

space, for example by preventing accesses to unallocated memory. If any of these checks fail an exception is raised, in the former case for invalid instructions and in the latter case for invalid address (sometimes called segmentation violation or bus error).

This may seem to the reader to be a very strange point of view. After all, doesn't it matter whether the program is well formed? Well, of course it does. But the problem is that it is not feasible to show that the program is well formed, both because of undecidability issues and because to show that a program is well formed it must conform to a specification which is, itself, a kind of program.

Furthermore, a program may be executed which is under development or otherwise buggy. An alternative is to require the compiler to produce higher quality code, but this is in part a programming language issue. And if the compiler is required to produce a proper (though not necessarily correct) binary, we have made the compiler part of the TCB.

While it may seem that the compiler is always part of the TCB, this is not necessarily so, since there may be one compiler (and perhaps one programming language) for writing security sensitive code while allowing other, unverified languages and compilers on the system.

The operating system kernel tends to be fairly difficult to attack. First bugs in the kernel can destabilize the system (causing crashes) and hence the kernel is very conservatively maintained. Kernel code is extensively read and reviewed by very skilled people. Yet kernels have been growing in size and number of maintainers and there are a disconcerting number of attacks discovered lately in the Linux kernel. In general, the weakest area of the kernel is the device drivers because of their variety and because most device drivers don't affect most users (since they don't have the requisite hardware), the drivers are not as well tested as the rest of the code. In open source code, there may be no (non-developer) testing of the driver.

4.1.4 Compilers

Compilers, like NVRAM boot loaders, play a transitory role since they are not present when the code they create executes. Yet, like a boot loader, the compiler can fail to faithfully translate the program, either through bugs or as an intended attack.

In Ken Thompson's Turing Award Lecture, he describes just such an exploit. The rumor has been that this exploit was invented in contemplation

of and on the event of shipping a UNIX tape to the **NSA (National Security Agency)**—the agency which oversees computer security and electronic eavesdropping for the US government.

The exploit involved changing the login program in the operating system so that it would have a **back door**—i.e. a hidden entrance. A user name and password was hard coded into the login program, so that anyone knowing this user name and password could log onto the system without being in its password file.

Of course, NSA would be expected to examine the code—that is, after all, one of the responsibilities they are charged with by the US government. In such an evaluation, the login program is of particular interest since it is critical to security. No doubt they would detect and remove the back door.

Hence, a second much more subtle attack was encoded into the system, in the compiler. The compiler had special code to detect the login program and insert the back door into it during compilation.

Moreover, the compiler was modified to insert the back door inserting code in the compiler if the back door inserting code was removed from the compiler source.

Now, all back door source code and back door inserting source code could be removed. The binaries contained the back door and back door inserting code, but nowhere is it apparent from reading the source code that there is a back door.

In order to discover the back door, it is necessary to examine the binaries.

This is why highly sensitive software is always developed with trusted developers, since it can be very hard to discover an attack such as this. In general, attacks at lower levels can be made more obscure.

4.1.5 Libraries

As we have described, a process relies on the kernel to perform some tasks for which it does not have sufficient privileges. Does that mean that all the non-kernel code is written by application programmers?

Not quite. Significant functionality is included in the standard (and non-standard) libraries. The **standard libraries** are the language-specific libraries as defined in its language standard. The non-standard libraries are either extensions to the standard libraries or are for system functionality that is beyond the scope of the language definition.

Since the standard libraries are heavily used, they are likely to be fairly