Conversations over the stable door
Anthony Furnivall
Buffalo News Inc,
A Division of Berkshire Hathaway
Buffalo NY 14240

This paper is not meant to be an exhaustive discussion of the security implications of the HP3000. Even if I were qualified to take on such a subject, the time needed to do it justice would far exceed the time available. Instead, what I will attempt to do is to present my own thoughts on the subject, and then perhaps we can explore together the implications of some of these thoughts.

There are two sides to the security coin, just as there are two sides to most important issues. On the one hand we have the needs of the user to be protected from long, involved sequences of commands in order to complete a task; to be protected, in effect, from the possibility of an error, a potential security violation. On the other hand, we have the need of an auditor to protect the data from the same sort of errors, or from the possibility of intentional violations. These needs are largely incompatible, and it falls to the system manager to attempt to keep everyone happy.

We have a classic good guy/bad guy conflict. We can find a similar conflict anywhere we look - the right to privacy and the public's need-to-know, or the rights of an employee and the needs of an employer. There are other aspects of this conflict that readily come to mind, but let us first examine the basic perspectives of the computer user, and the most typical conflicting perspective, that of the DP auditor. (Fig. 1) The user has explicit needs NOT to be bothered with extensive verification procedures, and NOT to be forever passing security checks. However, the user also has definite needs to be asked to verify intentions at critical points, to be saved from the possible consequences of his or her actions. This model of the user as "good guy" presupposes basically good intentions, a small amount of curiosity, and a fairly good grasp of what is possible, but an overwhelming need to get the job done as fast and as soon as possible.

Compare this with the need of the DP auditor, who requires the system to be able to go back and recover every key stroke of every session, to be able to identify who did what and with what and to what, and to be able to report the slightest apparent

malevolent intent as soon as it becomes apparent. This need is in self-evident conflict with the need of the user, and would indeed represent an intolerable burden if it were to be imposed only by manual means.

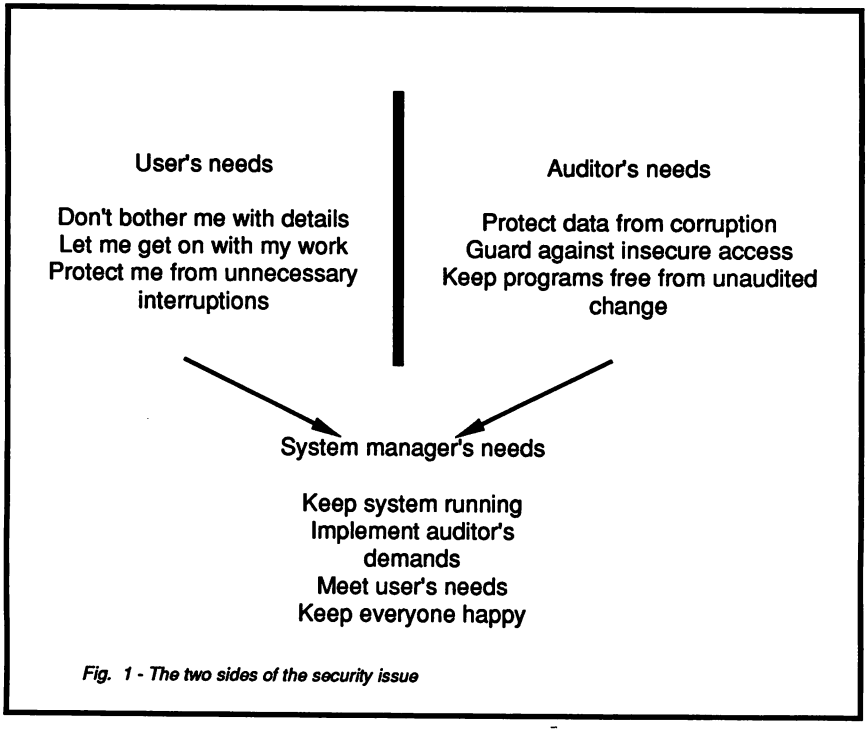


Fig. 1 - The two sides of the security issue

By contrast, the system manager or program designer has already got enough on his or her plate before worrying about the apparently unreasonable and conflicting demands of the 'User' and 'Auditor' roles. It is only too easy to set up the basic minimum of security and let things go at that. This has the added benefit of leaving time to track down disappearing disc space, unexplained program aborts, degraded system performance and other non-trivial problems.

Even though we have identified three basic role-models for those who worry about computer security, they are as I have said, reducible to two. The conflicting demands must be mediated by whatever means we choose to use when automating the security of our systems. In order to begin to do this we need to be aware of the 3 essential components of a security violation. These three components are Access, Awareness and Action. It is in the way that these three components combine that we have the different needs of a user and an auditor. Let us examine, for a moment, the validity of this proposition.

