

Principles of Secure Software

Wesley McGrew
wesley@mcgrewsecurity.com

The Protection of Information in Computer Systems

Jerome H. Saltzer, Michael D. Schroeder

Originally presented at the Fourth ACM Symposium
on Operating System Principles in 1973

...contains, among other things, principles for
developing more secure software



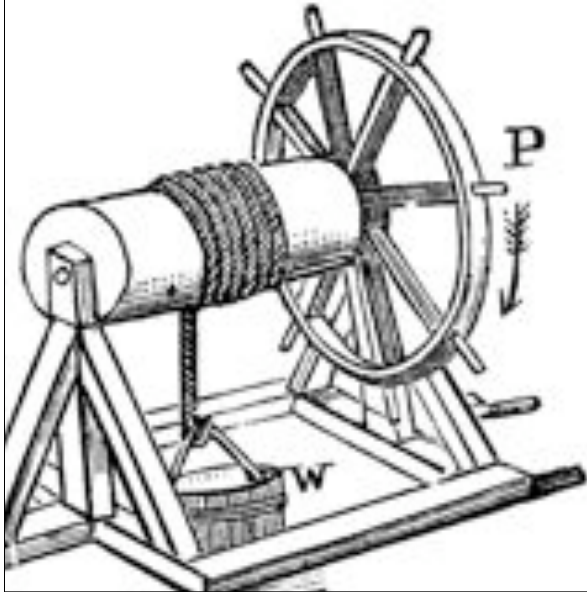
Paper available at : <http://www.cs.virginia.edu/~evans/cs551/saltzer/>

Highly recommended for the principles we'll be discussing today. Also, the opening glossary is very nice, if you're new to security and want to be able to hold your own in a conversation. Many of the concepts discussed in this presentation are covered in much more detail in the paper, but with a great deal of theory mixed with the hardware/software of the time the paper was published. Because of this, it's up to you as a reader to apply these principles to whatever modern software development framework, language, and/or environment you might be using.

The technology of how systems are attacked and the defenses against these attacks changes, however most issues can be boiled-down to this set of fundamental principles.

(Economy of Mechanism)

“Keep the design as simple and small as possible”



This is sometimes referred to as the KISS (Keep it Simple, Stupid) principle. From context, it can be determined whether "Stupid" refers to a lack of complexity in code, or the implementor.

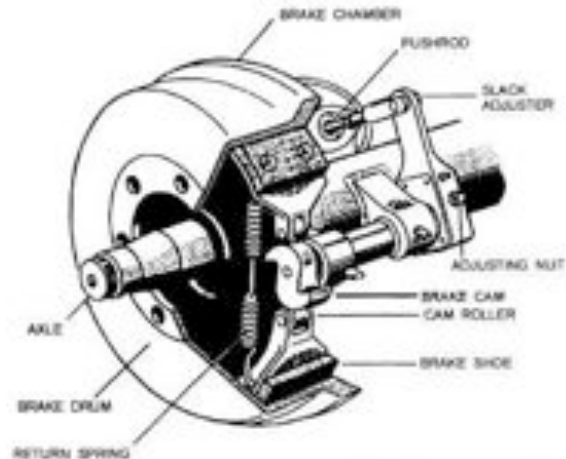
There's no shortage of "real world" examples of this. Mechanical pencils, as simple as they are, jam and get fouled up much more often than wood and lead pencils. Cars, once repairable under a shade tree with simple tools, often require computers and expensive equipment to diagnose and fix. There are many positive and convenient features that this complexity can bring, however it results in adding points-of-failure. This can decrease reliability, and increase the cost of diagnosis and repair.

By making each module or component of a software package as small and well-defined as possible, it becomes much easier to determine whether or not the implementation is correct. This is extremely important in security-critical portions of a program, where each additional line of code has the potential to become another avenue of attack.

How might this apply to filtering user input in a web application?

