SAFETY

In the *PEOPLE : IMPLICATIONS* chapter, we identified two sorts of safety, and remarked that we wanted a reliable long-term storage device and an effective security system to provide those. More generally, just what constitutes safety in a computer system depends on the system; the criteria are different for an aircraft autopilot, a worldwide communications system, and a word-processor. In this introductory account we shall concentrate on the word-processor level. This is not in any way to deny the importance of the other sorts of system, but we have to begin somewhere, and the safety techniques which we can use for simple tasks which don't threaten life or property will give us enough to be going on with.

We are discussing the issue of safety very early in our treatment because we believe it to be very important, and we therefore believe that it should be taken into account right from the start. It is easier to achieve safety if you design your system to be intrinsically safe than if you design it to do the work and then try to make it safe afterwards. If you want a vehicle that will only travel to certain places, it's much more effective to design a train and build a railway than to design a bus and then seek some foolproof way to confine it to an arbitrary set of roads. Analogies are not sound arguments, but they can illuminate a problem, and experience suggests that this analogy is quite good.

As a final preliminary exercise, you might like to consider the curious relationship, illuminated by that analogy, between safety issues and the normal process of computing. We have stressed that the function of the operating system is to help people to get their work done as easily and straightforwardly as we can manage; surely, though, that is more likely to be achievable by a system modelled on a freely manœuvrable bus than on a constrained train ? We see that safety concerns might well run counter to our primary aims in system development.

This is certainly true. Another point of view is that to every activity carried out by a system there corresponds a danger, or an opportunity for subversion. If we have files, they can be lost or stolen; if we have communications, they can be affected by noise or intercepted; and so on. Proceeding with this argument, we see that it is easy to make an unassailably safe system by removing all the possible activities. That leaves us with a basic system called the *perfectly secure brick*.

In practice, we seek a compromise. As with all compromises, it will satisfy few people. Some will complain that their legitimate activities are being restricted; others will bewail the vulnerability of their precious files; not a few will do both. We make no attempt to resolve the dilemma here, but restrict ourselves to describing a selection of fairly standard techniques which are widely used in practical systems.

WHAT CAN GO WRONG ?

In fact, of course, we're not really discussing safety; we're discussing unsafety. If everything were safe, we wouldn't need to discuss anything. The trouble arises because things can go wrong, and do. It's sensible, therefore, to begin by trying to identify the sources of trouble, and that must be anything which prevents us from doing the computing job we want to do. We can make a rough classification like this :

WHERE THE FAULT IS	WHAT HAPPENS	CAUSE OF FAULT (see below)		
		Accident	Error	Malice
The computer, or other hardware.	Not directly our problem – but we can be careful just in case.	Power failure – work disrupted; device failure; catastrophe – system destroyed.	Tapes or discs lost or wrongly mounted; communication rates set wrongly.	Computer stolen; communications intercepted.

What's in the computer.	Should be there, but isn't.	Lost data	Delete files too soon – or "rm *".	Unauthorised destruction of files; effects of viruses.
	Shouldn't be there, but is.			Viruses
	Should be there, is there, but is wrong.	Corrupt files.		Files infected with viruses.
Access to what's in the computer.	Not enough, can't get in.	Disc fills up.	Programme fault – fills memory or disc, monopolises processor.	Worms
	Too much, can get other people's things.		Ineffective access control	Security violation

WHAT MAKES IT GO WRONG ?

Safety in the abstract is all very well, but to achieve it in practice we have to know something about the threats from which we wish to be safe. In computer systems, three sorts of threat are important – accident, error, and malice. We can describe them briefly :

- ACCIDENT : no one (identifiable) is to blame; accidents can happen at any time, without anyone noticing. When they happen, there might be no one there – so protective measures must be automatic. Accidents happen both outside the computer system (earthquake, fire, flood, etc.), and inside (disc head crashes, dirty contacts, faulty integrated circuits), but it makes little difference to the problems they present.

- ERROR : someone has done something wrong, and might reasonably be expected to cooperate in putting it right. (Though the effects of the mistake might not become evident for a long time.)

- MALICE : someone has done something wrong, and won't cooperate in protective measures.

This table illustrates the relationship between the three classes :

Intentional	Agent	
	People	Not people
no	error	accident
yes	malice	

We'll fill in the blank at the bottom right if we ever find any malicious machines; despite occasional feelings that our computers are out to get us, we're reasonably confident that it will stay blank for quite some time. So far, only people are nasty.

We think of PROTECTION against accident and error, but SECURITY against malicious attack. The distinction between these two categories is that in cases of accident and error we can reasonably expect people to cooperate in measures which will help to reduce problems, whereas in cases of malice there is at least one agent with an interest in opposing safety measures. Yet another table :

Concern :	PROTECTION	SECURITY

Guards against events which are :	Accidental	Deliberate
Can expect :	Cooperation	Opposition
Effect on victim :	You lose something you should have.	Various – might be no obvious change, but you might have lost a secret.
Effect on attacker :		You get something you shouldn't have.