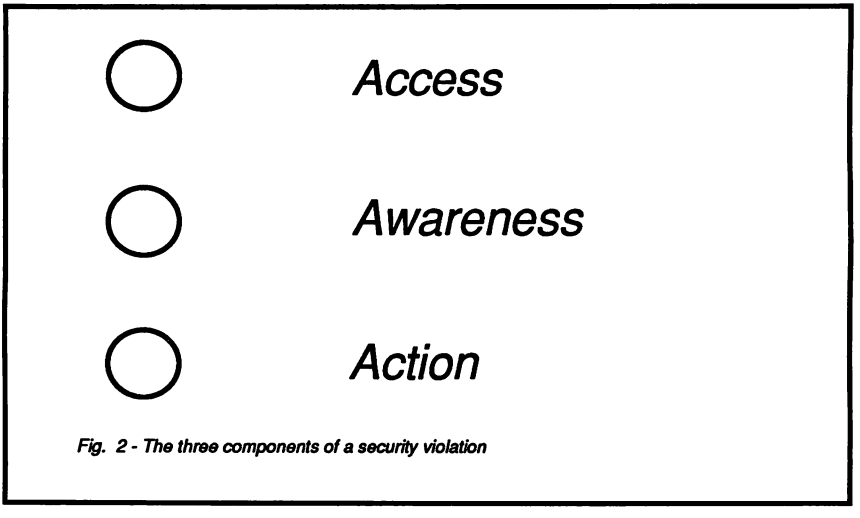


Fig. 2 - The three components of a security violation

In any computer system, there is a set of functions that are available. Each one of these functions has been designed for a specific purpose, and has been included for a good reason. Examples of such functions may range from reducing the balance of a loan by the amount of a payment, all the way down to providing a trace of the result of an IMAGE procedure call. The point is that every function has a purpose, an intended audience of users, and a set of precedents that determine the appropriateness of its use.

Now let us see how this set of functions fits in to our model of Access, Awareness and Action. To begin with, since the functions have been implemented by the designer, they must be accessible to at least some users. As an example of this, consider a standard that we have in place at the Buffalo News. (Fig. 3)

```
RUN CP525.PROG.CIRC;LIB=G
CP525     - (A.01.013) Carrier DM & Intro labels
CP525     - Tue, Apr 26, 1988,  4:55 AM


RUN CP525.PROG.CIRC;LIB=G;PARM=64
CP525     - (A.01.013) Carrier DM & Intro labels
CP525     - Tue, Apr 26, 1988,  4:55 AM
CP525     - CP525.PROG.CIRC          PIN 155 Parm=%000100
CP525     - TONY,MGR.CIRC,PRUN               Mode=%000007
CP525     - Flags: Test=N  Trace=Y  S220     Ldev 22
CP525     - Trace-file=CP525TR
```

Fig. 3 - Activating a feature of which the user may be aware

For programs that we design and code ourselves, running the program with ;PARM=64 will generally produce a trace of the program's action. This allows us to repeat a run which produced an error, and get a detailed internal listing of what went on. It is a useful diagnostic function which is intended for use by systems analysts in the event of problems. It is NOT intended for normal use because of the possible performance impact, and the possibility that a very large report would be generated, thus wasting both CPU cycles and paper - both of which are scarce commodities. ACCESS to this function is granted to all users who have access to MPE, and some of them are AWARE of this functionality. However, unless they choose to RUN the program in this way, they will not cause a violation of the intent of the function.

In much the same way, let us consider for a moment the payroll department of a company. Here is a group of people who are provided explicit ACCESS to some of the most sensitive information in the company, and who are very AWARE of the sensitivity of the

data. Again, unless we have the explicit ACTION of passing this information along to unauthorized persons, we do not have a security violation.

Here is where the different needs of the user and the auditor come into sharp distinction.(Fig. 4) The user expects warning and notification ONLY at the point where a security violation is about to happen; the auditor wants notification at the earliest possible moment.

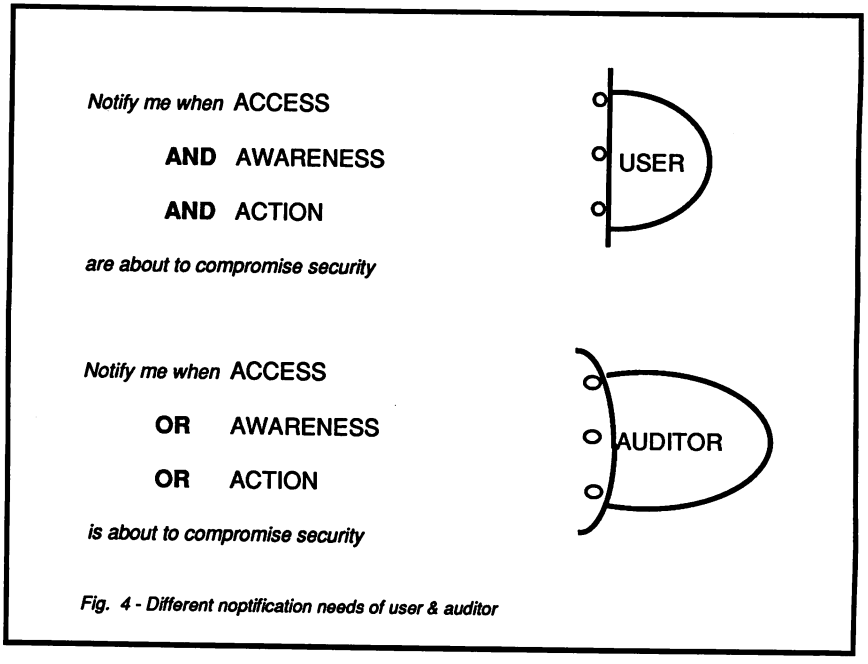


Fig. 4 - Different noptification needs of user & auditor

For the user, all of the components of the access/awareness/action model have to be active in a 'suspicious' context. For the auditor, the moment any 'suspicious' act occurs notification is necessary. Along these lines, we can see that any of the three elements of the model can be the source of a breach of security, and furthermore, that the determination of a violation is

not easily made. For this reason, most computerized data security mechanisms concentrate on the ACCESS phase of the model, because this the easiest one to secure.

