University of BRISTOL

COMS20012: The Protection of Information in Computer Systems

Joseph Hallett

bristol.ac.uk

So what is the point of an OS, again?

Operating systems give a mechanism to protect the *confidentiality integrity* and *availability* of data on a computer

- OS stands between the user of the computer and the underlying resources, and enforces a policy

So how do you design the OS (and systems in general) to protect information in a system?

related to hazard from lasers and other light sources," *Amer. J. Ophthalmol.*, vol. 66, p. 15, 1968.
[57] A. Vassiliadis, H. C. Zweng, N. A. Peppers, R. R. Peabody, and R. C. Honey, "Thresholds of laser eye hazards," *Arch. Environ. Health*, vol. 20, p. 161, 1970.
[58] P. W. Lappin, "Ocular damage thresholds for the helium–neon laser," *Arch. Environ. Health*, vol. 20, p. 177, 1970.
[59] W. T. Ham *et al.*, "Retinal burn thresholds for the He-Ne laser in the rhesus monkey," *Arch. Ophthalmol.*, to be published.
[60] T. P. Davis and W. J. Mautner, "Helium-neon laser effects on the eye," U.S. Army Med. Res. Develop. Com., Washington, D.C., Annu. Rep. Contr. DADA 17-69-C-9013, 1969.
[61] J. J. Vos, "Digital computations of temperature in retinal burn problems," Inst. Perception, Soesterberg, The Netherlands, RVO-TNO, Rep. IZF 1965016, 1963.
[62] M. A. Mainster, T. J. White, J. H. Tips, and P. W. Wilson, "Retinal-temperature increases produced by intense light sources," *J. Opt. Soc. Amer.*, vol. 60, p. 264, 1970.
[63] A. M. Clarke, W. T. Ham, W. J. Geeraets, R. C. Williams, and H. A. Mueller, "Laser effects on the eye," *Arch. Environ. Health*, vol. 18, p. 424, 1969.
[64] R. H. Stern and R. F. Sognnaes, "Laser beam on dental hard tissues," *J. Dent. Res.*, vol. 43, p. 873, 1964.
[65] R. H. Stern, "Dentistry and the laser," in *Laser Applications in Medicine and Biology*, vol. II, Dr. M. L. Wolbarsht, Ed. New York: Plenum, 1974, pp. 361–388.
[66] T. F. Gordon, Jr., and D. L. Smith, "Laser welding of prostheses—an initial report," *J. Prosth. Dent.*, vol. 24, p. 472, 1970.

The Protection of Information in Computer Systems

JEROME H. SALTZER, SENIOR MEMBER, IEEE, AND MICHAEL D. SCHROEDER, MEMBER, IEEE

Invited Paper

Abstract—This tutorial paper explores the mechanics of protecting computer-stored information from unauthorized use or modification. It concentrates on those architectural structures—whether hardware or software—that are necessary to support information protection. The paper develops in three main sections. Section I describes desired functions, design principles, and examples of elementary protection and authentication mechanisms. Any reader familiar with computers should find the first section to be reasonably accessible. Section II requires some familiarity with descriptor-based computer architecture. It examines in depth the principles of modern protection architectures and the relation between capability systems and access control list systems, and ends with a brief analysis of protected subsystems and protected objects. The reader who is dismayed by either the prerequisites or the level of detail in the second section may wish to skip to Section III, which reviews the state of the art and current research projects and provides suggestions for further reading.

GLOSSARY

THE FOLLOWING glossary provides, for reference, brief definitions for several terms as used in this paper in the context of protecting information in computers.

Access	The ability to make use of information stored in a computer system. Used frequently as a verb, to the horror of grammarians.
Access control list	A list of principals that are authorized to have access to some object.
Authenticate	To verify the identity of a person (or other agent external to the protection system) making a request.
Authorize	To grant a principal access to certain information.
Capability	In a computer system, an unforgeable ticket, which when presented can be taken as incontestable proof that the presenter is authorized to have access to the object named in the ticket.
Certify	To check the accuracy, correctness, and completeness of a security or protection mechanism.
Complete isolation	A protection system that separates principals into compartments between which no flow of information or control is possible.
Confinement	Allowing a borrowed program to have access to data, while ensuring that the program cannot release the information.
Descriptor	A protected value which is (or leads to) the physical address of some protected object.
Discretionary	(In contrast with *nondiscretionary*.) Controls on access to an object that may be changed by the creator of the object.
Domain	The set of objects that currently may be directly accessed by a principal.
Encipherment	The (usually) reversible scrambling of data according to a secret transformation key, so as to make it safe for transmission or storage in a physically unprotected environment.
Grant	To authorize (*q.v.*).
Hierarchical control	Referring to ability to change authorization, a scheme in which the record of

Manuscript received October 11, 1974; revised April 17, 1975.
The authors are with Project MAC and the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Mass. 02139.

