architects and designers, awareness of these foundational concerns across the team is a powerful force for reducing the risk to the software posed by security issues. Brief descriptions are given here for each principle; more detailed descriptions and examples are available on the Principles content area on the BSI Web site.

The Principles for Software Security

- Least privilege
- Failing securely
- Securing the weakest link
- Defense in depth
- Separation of privilege
- Economy of mechanism
- Least common mechanism
- Reluctance to trust
- Never assuming that your secrets are safe
- Complete mediation
- Psychological acceptability
- Promoting privacy

The Principle of Least Privilege

Only the minimum necessary rights should be assigned to a subject[10] that requests access to a resource and should be in effect for the shortest duration necessary (remember to relinquish privileges). Granting permissions to a user beyond the scope of the necessary rights of an action can allow that user to obtain or change information in unwanted ways. In short, careful delegation of access rights can limit attackers' ability to damage a system.

The Principle of Failing Securely

When a system fails, it should do so securely. This behavior typically includes several elements: secure defaults (the default is to deny access); on failure, undo changes and restore the system to a secure state; always check return values for failure; and in conditional code/filters, make sure that a default case is present that does the right

10. The term *subject* used throughout the Principle descriptions is intended to denote a user, process, or other entity acting on the software system.

thing. The confidentiality and integrity of a system should remain unbreached even though availability has been lost. During a failure, attackers must not be permitted to gain access rights to privileged objects that are normally inaccessible. Upon failing, a system that reveals sensitive information about the failure to potential attackers could supply additional knowledge that threat actors could then use to create a subsequent attack. Determine what may occur when a system fails and be sure it does not threaten the system.

The Principle of Securing the Weakest Link

Attackers are more likely to attack a weak spot in a software system than to penetrate a heavily fortified component. For example, some cryptographic algorithms can take many years to break, so attackers are unlikely to attack encrypted information communicated in a network. Instead, the endpoints of communication (e.g., servers) may be much easier to attack. Knowing when the weak spots of a software application have been fortified can indicate to a software vendor whether the application is secure enough to be released.

The Principle of Defense in Depth

Layering security defenses in an application can reduce the chance of a successful attack. Incorporating redundant security mechanisms requires an attacker to circumvent each mechanism to gain access to a digital asset. For example, a software system with authentication checks may prevent intrusion by an attacker who has subverted a firewall. Defending an application with multiple layers can eliminate the existence of a single point of failure that compromises the security of the application.

The Principle of Separation of Privilege

A system should ensure that multiple conditions are met before it grants permissions to an object. Checking access on only one condition may not be adequate for enforcing strong security. If an attacker is able to obtain one privilege but not a second, the attacker may not be able to launch a successful attack. If a software system largely consists of one component, however, it will not be able to implement a scheme of having multiple checks to access different components. Compartmentalizing software into separate components that require multiple checks for access can inhibit an attack or potentially prevent an attacker from taking over an entire system.

The Principle of Economy of Mechanism

One factor in evaluating a system's security is its complexity. If the design, implementation, or security mechanisms are highly complex, then the likelihood that security vulnerabilities will exist within the system increases. Subtle problems in complex systems may be difficult to find, especially in copious amounts of code. For instance, analyzing the source code that is responsible for the normal execution of a functionality can be a difficult task, but checking for alternative behaviors in the remaining code that can achieve the same functionality may prove even more difficult. Simplifying design or code is not always easy, but developers should strive for implementing simpler systems when possible.

The Principle of Least Common Mechanism

Avoid having multiple subjects share those mechanisms that grant access to a resource. For example, serving an application on the Internet allows both attackers and users to gain access to the application. In this case, sensitive information might potentially be shared between the subjects via the same mechanism. A different mechanism (or instantiation of a mechanism) for each subject or class of subjects can provide flexibility of access control among various users and prevent potential security violations that would otherwise occur if only one mechanism were implemented.

The Principle of Reluctance to Trust

Developers should assume that the environment in which their system resides is insecure. Trust—whether it is extended to external systems, code, or people—should always be closely held and never loosely given. When building an application, software engineers should anticipate malformed input from unknown users. Even if users are known, they are susceptible to social engineering attacks, making them potential threats to a system. Also, no system is ever 100 percent secure, so the interface between two systems should be secured. Minimizing the trust in other systems can increase the security of your application.

The Principle of Never Assuming That Your Secrets Are Safe

Relying on an obscure design or implementation does not guarantee that a system is secure. You should always assume that an attacker can obtain enough information about your system to launch an attack. For example, tools such as decompilers and disassemblers may allow attackers to obtain sensitive information that may be stored in binary files. Also, insider attacks, which may be accidental or malicious, can lead to security exploits. Using real protection mechanisms to secure sensitive information should be the ultimate means of protecting your secrets.

The Principle of Complete Mediation

A software system that requires access checks to an object each time a subject requests access, especially for security-critical objects, decreases the chances that the system will mistakenly give elevated permissions to that subject. By contrast, a system that checks the subject's permissions to an object only once can invite attackers to exploit that system. If the access control rights of a subject are decreased after the first time the rights are granted and the system does not check the next access to that object, then a permissions violation can occur. Caching permissions can increase the performance of a system, albeit at the cost of allowing secured objects to be accessed.

The Principle of Psychological Acceptability

Accessibility to resources should not be inhibited by security mechanisms. If security mechanisms hinder the usability or accessibility of resources, then users may opt to turn off those mechanisms. Where possible, security mechanisms should be transparent to the users of the system or, at most, introduce minimal obstruction. Security mechanisms should be user friendly to facilitate their use and understanding in a software application.

The Principle of Promoting Privacy

Protecting software systems from attackers who may obtain private information is an important part of software security. If an attacker breaks into a software system and steals private information about a vendor's customers, then those customers may lose their confidence in the software system. Attackers may also target sensitive system information that can supply them with the details needed to attack that system. Preventing attackers from accessing private information or obscuring that information can alleviate the risk of information leakage.