The last model that we need to consider in a preliminary examination of the issues is the model of the components of a secure system. We need to consider this model in the light of the others - that is to say, in the light of the different needs of a user and the system, and bearing in mind the three areas that need to be monitored.

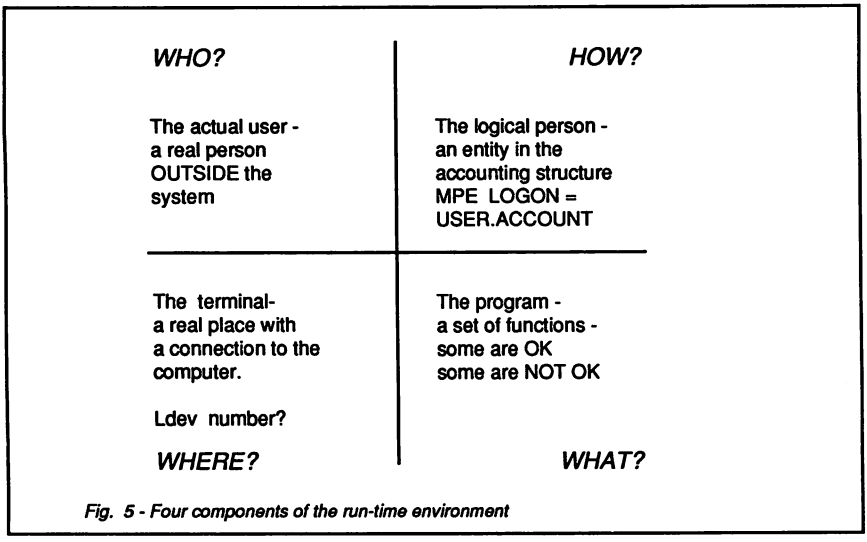


Fig. 5 - Four components of the run-time environment

This model of computer security focuses on the four basic questions that reporters are trained to ask every time they begin to write a story -the Who, What, Where, How questions. As we shall see, in terms of computer security these same questions apply. Let us consider these four components one by one, and see how they contribute to a fuller understanding of our other models. I propose to deal with them in the following order: Who, How, Where, What.

The question of WHO is doing something on a computer system seems, at first sight, to be almost trivial. (Fig. 6) After all, we have to identify ourselves at sign-on time, and this determines who we are. Or does it? Using MPE as a model, we can see that USERs are related in a sub-ordinate way to ACCOUNTS, and thus are explicitly differentiated between accounts. MPE enforces this separation almost without exception (the :ALLOW command does in fact allow an orthogonal view of the USER/ACCOUNT relationship). Our need is for a relationship which is super-ordinate to the accounting structure; an ability to model a pervading user who can (with any luck) access ANY account. The accounting structure, then, does not offer any support for our concept of the pervading user. MPE does provide such support, however,in the optional session-name, or job-name component of the sign-on sequence. Since this is not related in any way to the accounting structure of the system, we are free to

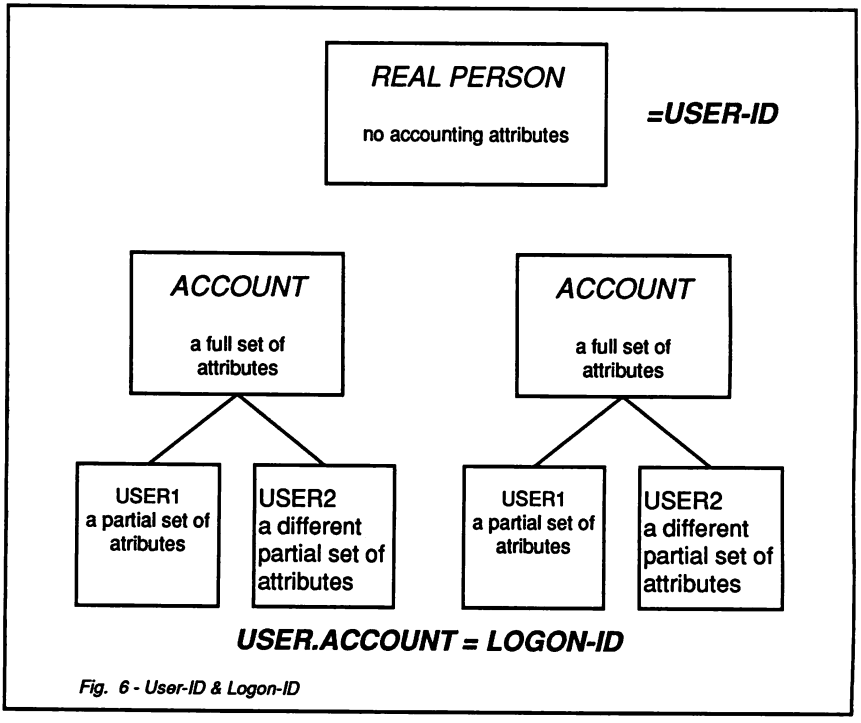


Fig. 6 - User-ID & Logon-ID

use it as a model of the real person who is using the system. This model I propose to call the USER-ID, to enable it to be easily identified later on.