(Fail-Safe Defaults)

“Base access decisions on permission rather than exclusion”



When things break (and they will break), you often have a choice:

- * Do you fail "open", in a way that might allow work to continue, however it leaves you open for attack
- * ...or do you fail "safe", in a way that halts work, but keeps your information safe

In action movies, occasionally there will be a scene where a large truck is out of control, and the driver cannot stop it because the air lines to the "air brakes" have been cut. In this case, the brakes have failed "open": The truck is still operable, though in a very unsafe state.

In reality, air brakes use air pressure to keep the brakes *off*. If the lines are cut, the pressure drops, and the brakes engage, regardless of the driver's actions. This may not be a pleasant experience, but it's overall a lot safer. This is failing "safe".

In software, what information do you provide to users in error messages? What are your default permissions for new objects?

(Complete Mediation)

"Every access to every
object must be checked
for authority"



Your passport is checked upon entry to the country, to verify that you allowed to be enter. This is done at many entry points: planes, border crossings, etc. A "bad guy" without a valid and accepted passport would need to find some other, more obscure, way into the country, where he wouldn't be checked. If someone becomes a "bad guy" *after* being allowed entry with a valid passport, then the task of stopping that person becomes much more difficult.

The same concept applies to software. Imagine a web application, where it's trivial for a user to simply type in a URL that goes directly to a portion of the application that would normally be behind a password. Does your application check to make sure the user's session is valid on each access to that resource? Does it assume that the user is navigating the site normally, and therefore wouldn't be at that location if they didn't already have access?

What happens in *your* application if a user's access credentials are revoked by an admin in the middle of a session?



It should be said, right upfront, that this has little to do with the open-source movement, or the supposed security benefits of open licenses (GPL, BSD, etc.) have.

The question is: Does the security of this system rely on the attacker not knowing details of its implementation?

If a burglar knows your key is hidden in a fake rock, then it's relatively easy for them to find. If your web application has a `/top_secret/logs/` directory, and you assume no one will ever find it, you're probably mistaken. If you're afraid that an attacker could break your encryption program if they gained access to its source code, it's probably not that good.

Bad guys are a lot better at reverse-engineering things than you are (and likely better than you would imagine).

There are always secrets involved when it comes to secure software, but it's far better for those secrets to be passwords, cryptographic keys, or hardware tokens that are more easily protected.

(Separation of Privilege)

“Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.”



This Swiss safe-deposit box lock is neat. It's described here:

<http://swiss-banking-antiques.com/e/shadow-boxes/combination/lock.html>

Besides being very attractive, the functionality is interesting to us. A number of levers holds the lock closed. To open it, these conditions have to be met:

1. A bank worker's key has to be inserted and turned a half-turn clockwise, then removed
2. The client's key has to be inserted
3. The 4 alphabetic dials are turned to a secret code
4. The client's key is turned 1/4 turn
5. The dials are then scrambled by the client
6. The client's key is turned another 1/4 turn, opening the lock

At least two people are required to open this lock, maybe three if two clients split up the key and code.

Think of how this might apply to situations where you might want the approval of more than one admin to change a critical file, or remove a user.

(Least Privilege)