Back in 1975…

Saltzer and Schroeder created a catalogue of design principles for protecting information in computer systems.

It is a classic paper, that's stood the test of time.

You **need** to read it (*that is I'm very likely to set an exam question on it ;-)*)

Saltzer & Schroeder Design Principles

1. Economy of mechanism
2. Fail-safe defaults
3. Complete mediation
4. Open design
5. Separation of privilege
6. Least privilege
7. Least common mechanism
8. Psychological acceptability
9. Work factor
10. Compromise recording

Economy of Mechanism

a) Economy of mechanism: Keep the design as simple and small as possible. This well-known principle applies to any aspect of a system, but it deserves emphasis for protection mechanisms for this reason: design and implementation errors that result in unwanted access paths will not be noticed during normal use (since normal use usually does not include attempts to exercise improper access paths). As a result, techniques such as line-by-line inspection of software and physical examination of hardware that implements protection mechanisms are necessary. For such techniques to be successful, a small and simple design is essential.

Fail-safe Defaults

b) Fail-safe defaults: Base access decisions on permission rather than exclusion. This principle, suggested by E. Glaser in 1965,[8] means that the default situation is lack of access, and the protection scheme identifies conditions under which access is permitted. The alternative, in which mechanisms attempt to identify conditions under which access should be refused, presents the wrong psychological base for secure system design. A conservative design must be based on arguments why objects should be accessible, rather than why they should not. In a large system some objects will be inadequately considered, so a default of lack of permission is safer. A design or implementation mistake in a mechanism that gives explicit permission tends to fail by refusing permission, a safe situation, since it will be quickly detected. On the other hand, a design or implementation mistake in a mechanism that explicitly excludes access tends to fail by allowing access, a failure which may go unnoticed in normal use. This principle applies both to the outward appearance of the protection mechanism and to its underlying implementation.

Complete Mediation

c) Complete mediation: Every access to every object must be checked for authority. This principle, when systematically applied, is the primary underpinning of the protection system. It forces a system-wide view of access control, which in addition to normal operation includes initialization, recovery, shutdown, and maintenance. It implies that a foolproof method of identifying the source of every request must be devised. It also requires that proposals to gain performance by remembering the result of an authority check be examined skeptically. If a change in authority occurs, such remembered results must be systematically updated.

Open Design

d) Open design: The design should not be secret [27]. The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, more easily protected, keys or passwords. This decoupling of protection mechanisms from protection keys permits the mechanisms to be examined by many reviewers without concern that the review may itself compromise the safeguards. In addition, any skeptical user may be allowed to convince himself that the system he is about to use is adequate for his purpose.[9] Finally, it is simply not realistic to attempt to maintain secrecy for any system which receives wide distribution.

Open Design (Footnote)

[9] We should note that the principle of open design is not universally accepted, especially by those accustomed to dealing with military security. The notion that the mechanism not depend on ignorance is generally accepted, but some would argue that its design should remain secret. The reason is that a secret design may have the additional advantage of significantly raising the price of penetration, especially the risk of detection.

Separation of privilege

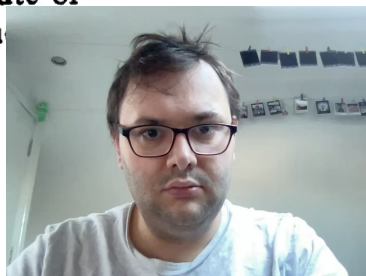
e) Separation of privilege: Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key. The relevance of this observation to computer systems was pointed out by R. Needham in 1973. The reason is that, once the mechanism is locked, the two keys can be physically separated and distinct programs, organizations, or individuals made responsible for them. From then on, no single accident, deception, or breach of trust is sufficient to compromise the protected information. This principle is often used in bank safe-deposit boxes. It is also at work in the defense system that fires a nuclear weapon only if two different people both give the correct command. In a computer system, separated keys apply to any situation in which two or more conditions must be met before access should be permitted. For example, systems providing user-extendible protected data types usually depend on separation of privilege f their implementation.

Least Privilege

f) Least privilege: Every program and every user of the system should operate using the least set of privileges necessary to complete the job. Primarily, this principle limits the damage that can result from an accident or error. It also reduces the number of potential interactions among privileged programs to the minimum for correct operation, so that unintentional, unwanted, or improper uses of privilege are less likely to occur. Thus, if a question arises related to misuse of a privilege, the number of programs that must be audited is minimized. Put another way, if a mechanism can provide "firewalls," the principle of least privilege provides a rationale for where to install the firewalls. The military security rule of "need-to-know" is an example of this principle.