The next question we need to ask is HOW is the user represented inside the computer system. This is where the computer's accounting structure becomes of primary importance. From the MPE perspective, when we log on to the 3000, we specify a combination of account and user which together constitute a LOGON-ID. This logon-id has certain attributes associated with it, and these attributes are used extensively by MPE to determine the permissibility of almost everything that the user wishes to do. Some linking of the logon-id with terminal attributes is done at logon time, but this is rather limited in nature. It is important to note that MPE allows a logon-id to have a subset of the attributes available to the account. For example if an account is granted SM capability, there is no need for all the users to have this capability. Depending on which LOGON-USER is in effect, the capabilities of the account will be restricted to the set that has been granted to the LOGON-USER.

Our examination of the four part model of computer security needs also to consider the WHERE question. What I mean by this is, "Where is the actual user presently located, and is it appropriate to allow this request to continue?" It is easy to imagine instances when this question needs to be asked - if the user is seated in the front lobby of the building, it might perhaps NOT be a good idea to allow the salary of the president to be displayed. Similarly, if a given function is requested by a terminal located in a user department that is more appropriately performed by the system manager's terminal, this might also be grounds for the system to consider the request a violation. Note that, in this regard the terminal has some of the attributes of both a user and a resource.

Unfortunately, the question of identifying the location of the user is a vexing one for two situations, and in these situations MPE is not a very helpful partner when it comes to tracking down potential security violations. (Fig. 7) These two areas are batch jobs and data-communications devices. For a batch job it would be desirable to know the source of the batch job - that is to say, who (in the model's sense of the word) streamed the job, and what file was used (including $STDIN). This monitoring would need to be

extended down through however many levels of jobs are initiated by the original stream command. This is provided at the present time by various stream management packages, but, as far as I am aware, the data is NOT available programmatically to processes running under those jobs.

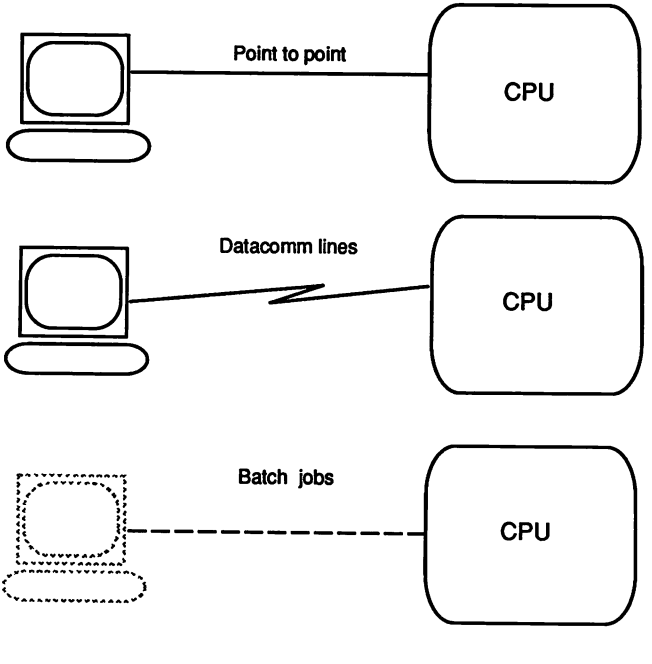


Fig. 7 - Terminal location, and the question of 'Where?'

A similar situation exists for those logical devices on an HP3000 machine which are connected via modems, data switches and other similar boxes where the end to end connection between the port of the computer, and the terminal is not fixed and unchanging. Such a situation obviously covers a large number - possibly even a majority - of cases. In these situations it is also impossible for a program to

determine the true location of the user, and thus to include this in its own decision making. NS/3000, the software component of HP's local area networking product does in fact address this problem, and it is possible to work backwards through an NS link and discover where the user is located, although this may be a rather torturous process.

Considering the WHAT question is considerably easier - computers use programs to change the data, and the WHAT of our model is unequivocally related to the program that our user is running. While the model is easy to define in this instance, it has some disturbing implications.

