, can potentially, on its own or in combination with other components in an operating environment, either directly or indirectly, and over relevant time frames, result in serious negative consequences.

But perhaps that's for the next EO when we find out that the software worked correctly and the people did the wrong things… Or is that the last even that triggered the current EO?[1]

1 Fred Cohen – CEO – fc@all.net – please consider me as a speaker at the workshop