Least common mechanism

g) Least common mechanism: Minimize the amount of mechanism common to more than one user and depended on by all users [28]. Every shared mechanism (especially one involving shared variables) represents a potential information path between users and must be designed with great care to be sure it does not unintentionally compromise security. Further, any mechanism serving all users must be certified to the satisfaction of every user, a job presumably harder than satisfying only one or a few users. For example, given the choice of implementing a new function as a supervisor procedure shared by all users or as a library procedure that can be handled as though it were the user's own, choose the latter course. Then, if one or a few users are not satisfied with the level of certification of the function, they can provide a substitute or not use it at all. Either way, they can avoid being harm
a mistake in it.



Psychological Acceptability

h) 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. Also, to the extent that the user's mental image of his protection goals matches the mechanisms he must use, mistakes will be minimized. If he must translate his image of his protection needs into a radically different specification language, he will make errors.

The next two principles…

The final two principles Saltzer and Schroeder say apply imperfectly to computer systems but have been suggested by physical security folks…

Almost 50 years later…

…Yeah they've been shown right too.

(Did I mention this work is a classic?)

Work Factor

a) Work factor: Compare the cost of circumventing the mechanism with the resources of a potential attacker. The cost of circumventing, commonly known as the "work factor," in some cases can be easily calculated. For example, the number of experiments needed to try all possible four-letter alphabetic passwords is $26^4 = 456\,976$. If the potential attacker must enter each experimental password at a terminal, one might consider a four-letter password to be adequate. On the other hand, if the attacker could use a large computer capable of trying a million passwords per second, as might be the case where industrial espionage or military security is being considered, a four-letter password would be a minor barrier for a potential intruder. The trouble with the work factor principle is that many computer protection mechanisms are *not* susceptible to direct work factor calculation, since defeating them by systematic attack may be logically impossible. Defeat can be accomplished only by indirect strategies, such as waiting for an accidental hardware failure or searching for an error in implementation. Reliable estimates of the length of such a wait or search are very difficult to make.

Compromise Recording

b) 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. For example, if a tactical plan is known to have been compromised, it may be possible to construct a different one, rendering the compromised version worthless. An unbreakable padlock on a flimsy file cabinet is an example of such a mechanism. Although the information stored inside may be easy to obtain, the cabinet will inevitably be damaged in the process and the next legitimate user will detect the loss. For another example, many computer systems record the date and time of the most recent use of each file. If this record is tamperproof and reported to the owner, it may help discover unauthorized use. In computer systems, this approach is used rarely, since it is difficult to guarantee discovery once security is broken. Physical damage usually is not involved, and logical damage (and internally stored records of tampering) can be undone by a clever attacker.[10]

As is apparent, these principles do not represent absolute rules—they serve best as warnings. If some part of a design violates a principle, the violation is a symptom of potential trouble, and the design should be carefully reviewed to be sure that the trouble has been accounted for or is unimportant.

…but…

Since no one knows how to build a system without flaws, the alternative is to rely on eight design principles, which tend to reduce both the number and the seriousness of any flaws: Economy of mechanism, fail-safe defaults, complete mediation, open design, separation of privilege, least privilege, least common mechanism, and psychological acceptability.

Finally, some protection designs can be evaluated by comparing the resources of a potential attacker with the work factor required to defeat the system, and compromise recording may be a useful strategy.

So where have we gone from here?

The Saltzer & Schroeder design principles have been shown to be classics and are the basis for an awful lot of modern security.

…but have we come up with any other principles since?

Security problems are just bugs

http://lkml.iu.edu/hypermail/linux/kernel/1711.2/01701.html

On Fri, Nov 17, 2017 at 12:35 PM, Kees Cook <keescook@xxxxxxxxxxxx> wrote:
>
> *This is why I introduced the fallback mode: with both kvm and sctp*
> *(ipv6) not noticed until late in the development cycle, I became much*
> *less satisfied it had gotten sufficient testing.*

So honestly, this is the kind of completely unacceptable "security person" behavior that we had with the original user access hardening too, and made that much more painful than it ever should have been.

IT IS NOT ACCEPTABLE when security people set magical new rules, and then make the kernel panic when those new rules are violated.

That is pure and utter bullshit. We've had more than a quarter century _without_ those rules, you don't then suddenly walz in and say "oh, everbody must do this, and if you haven't, we will kill the kernel".

The fact that you "introduced the fallback mode" late in that series just shows HOW INCREDIBLY BROKEN the series started out.

Seriously.

As a security person, you need to repeat this mantra:

"security problems are just bugs"

and you need to _internalize_ it, instead of scoff at it.