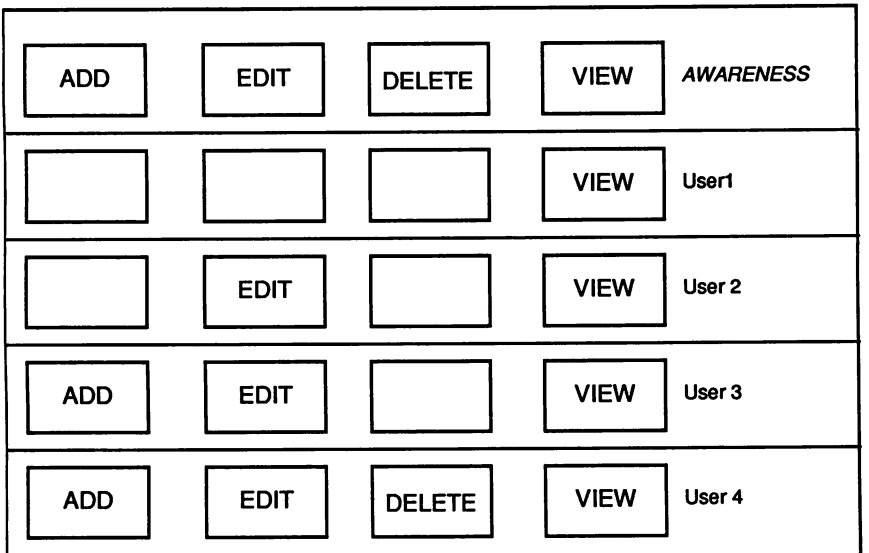


Fig. 8 - Programmatic functions

If our model is to be aware of and sensitive to the complete environment, that is to say the combination of USER-ID, LOGON-ID, TERMINAL-ID and PROGRAM-ID, this implies that programs may need to produce different results for different users! How many

programs do you know that are capable of dynamically adjusting themselves to a different user? IMAGE does in fact include this capability, but the programs that use IMAGE procedures need to accomodate the varying results of IMAGE calls. It seems likely that most programs do NOT provide a dynamic functionality based on the actual user-id, but rely instead on some other means of preventing abuse.

As an example of what I mean in this regard, I would like to present a brief model of a typical data-maintenance program. (Fig. 8)This program uses function keys f1 thru f4 to request Add, Edit, Delete and View functions. From the point of view of consistency, all data-maintenance in a system will use this model. It seems reasonable then, for a program to disable these functions when they are not appropriate for use by a given user-id. The user may well know that f1 means Add, but the program must not only disable the 'awareness' of the user (by hiding the function key label) but must respond in an appropriate manner to both the user and the system manager. The user needs either a simple message that the function is unavailable, or preferably, a null response from the program. The system manager, or the auditor may need to know that an attempt was made to execute a function that would have resulted in a security violation.

```
FILE SENSITIV;DEV=PRIVATELP

FILE SENSITIV;DEV=PUBLICLP

FILE SENSITIV;DEV=DISC;SAVE

FILE SENSITIV;DEV=MODEM

FILE SENSITIV;DEV=IBMPC
```

Fig. 9 - Run time detection of report destination

Another example of the need for a program to respond to its run-time environment is found in the ability of a file equation to redirect the destination of a report. (Fig. 9) What may be intended to go to a private line-printer may end up being directed to a public line-printer, a modem or a PC! This is something that can only be detected at run-time, but at least it can bedetected.

MPE then, provides a reasonably good matching of the four phase model of computer security, and provided we can access the attributes that are provided, and define relationships between them, we ought to be able to make a good decision about the permissibility of any request by a user. One of the important benefits of the four component model is that once access to the computer system has been established, three of those components do not change. (Fig. 10) (For now I will exclude the problem of an unattended terminal).

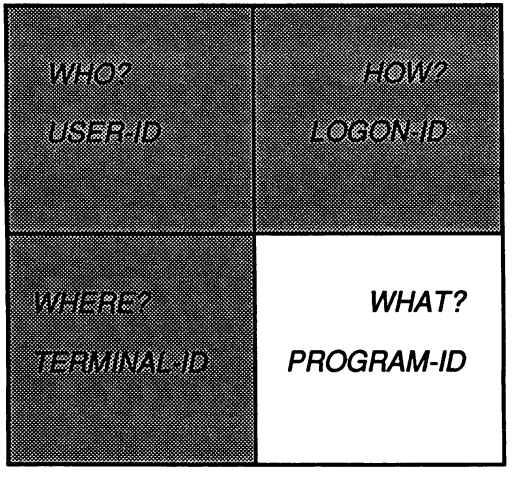


Fig. 10 - After logon, most of the components remain the same

This makes the task of security checking easier - since part of the checking is handled once at logon time, and can then be implied after that. We must remember though, that we have explicitly included an entity outside the knowledge of MPE, and the directory structures. For this reason, we are faced with the fact that the only way we can determine the legitimacy of access to the system is to allow access, and then deny it if necessary. Further, to provide true security, we need to find some means of determining whether any potential violation of access or awareness is contained in any given request for an action to be performed.

It is as we consider this need that the importance of all three models becomes apparent: The user has different needs from the Auditor (or the auditor's representative - the system manager); we need to monitor access, awareness and actions, and we need also to keep in mind the binding represented by the user-id, the logon-id, the terminal-id and the program-id. What would be reasonable would be to consider various strategies for keeping track of all these entities, and possibilities.

If we are to do anything to meet the needs of the system manager, it is obvious that we must monitor every action. Equally obviously if we are to meet the needs of the user, we must not make such monitoring an obstacle to getting useful and productive work accomplished. Consider the common practice of providing employees with an identification badge. The purpose for this is to allow some mechanism (generally a security guard, or a card-key door) to identify employees, and to grant them access. Typically, such programs start out with a lot of attention, and the security guard is very active checking every employee for their badge. This is generally viewed as a hassle and as an obstacle to productivity! In addition, after a while it is possible to notice that the guards adopt a different strategy towards checking employees - they get to know who is who, and the frequency of checks becomes less. What we need to find is a way for the intrusion to be minimized, but for the vigilance to be maintained.