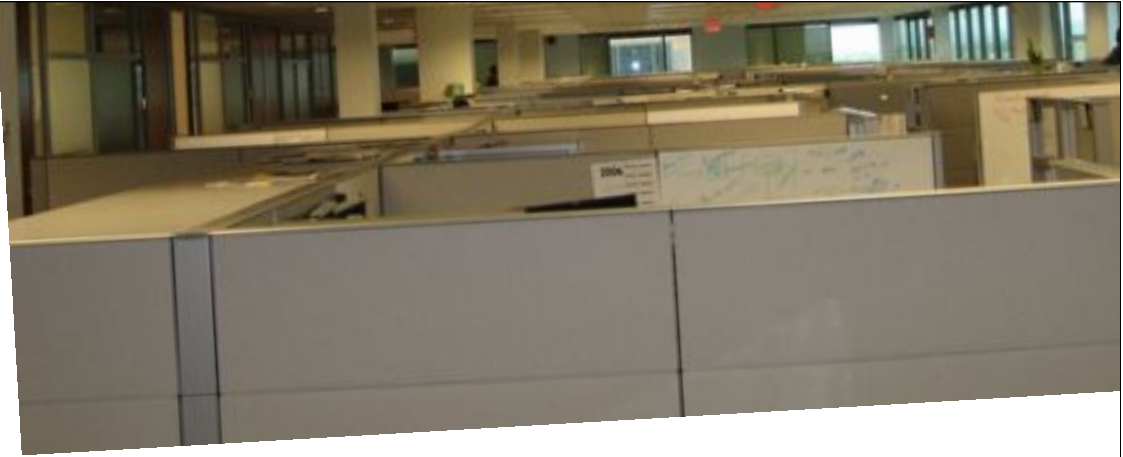
“Every program and every user of the system should operate using the least set of privileges necessary to complete the job.”



This is one of the most important principles, I think.

If the code you are running requires a certain set of privileges, say for example: the ability to access a hardware device directly, only give that code the minimum permissions necessary to do what it needs to do. It shouldn't access other devices. It shouldn't have "root" access to everything on the file system. It should be "sandboxed" to exactly what it needs, in much the same way compartmentalization works in the government, with levels of classification and "need-to-know".

This also relates to the economy of mechanism: by keeping security-critical portions of your software small and modular, you are reducing the amount of code that has to run with elevated privileges.



(Least Common Mechanism)

“Minimize the amount of mechanism common to more than one user and depended on by all users”

Shared resources invite subtle attacks. Much like cubicle walls separate tiny little islands of independent workspace, isolating your users from each other on a shared computer system will improve the confidentiality and security of each users actions.

If more than one user is sharing the use of a software library or resource that keeps its own variables, or state, then there is the potential that each user could monitor the actions of the others by inspecting that state. This is especially bad when it comes to the generation of random numbers for secrets, or other cryptographic routines. Race conditions can also arise, where one user can take control of a resource (say, a file being used for temporary storage) before another user uses it.

(Psychological) Acceptability)

“It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.”



The picture says it all. If the users of your software find your security mechanisms to be more of a pain than they're worth, then they will do everything they can to bypass them.

What can you do? :

- * Educate your users on how serious the threat is (not likely to always work)
- * Secure defaults
- * Ease of use
- * Automation
- * Policy and punishment (getting to the last resorts here...)



vs



(Work Factor)

"Compare the cost of circumventing the measure with the resources of a potential attacker."

There are many variables that determine whether or not you will be attacked, if that attack will be successful, and what the impact will be:

- * What is the value of the information you're protecting, to the people you're protecting it from?
- * To whom is this information valuable?
- * How long would it take a skilled attacker to successfully break the system?
- * $\text{Value} > \text{skilled_attacker_cost} * \text{time}$?

Don't cut these calculations too close, as it's very easy to underestimate the skill of your opponent, or the value of the data you're protecting.

Design security mechanisms that raise the bar as high as you feasibly can, while still fitting the psychological acceptability.

(Compromise) Recording

“It is sometimes suggested that mechanisms that reliably record that a compromise of information has occurred can be used in place of more elaborate mechanisms that completely prevent loss.”



As we discussed before, things will break. Attackers will be successful.

Tamper-evident seals on pill-bottles don't prevent people from poisoning the drugs that are in the bottle, they simply make it very difficult to do without obviously damaging the seal. If you purchased a bottle, took it home, and noticed that the seal was broken, you wouldn't take the medicine.

In the same way, you want to know when your software security has been compromised. Good, off-system logging, alerts, and other protection mechanisms can be used to help raise some alarms when an attacker is successful, and hopefully log enough forensic data to find determine what happened, and who is responsible.

Questions and Conclusions

How does this apply to projects you've worked on here
in the past?

How is this relevant to the ways information is stored
and transmitted (network, volatile memory, persistent
storage)?