Design principles for secure systems

Lecturer: Professor Fred B. Schneider

Lecture notes by Lynette I. Millett
Revised by Borislav Deianov
Revised by Michael Frei

The classic treatment of design principles for secure systems is The Protection of Information in Computer Systems by Saltzer & Schroeder, *Proceedings of the IEEE, 63*, 9 (Sept 1975), 1278--1308. After 25 years, this paper remains a gem.

The presentation here also borrows from Computer Security in the Real World by Butler Lampson, *IEEE Computer 37*, 6 (June 2004), 37--46.

Today, we begin our discussion of security. What properties are we interested in and what are good strategies in implementing security?

General Security Issues

We first consider what we can learn from "real world" security. In the real world, security decisions are based on three things:

- value,
- locks, and
- police

We try to buy good enough locks so that the "bad guys" can't break in too often. The terms "good enough," "break in" and "too often" are key. We also assume that the police and courts work, so "bad guys" are caught and punished. "Police" in this context is a generic term for any agency that might pursue offenders; it includes the corporate hierarchy or the legal system. Similarly, the "bad guys" could be anyone, anywhere, including system operators for the system being secured. By "often enough" we don't mean *always* but enough so that crime doesn't pay. In other words, the expected gain from committing a crime must be negative. Value is an important aspect of this characterization, because generally we do not protect things of little value.

A constraint we place on any security mechanism is that it add a minimum amount of interference to daily life. For example, locks must not be difficult or annoying to use. If they are, it's likely that people will find ways to circumvent the annoyance and thus nullify the security protect0ions the locks offer. It should also be noted that with rare exceptions is a security breach of some sort the end of the world. Risk management allows recovery from a security problem and decreases the need for complex and annoying locks. For example, rather than installing a complicated locking system for automobiles we buy auto insurance to help deal with costs that arise in the event of damage or theft.

Externalities also have a role to play. Briefly, an externality occurs when somebody or some agency does something in which the cost implications for the doer are not the same as (usually significantly less than) the cost

implications for society. For example, think of companies that pollute the environment. The cost of cleaning pollution is usually great, and until recently there was no corporate penalty for *not* fixing a pollution problem. In short, an externality exists when it is cheaper to do the wrong thing. This has obvious large implications for security--an insecure subsystem may enable a system wide attack of great consequence.

There are number of things to observe. First, note that all locks are not the same. They typically have different keys as well as different strengths. The strength of the lock tends to be chosen according to the value of what is being protected. The environment also influences the type and strength of the locks being used as well. For example, apartments in Ithaca likely have fewer and weaker locks than apartments in Manhattan. Second, people pay for security they believe they need. Security is not monolithic and there is not one mechanism for everyone. Security is scaled with respect to both the value of the thing being secured and the threat against it. People's security "needs" are usually based on the perception of what's going on around them. If your neighbors are being broken into, then it's likely that you'll buy more security equipment than if not. Third, the police are central to the picture. The system still works even if locks are completely removed. Locks are only a deterrent; however, it is essential that there be enforcement and punishment strategies in place. There will undoubtedly be some security breaches no matter how good the locks are. Thus, it is critical that bad guys be found. Locks reduce temptation as well as reducing the police workload. Finally, security, as we have portrayed it, is holistic. It is only as good as its weakest link. Attackers will look for the weakest link, and thus it is generally best to expend effort in determining where the weaknesses are and shoring them up. Given limited resources, the best approach is to make all elements equally strong, thus eliminating weakest links.

Applications to Computer Security

We now move from an abstract discussion of security in our day-to-day lives to the world of computer security. How can the above discussion be applied in this new context? With regard to computer security, the story is told in terms of three terms:

- *Vulnerability*: A weakness that can be exploited to cause damage.
- *Attack*: A method of exploiting a vulnerability.
- *Threat*: A motivated, capable adversary that mounts attacks.

Bugs in a software system are vulnerabilities. Since we are not really good at building large systems, it seems clear that any large software system will have many vulnerabilities. While a first strategy for addressing a security problem might be to find and fix each vulnerability, in fact, this is likely to be too costly to be practical. Rather, it is better to first identify *threats*, and then work on eliminating only those vulnerabilities that those threats would exploit.

As an example, consider the problem of intercepting cellular phone transmissions. This possibility is clearly a result of a design vulnerability--a consequence of the way cellular phone signals are encoded and transmitted. A threat that exploits this vulnerability would be the small number of people who want to do this and have the knowledge and equipment to intercept transmissions. When cell phones were first introduced, the equipment was hard to come by and few people had the knowledge to mount an attack. Thus, the threat was small. Currently, just about anyone can buy the equipment; the threat is huge. The vulnerability has remained the same, but the nature of the threat has changed. Currently, there is a large amount of cellular-phone fraud.