MPE provides a very good example of incorporating security consciousness into its activities. In addition, it does a fair job of meeting the conflicting needs of the user and the auditor. I shall

show how this occurs, and then go on to discuss the strategy used to grant approval.

```
USER'S TERMINAL

ENTER LOGON PASSWORD (MGR):

ENTER LOGON PASSWORD (MGR):

ENTER LOGON PASSWORD (MGR):

INCORRECT PASSWORD (CIERR 1441)


SYSTEM CONSOLE

13:12/#S545/229/INVALID PASS FOR "TONY,MGR.CIRC,TRUN" ON
LDEV "65"

13:12/#S545/229/INVALID PASS FOR "TONY,MGR.CIRC,TRUN" ON
LDEV "65"

13:12/#S545/229/INVALID PASS FOR "TONY,MGR.CIRC,TRUN" ON
LDEV "65"
```

Fig. 11 - MPE accomodates both reporting needs

When a user fails to provide the correct password at logon time, MPE notifies the console immediately, reporting the complete logon string, and the logical device. (Fig. 11) The user receives another prompt, with no indication that the first attempt was inaccurate. Only after 3 failed attempts does MPE indicate that the user has supplied the wrong value. Notice how our two conflicting needs have been met. The system has supplied notice to the system manager at the earliest possible moment, and the user is not obstructed until after three attempts. An area where MPE does NOT do as good a job of taking care of the system manager's needs is in the area of reporting file security violations. These are reported back to the user, and can be handled as appropriate or necessary. However MPE provides no notice to the console of such a security

violation. New versions of MPE will permit the logging of file opens, and this will help in some way to rectify the situation.

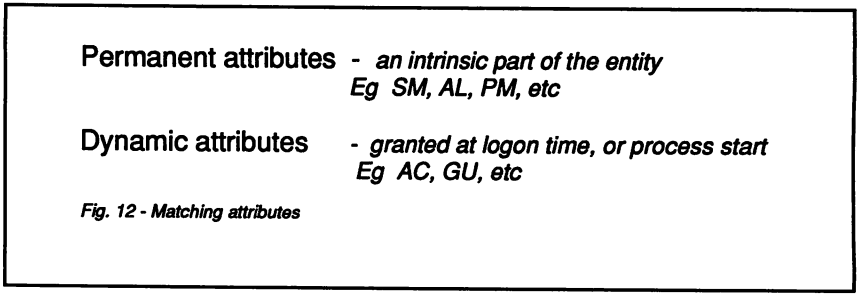


Fig. 12 - Matching attributes

We have seen that MPE is evidently continually checking security parameters as it does its work. Two distinct strategies can be defined in the approach to security. The first is the strategy of matching attributes. (Fig. 12) These attributes are of two varieties - permanently assigned, and dynamically assigned. Consider the attribute AL, for account librarian. This attribute is available to the logon-id (since it is available at both the account and user level), and is always present whenever someone is logged on in this way. This attribute is used to determine the legitimacy of a file access.Or take the attribute OP. This permanent attribute is used to determine the legitimacy of (among other things) a STORE command.

The second sort of attribute is the attribute which exists ONLY for the duration of the current session or job. Such attributes include the Group User (GU) attribute, and the Account User (AC) attribute. These attributes are also useful in evaluating security, especially since they are a truer reflection of the present environment.

Another aspect of attribute matching is that attributes may be either required or permitted. (Fig. 13a) A required attribute is an attribute that must be present in order for a request to succeed. MPE uses its attributes in this way. For example, if I want to issue a NEWACCT command, I must have the SM attribute active in my logon-id. Similarly, if I wish to use data communications devices, I must have the CS attribute active. It is equally possible to disallow

attributes. While MPE does not define any such attributes, they are easy to think of. Multiple logons would be a good candidate. (Fig. 13b).

Required attributes - *attribute must be present to gain access*
Permitted attributes - *attribute must be permitted if access is access is sought with the attribute active*

Fig, 13a

Multiple logons as Required attribute	*I must have this attribute granted to me in order to gain access*
Multiple logons as Permitted attribute	*If I am logged on more than once (i.e. the attribute is active), then the attribute must be permitted in order for me to gain access*
Net effect	*I may only gain access if:* *a) I MAY log on more than once and* *b) This is my current FIRST logon*

fig 13b

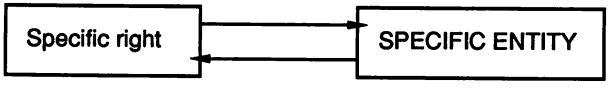
Fig. 13 - Various aspects of attributes

In this example, access to a logon-id would be denied if the access would result in multiple logons for the user-id. Note that an attribute such as this can be used in both ways. For example, it may make sense to limit access to a logon-id to those people who

have the required attribute of multiple logons. It is even possible to model some pretty exotic situations using something as simple as required and permitted attributes.