DRAMATIS PERSONÆ.

Consider the structure of an action in which some sort of safety mechanism might be desirable. Generally, something acts upon something else, using the machinery (not necessarily restricted to the hardware) of the computer – which is not a big surprise, but identifies three participants in the action :

the **subject**, which takes the initiative and performs the action;
the **object**, on which the action is performed; and
the **machinery**, hardware or software, which does the work.

We use these fairly anonymous terms because participants in an operation may be of many sorts. Subjects (alternatively called *principals*) may be people, or programmes, or command files (or, in the case of accidents, pretty well anything); objects may be files, programmes, or devices; machines may be computers, devices, or operating systems. Clearly, there is much overlap between the categories – because in a computer system all entities at the software level at least are composed of much the same materials. Both a programme and its data are expressed as arrays of bits.

Subject (principal)	Machinery	Object
Active	Executive	Passive
person, programme, command file, client	computer, device, operating system	file, programme, device, message, server

As an example, consider a shared computer system which must maintain information about all those who use it in its *user database*. Various protection and security measures are necessary for the reliable operation of the user database, and these might be quite different for different items of information recorded. Three sorts of item are the login password, the search path, and the accounting information. Who or what should have access to each item, and what sort of access should be permitted ? This table shows an attempt at an answer.

Object	Subject :	owner	system	administrator
	Access type :	write (NOT read)	check (NOT read)	write (NOT read)

Password	Reason :	Regular changes are desirable; only the owner should know the password; read access is not permitted to ensure that people can't find the owner's password from an unattended terminal.	The login software must be able to check on logging in. Without a check, the password is useless. The check should not return the password, but merely check a string.	The user registration administrator must be able to establish initial passwords; there must be some way to cope with forgotten passwords.
Search path	Access type :	read, write	read	none
	Reason :	There is nothing particularly sensitive about the search path, but equally nobody else need know about it. The owner must be able to set it to suit requirements.	The file access software must be able to read the search path to find out where to look for files; there is no reason for the system to change the search path.	*There is no **need** for any access; in a secure system, none should be granted.*
Accounting information	Access type :	read	read, write	read, write
	Reason :	The owner can reasonably expect to find out the state of the account balance, but – for obvious reasons – should not be able to change the balance.	As resources are used, the accounting software must adjust the balance accordingly.	The accounts administrator must be able to read the balance to issue accounts, and to change the balance to reflect new allocations of funds.

POSSIBLE DEFENSIVE MEASURES.

How is the desired safety to be implemented ? We can sum up our safety requirements by constructing a large matrix in which subjects are tabulated against objects. This is called the *access matrix*. In the cells of the access matrix we record the sort of access which the subject may have to the object. The table above is an example of a simple access matrix, though access restrictions need not be limited to the simple examples shown : for example, it might be desirable to restrict some subjects' access to certain object to working hours only, or an object may only be used by a single subject at any instant. Clearly, a complete access matrix can be a very complex object, but that doesn't matter very much because there is rarely any reason to construct the matrix as an identifiable object. It remains a valuable abstraction for thinking about protection and security systems, but practical techniques rarely require access to more than one or two rows or columns of the full matrix for any single operation, and the required data can usually be kept quite compactly.

How can we build safety facilities into a system ? Unless we are going to introduce some completely new entity, the safety mechanism must be associated with the subject, object, or machinery – or, in principle, a combination of two or three, but in practice that doesn't seem to happen.

Methods have been developed based on all three participants. In subject-based methods, some characteristic associated with the subject must be identified before access to the object or the operation is granted; in object-based methods, the object is given a criterion against which it can judge requests for access; while, in machine-based methods, some properties of subject and object are compared to determine whether access will be permitted. In each case, methods of different degrees of sophistication are used – simple methods, useful for protection but not necessarily safe against deliberate attack, and more complex, and safer, techniques. This table summarises the methods in common use :

Level	Focus		
	Subject	Object	Machine
Simple (protection)	Passwords	Protection codes	Supervisor mode
Complex (security)	Capabilities	Access control lists	Protection rings

Cryptographic methods don't fit into this neat classification too well, but only because they don't address quite the same problem. Rather than controlling the access of subjects to objects, cryptography is focused on the *usefulness* of the object to the subject – if you can't understand my file, it doesn't much matter whether or not you can get it. Cryptographic methods can be used in the obvious way to encode files so that they can't easily be read by people who don't have the knowledge needed to decode them, and they can also be used as means of implementing some of the access control methods mentioned above. They are particularly useful in distributed systems, where it isn't possible to rely on local security imposed by hardware or operating system.

COMPARE :

Silberschatz and Galvin[INT4], Part 4. (Notice that their definitions of protection and security are similar to, but not quite the same as, ours.)

QUESTIONS.

Would there be any advantage in recovering a file system from incremental backup tapes by reloading the backup tapes in reverse order ? Give an algorithm. Would it be possible to stop before getting back to the original complete backup ?
