4.3. The Role of Trust

The role of trust is crucial to understanding the nature of computer security. This book presents theories and mechanisms for analyzing and enhancing computer security, but any theories or mechanisms rest on certain assumptions. When someone understands the assumptions her security policies, mechanisms, and procedures rest on, she will have a very good understanding of how effective those policies, mechanisms, and procedures are. Let us examine the consequences of this maxim.

A system administrator receives a security patch for her computer's operating system. She installs it. Has she improved the security of her system? She has indeed, given the correctness of certain assumptions:

1. She is assuming that the patch came from the vendor and was not tampered with in transit, rather than from an attacker trying to trick her into installing a bogus patch that would actually open security holes. Winkler [1052] describes a penetration test in which this technique enabled attackers to gain direct access to the computer systems of the target.
2. She is assuming that the vendor tested the patch thoroughly. Vendors are often under considerable pressure to issue patches quickly and sometimes test them only against a particular attack. The vulnerability may be deeper, however, and other attacks may succeed. When someone released an exploit of one vendor's operating system code, the vendor released a correcting patch in 24 hours. Unfortunately, the patch opened a second hole, one that was far easier to exploit. The next patch (released 48 hours later) fixed both problems correctly.
3. She is assuming that the vendor's test environment corresponds to her environment. Otherwise, the patch may not work as expected. As an example, a vendor's patch once reset ownerships of executables to the user **root**. At some installations, maintenance procedures required that these executables be owned by the user **bin**. The vendor's patch had to be undone and fixed for the local configuration. This assumption also covers possible conflicts between different patches, as well as patches that conflict with one another (such as patches from different vendors of software that the system is using).
4. She is assuming that the patch is installed correctly. Some patches are simple to install, because they are simply executable files. Others are complex, requiring the system administrator to reconfigure network-oriented properties, add a user, modify the contents of a registry, give rights to some set of users, and then reboot the system. An error in any of these steps could prevent the patch from correcting the problems, as could an inconsistency between the environments in which the patch was developed and in which the patch is applied. Furthermore, the patch may claim to require specific privileges, when in reality the privileges are unnecessary and in fact dangerous.

These assumptions are fairly high-level, but invalidating any of them makes the patch a potential security problem.

Assumptions arise also at a much lower level. Consider formal verification (see Chapter 20), an oft-touted panacea for security problems. The important aspect is that formal verification provides a formal mathematical proof that a given program **P** is correct—that is, given any set of inputs **i**, **j**, **k**, the program **P** will produce the

output **x** that its specification requires. This level of assurance is greater than most existing programs provide, and hence makes **P** a desirable program. Suppose a security-related program **S** has been formally verified for the operating system **O**. What assumptions would be made when it was installed?

1. The formal verification of **S** is correct—that is, the proof has no errors. Because formal verification relies on automated theorem provers as well as human analysis, the theorem provers must be programmed correctly.
2. The assumptions made in the formal verification of **S** are correct; specifically, the preconditions hold in the environment in which the program is to be executed. These preconditions are typically fed to the theorem provers as well as the program **S**. An implicit aspect of this assumption is that the version of **O** in the environment in which the program is to be executed is the same as the version of **O** used to verify **S**.
3. The program will be transformed into an executable whose actions correspond to those indicated by the source code; in other words, the compiler, linker, loader, and any libraries are correct. An experiment with one version of the UNIX operating system demonstrated how devastating a rigged compiler could be, and attackers have replaced libraries with others that performed additional functions, thereby increasing security risks.
4. The hardware will execute the program as intended. A program that relies on floating point calculations would yield incorrect results on some computer CPU chips, regardless of any formal verification of the program, owing to a flaw in these chips [202]. Similarly, a program that relies on inputs from hardware assumes that specific conditions cause those inputs.

The point is that **any** security policy, mechanism, or procedure is based on assumptions that, if incorrect, destroy the superstructure on which it is built. Analysts and designers (and users) must bear this in mind, because unless they understand what the security policy, mechanism, or procedure is based on, they jump from an unwarranted assumption to an erroneous conclusion.