Range of Threats

What are the range of threats that network information systems face? The Defense Science Board has issued a report that includes their view of current threats to the national infrastructures. Their list, in order of increasing severity, is as follows:

- Incomplete, inquisitive and unintentional blunders.
- Hackers driven by technical challenges.
- Disgruntled employees or customers seeking revenge.
- Criminals interested in personal financial gain or stealing services.
- Organized crime with the intent of hiding something or financial gain.
- Organized terrorist groups attempting to influence U.S. policy by isolated attacks.
- Foreign espionage agents seeking to exploit information for economic, political, or military purposes.
- Tactical countermeasures intended to disrupt specific weapons or command structures.
- Multifaceted tactical information warfare applied in a broad orchestrated manner to disrupt a major U.S. military mission.
- Large organized groups or nation-states intent on overthrowing the United States.

Note that as we go down the list, more and more resources are presumed available for attacks. It seems, then, that the severity of a threat is related to the resources available. On the other hand, available resources will also include knowledge. Cash can buy knowledge, of course, but these days attacks can be packaged and broadly distributed so that considerably less knowledge is needed to launch an attack. For example, SATAN is a package used to identify breaches of security in UNIX systems and break into them. Ostensibly this package is for UNIX system administrator's to use to aid in securing their own systems. However, its wide availability on the Internet means that anyone can use it. Note also that the existence of such packaged software means there is a discontinuity between the technical expertise of a particular threat (such as the blunderers) and the severity of the problem they can cause with access to methods produced by those at the higher end of the scale. This is troubling.

What are we trying to protect?

Another concern when building a secure system is the identification of what exactly we're trying to protect. With network information systems, common properties we worry about are:

- *secrecy and confidentiality*: improper disclosure of information
- *integrity*: improper alteration of data
- *availability*: a service should be there when it is sought

Secrecy and confidentiality are subtle problems. It is difficult to define what constitutes a secret, and privacy is a social question. Raw data, such as a web browser history listing for some particular user, is not necessarily a secret. However, the *interpretation* of that data could be information that the user wishes to keep private. The *aggregation* and *disaggregation* of information can also make a secret. For example, knowing the number of troops in a US military base may not necessarily be useful or interesting, but to a neighboring, hostile country, knowing the US troop strength for a whole region may not be something we'd like to keep secret. This is known as aggregation. Conversely, an example of disaggregation would be a group of terrorists planning an attack on a particular US base. The troop strength of a region is probably not important, but the number of troops at that base might be something we would want to obscure from them. It depends on the context.

Attributes of Computer Security

How do locks, value, and police make sense in connection with computer security? Locks can be thought of as authorization or access control mechanisms. A "key" or authentication is generally required to open a lock. This "key" can be something the user knows, has, or is. For example, the user might know a password or possess an ID card. Retinal scans and fingerprint verification would take advantage of something that the user *is*. The police for computer security are the same as the "real world" police. The difference is that in the realm of computers, attacks can be launched remotely, and thus the equivalent of video cameras (to help the police and courts convict the offender) will be necessary. We call this equivalent an *audit facility*. Noting the chemical symbol, Au, for gold, Butler Lampson has proposed the "Gold Standard of Security" consisting of

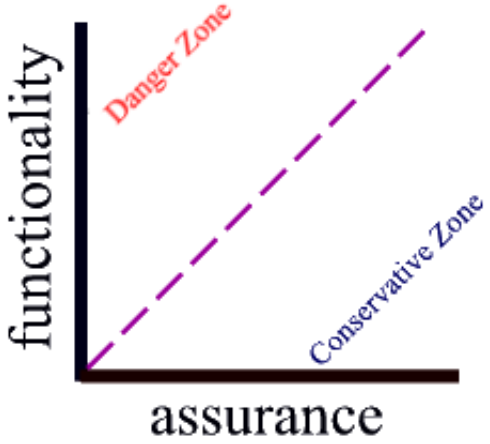
- **au**thorization,
- **au**thentication, and
- **au**diting.

Any system purporting to be secure will contain these elements. Note that, as before, locks are only the first line of defense. Security cannot work if locks are the *only* protection, however. Nevertheless, it is instructive to contemplate what are good general attributes of a lock. What should it do? How should we evaluate it?

Assurance

Assurance is concerned with the question of what do you know and how do you know it? It is not enough to know that a system is trustworthy or secure; we must also convince ourselves that this will be the case. A system satisfies *functional requirements* if it does what it is suppose to. In other words, inputs produce the correct outputs. A system satisfies *non-functional requirements* if, for some specified set of contexts (depends on the threat), the system doesn't do more or less than its functional requirement. Not "more" because that might mean revealing secrets; not "less" because that might mean failure to render a service.

