As a group we discussed the security implications of TrueCrypts (file encryption services) software implementation.

My discussion initial discussion post:

In 1992 Linus Torvalds and Andrew Tanenbaum debated the 'best' architecture for operating system kernels. A kernel is the lowest layer of an operating system that interfaces with the hardware and manages system resources. Linus Torvalds was the primary developer of Linux and Andrew Tanenbaum was the primary developer of an operating system called Minix. Andrew argued that Linux' monolithic design was flawed and that microkernels designs are superior. Tanenbaum argued that Linuxs' monolithic design made the system hard to port. A monolithic kernel implements all the necessary features of the kernel in one, whereas a microkernel is broken into multiple separate services that implement the whole kernel. Monolithic kernels operate faster than microkernels, due to additional communication through each services interface. Microkernels can be more secure, errors in individual services do not crash the system.

I will now consider the statement "Torvalds has been proven wrong and it only took nearly thirty years. Microservices and microkernels are the future". Firstly, we can compare the success of the operating systems of each developer. Linux went on to have far greater prominence than Minix. The most common modern OS's are a hybrid between Monolithic and Microkernels. Windows and Linux are closer to Monolithic kernels. From this perspective, I would disagree with the statement. However, Microservices architectures have become dominant for modern web applications. For web applications, microservices offer greater reliability and scalability. The reason this architecture may be more applicable for web applications maybe because of the different performance requirements. Operating systems and Web applications are software that runs at different layers. The application layer has lower performance needs. A web application ironically may be best suited to an Microservice design, but the code will likely run-on operating systems with monolithic architectures. As such, I partially agree with this statement. As such, I partially agree with this statement.

[308 words]

My response to my colleague Davids post:

Hi David,

Thank you for this very interesting post. I found your deep dive into paging very interesting and enjoyed reading the Microsoft material you referenced.

I was surprised to find that Microsoft (as of the 2009) simply left the system to execute un-defined behavior when if the paged pool is exceeded. I would think that in circumstances where you can not verify program or (or in this case OS) behavior the safest thing to do would be to 'crash gracefully'. I am surprised an error message was not displayed to handle this situation explaining to the user what has happened.

I think it is worth discussing the benefits of virtual memory paging. Developers do not need to worry about allocating memory in the system, and instead have a continuous virtual block. The operating system handles the 'fragmented' data. It also becomes easier for operating systems to ensure security as applications have no means to specify addresses outside of their virtual pool.

My response to my colleague Davids post:

Hey Anrich,

I think you raise an interesting point regarding signed and unsigned integers. I have read arguments for and against using signed vs unsigned integers in C. Signed integers in C can be dangerous because overflow is undefined.

If two signed integers are added together that is greater than the size of the integer then the result will be undefined. On the other hand, Unsigned overflow is clearly defined.

Robert Elder argues that a weakness of using unsigned integers is that the C standard library uses signed integers and therefore compatibility issues can occur.

Elder, R. (2015) Should I use Signed or Unsigned Ints In C? Available from: https://blog.robertelder.org/signed-or-unsigned/[Accessed 21 July 2021].