Consider for example the possibility that access to a logon-id is available only to those who have the required attribute of multiple logons (i.e. they CAN log on more than once), but who are NOT currently logged on more than once (the same attribute is not a permitted attribute). We have effectively said that access to this logon-id is limited to the FIRST logon by a given user-id. Attribute matching in this way can provide a very strong degree of protection in ways that would be otherwise difficult to describe. Attribute matching, according to changeable specifications, is one of the two strategies that MPE seems to employ to manage the security function.

The second one is the concept of assigned rights. This rather awkward term is what I use to describe the ability of a file's creator to change the name. In effect, the right to do this is assigned BY the file TO the logon-id. This strategy is a very useful one, because it provides a 'hot-line' to a particular functionality.



Rename a file	CREATOR OF FILE
Use system console	LOGON AS OPERATOR.SYS

May be used to extend or limit access

Fig. 14 - Assigned rights, another strategy

Assigning a right is dependent on being able to define two things - the right that is being assigned, and the entity to which it is being assigned. (Fig. 14) It is easy to consider, for example, the possibility of extending MPE security to assign the right to use the system console to a particular logon-id (OPERATOR.SYS) for example. By assigning a right in this way we create a fixed set of rights that can be scanned for validity. If a right does not exist in the set then any request to exercise that right will fail. Another example of an assigned right is the way in which programs such as DBUTIL will operate only on databases in your logon group and account. Here what has happened is that the right to reference files in other groups has NOT been assigned. It is therefore missing from the set of assigned rights and, as a result, the request will fail.

It is interesting to speculate on ways in which MPE could be enhanced by using this concept. Perhaps we could assign a terminal to a specific logon-id. In this way we could guarantee that only PERATOR.SYS could log on at the system console, or perhaps we could limit use of the dial-in port to MGR.TELESUP (or possibly NOT MGR.TELESUP!). In addition to allowing specific rights, the concept can also be used to limit functionality. In the example we used above, it might be that the only device that OPERATOR.SYS can use is the console.

A third strategy that is not used for security checking in MPE (so far as I know), is the concept of security level. This is the same concept as priority level, except that what is being ordered is not the sequence of events, but their permissability. Let us see how such a concept works. (Fig. 15) We assign an arbitrary level of security to a logon-id, eg 4. Only users with a security level of 4 or above can access this logon-id. Or again, we assign a particular function within a program a security level of 3. When the program is run by a user with a security level of 2 or less, the function is disabled, and when the program is run by a user with a security level of 3 or more, the function is re-enabled. Several benefits accrue from this strategy. To begin with, it is a rule rather than a reality. We can compare the two security levels involved and gather our result based on a rule, rather than on the result of finding this specific instance specified. Compare this with the concept of assigned rights, where every possible instance must be accomodated. Using security levels, new entities can be added to the environment without modifying the set

of assigned rights. Furthermore, if we want to restrict access to a logon-id, we merely raise its security level.

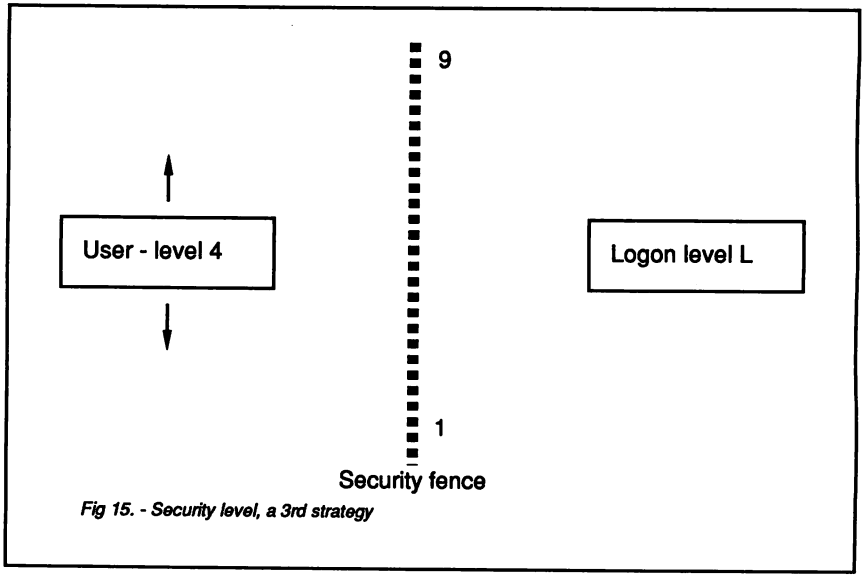


Fig 15. - Security level, a 3rd strategy

To summarize, then, we have three possible strategies for determining the approval of a request. They are: the strategy of matching attributes, either static, permanent attributes or dynamic attributes; the strategy of assigned rights and the strategy of security level. By combining these in various ways it is possible to achieve a very flexible degree of control over security requests. Even so, we are only handling explicit requests. Another large problem for the security function is the problem of nothing happening.

When a terminal is inactive for any substantial period of time, the binding of the user-id, the logon-id, the terminal-id and possibly the program-id becomes weaker and more tenuous. (Fig. 16)`There exists some threshold of inactivity beyond which we can not assert that the binding is valid. In some instances this may not

be a problem, but in others it will be.