There is an annoying tradeoff between functionality and assurance. A system that supports increased functionality is likely to be more complex, hence harder to understand, analyze or test. A system with lower functionality can be simpler, easier to understand, analyze, and text. One must choose a good point in the functionality vs. assurance space



Two general principles to promote higher assurance are:

- **Economy of Mechanism**. Use small and simple mechanisms wherever possible. Bells and whistles add complication, and that introduces errors.

- **Open Design**. The security of a mechanism should not depend on attacker's ignorance of how the mechanism works or is built. (This is basically a corollary of Kerckhoff's 1883 principle that the design of a cryptosystem not depend on keeping the algorithm secret.)
  The principle of open design is controversial. Showing a design or source code to attackers certainly makes their task easier. Someone could find a vulnerability, not inform the designers, and attempt to exploit it or sell the information later. NSA, for instance, refused to publish their analysis that the DES encryption algorithm is secure. They argued that publicizing such information could make it easier to break the cryptosystem. The lack of information, however, decreases people's assurance in the security of DES.

Functionality

What should a lock do? There is a sense in which "less is more". System builders must design systems that support the needs of their users. This means that one should strive to define, design, and build mechanisms that support a wide variety of policies. Mechanisms that support a small number of fixed policies are a bad idea, for they impose restrictions on their users. We speak of separating "policy" from "mechanism" and desire mechanisms that do not exert a policy bias.

A general principle for security policies is:

- **Principle of Least Privilege**. Every program and every user should operate using the minimum set of privileges necessary to complete the job.

Practicing the Principle of Least Privilege limits the damage that can result from an accident or error, and it limits the number of privileged programs (which could be compromised) in the system. It also helps in debugging, is good for increasing assurance, and allows isolation of small critical subsystems. Instantiating the Principle of Least Privilege does raise two questions:

1. What defines the "minimum set of privileges"?
2. When does this set change?

Implicit in the Principle of Least Privilege is

- **Complete Mediation**. Every access to every object is checked.

Thus, some interfaces must be monitored---the interface might be defined in terms of method calls or it might be defined in terms of individual instructions. Observe also that the integrity of a complete mediation mechanism must be protected.

Corollaries of the Principle of Least Privilege include:

- **Separation of Privilege**. Two keys are better than one. Each privilege should require a separate secret. The implication being: separate passwords for separate objects. This allows fine-grained control of access

to the system, and limits what can be compromised if a single secret is revealed.

- **Failsafe Defaults**: No access by default. It is much better (and less prone to error) to define who *can* have access that to directly define who *cannott*.

Separation of Privilege can turn into a user's nightmare, with many different access rights to set or unset. Failsafe Defaults also can be problematic, with users unable to get their work done because they need access to all sorts of objects (tmp files, etc) that they do not realize are needed.

Where to Deploy Locks

Every system and every lock is likely to have vulnerabilities. Different locks are not likely to have the same vulnerabilities, though. We thus have two more useful principles.

- **Diversity of Mechanism** Diverse mechanisms are unlikely to share vulnerabilities.

- **Multiple Lines of Defense** If multiple lines of defense are employed, the odds are increased that no single vulnerability is common to all.

There are now several possible strategies for implementing a secure system:

1. Keep everyone out. If there is no sharing, then there is no need to worry about attacks. This technique is used for single-user machines. The problem is that sharing is a good way to communicate and get things done.

2. Keep the bad guys out. There are technologies that facilitate this. Firewalls can be used to act as a gateway and filter information going in and out of a subnetwork. Similarly, code signing is used by web browsers to allow restrictions on what code can be run. The problem here is that if a firewall is penetrated or a code signature is stolen, security is compromised.

3. Let the bad guys in, but prevent them from doing damage. There are various techniques for this. Most internal system security mechanisms we discuss do exactly this. Think of standard operating systems, which place restrictions on which processes have which privileges.

4. Let the bad guys do anything, keep track of what happens, catch the bad guys and then prosecute them. This is facilitated through the use of good audit facilities.

Notice that the latter three strategies of the four possible strategies (1)-(4) require being able to distinguish the bad guys from the good guys, and that means some sort of authentication process. Keeping the bad guys out requires having a list of the good guys. Letting the bad guys in but preventing them from doing harm requires not only a list of the good guys, but an authorization mechanism as well. This authorization can be done at many levels of the system. For example, read/write/execute privileges in a UNIX system are done at the file level. Authorization could also be done at the level of fields in data records. The closer to the application level, the greater the possibility of authorizing precisely what is required. However, if done at a higher level, it is possible for the success of the mechanism to become application dependent. At a lower level, authorization is likely to be simpler. The less complex such facilities are, the fewer bugs there will be. Thus lower level authorizations can provide higher assurance. This is a classic trade-off: functionality vs. assurance. The fourth option above depends heavily on audit facilities that cannot be subverted by the bad guys. Thus, audit requires authorization.