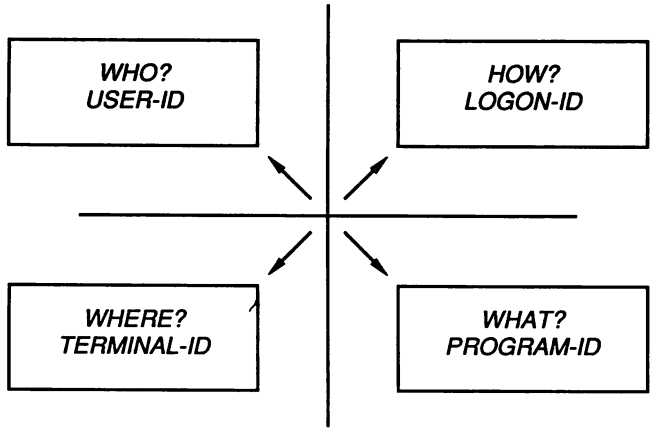


Fig. 16 - The problem of the inactive terminal

What we need to be able to do is to define the maximum period of time that any one of the four components can be inactive; and then, to define the action to be taken once this threshold is reached. If the period of time is of no consequence, then a value of 0 could easily indicate this. Possible actions to be taken would include re-verification of the user-id, or the terminal-id, forced termination of the program or forcing an end to the session. What is important to realise, I think, is that each of the components needs to be defined in this way. Only thus can we preserve the integrity of all four components.

The last area that I wish to address in this paper is the problem of data integrity. This subject warrants an entire paper on its own, but I wish to consider it from the point of view of ensuring that the program which is requesting permission to access and manipulate data is in fact the program that it purports to be. The idea of asking a person to prove who they are by supplying the

answer to a question (such as a request for a passsword, or a personal knowledge question) is not new. However, it is not possible to ask for an animate response from an inanimate program.

It is easy to ask a system manager to ensure that only the right version of a program is allowed into production, but not so easy to make it work. We need to define a sufficient set of data to make it impossible (or at least sufficiently unlikely to be almost impossible) for a program to be a wolf in sheep's clothing. What constitutes such a set of data?

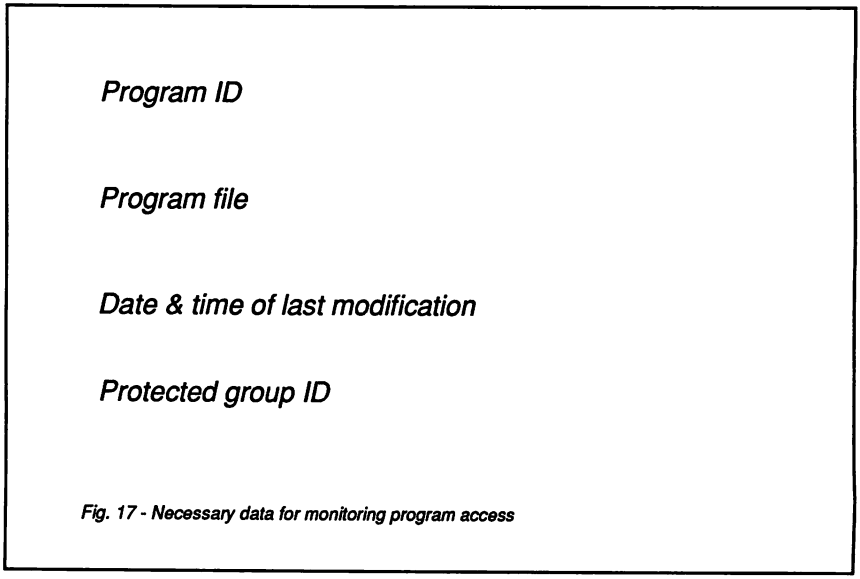


Fig. 17 - Necessary data for monitoring program access

The first element we need is obviously the name of the program. (Fig. 17) This is not the same as the name of the program file, but the name by which the program is known to the programmers, designers and other people. In short we need the PROGRAM-ID. In addition, we need the name of the program file. This is a piece of data which can easily be extracted at run time. We also need to know the date and time of the last change to the

program-file. Armed with this data, we can compare it with known values, and determine if the program that is requesting permission to proceed is in fact the program that is allowed to proceed. In order to allow testing, the checking is only done when the logon-id indicates that a specific protected group is being referenced, either directly, or by file equations.

The other aspect of a program which bears scrutiny is the frequency with which it is run. If a program is run more frequently than expected, it may represent a possible security risk. At the very least it is worth bringing to the attention of the system manager. What we need to specify in this instance is a range of time that we can reasonably expect between activations of the program. Naturally, for programs where this is of no concern, nothing needs to be done.

At the start of this paper I said that I would follow the implications of my own thoughts on some security related issues. Where I think we have arrived is the description of various strategies for implementing the security function in the programs that run in your own shops. Some of these may be programs that are written in your own shops, and over these you can exercise as much control as you wish. Yet other programs may be aquired from 3rd party software vendors. Over these you can obviously exercise less control.

There has been much discussion recently about the importance of security, and the importance of being concerned about it. The place for such concerns is not only in the operating system software, but also in every program that runs on a system. Ultimately, the value of security consciousness is defined by the possibility of loss and the cost of loss. This is an equation which must be solved anew for every site. I hope that the ideas presented here will stimulate your own thinking, and lead you into defining your own solution to that equation.