# BURN BEFORE READING - HP3000 SECURITY AND YOU.

by Eugene Volokh,
VESOFT, INC.

## DISCLAIMER

One of the most important things you can do for your security system is to plug holes that may exist in it. To help you do this, this paper shows some ways in which most inadequately secured systems can be penetrated. Although this information can be used by would-be security violators to break into a poorly secured system, it is in my opinion more important that it can be used by you to protect yourself against these violators.

ATTENTION WOULD-BE THIEVES! DO NOT READ ANY FURTHER!

## INTRODUCTION

Life's not fair.

Just when you think you've got it made, just when you (or your company) have found the pot of gold at the end of the rainbow, you find it out.

**Someone else wants the same pot.**

To prevent your property from becoming theirs, you set up a series of obstacles between the would-be thief and your property. It is these obstacles that comprise your security system, and it is the quality of these obstacles that determines whether or not your property (in our case, your computer data) is secure.

These obstacles come in two flavors:

* Obstacles to unauthorized retrieval of data. Data is often valuable in and of itself, whether it is salary information you want to keep secret from your employees, financial information you want to keep secret from your competition, or military information you want to keep secret from THEM.

* Obstacles to unauthorized modification of data. Data does not exist for its own sake; real-life decisions are made based on that data, and unauthorized modification of data can affect those decisions in an undesirable way.

This paper will try to give you some useful tips on making your valuable data more secure.

## THE ROAD TO YOUR DATA

Consider Joe Q. Sinister, who has his sights set on your payroll database. There is a fixed road that he must travel to reach the data stored in it and change it; knowing this road will help us erect the proper roadblocks.

His first step must be to log on to the computer; if we can frustrate his attempts to do that, our data is secure. The techniques used to prevent unauthorized users from logging on to the computer are called LOGON SECURITY.

## LOGON SECURITY

Logon security is probably the most important component in your security fence. This is because many of the further security devices (e.g. file security) use information that is established at logon time, such as user id and account name. Thus, we must not only forbid unauthorized users from logging on, but must also ensure that

even an authorized user can only log on to his user id.

So, logon security essentially involves ensuring that the person logging on is authorized to use the user id he is logging on to. How is this to be done?

The optimal approach, of course, would be to somehow identify who the person is (fingerprints? retina scan?) and check to see if he is on the authorization list for the particular user id. Unfortunately, these approaches are not within the means of most HP 3000 users. However, another good method is.

A person can be identified by what he knows almost as well as by what he looks like. For instance, a user id may be assigned a password, and only the people authorized to use that user id may be told that password. Then (assuming no one else somehow learns the password), if a person knows the password, it follows that he is authorized. Alternatively, if one and only one user is allowed to use a particular user id, he may be asked to enter some personal information (mother's maiden name?) when he is initially added to the system, and then be asked that question (or one of a number of such personal questions) every time he logs on. This general method of determining a user's authorizations by what he knows we will call "knowledge security".

Unfortunately, the knowledge security approach, although one of the best available has one major flaw -- unlike fingerprints, information is easily transferred, be it revealed voluntarily or involuntarily; thus, someone who is not authorized to use a particular user id may nonetheless find out the user's password. You may say: "Well, we change the passwords every month, so that's not a problem". The very fact that you have to change the passwords every month means that they tend to get out through the grapevine! A good security system does not need to be redone every month, especially since that would mean that at least towards the end of the month, the system is already rather shaky and subject to penetration.

Ironically, the biggest culprit in this respect is the user himself. Users have been often known to write down passwords and post them in prominent places so they will not forget them; reveal passwords to people who really shouldn't know them; and, in general, wreak havoc on your logon security system. Some ways have been designed to cope with this, such as the personal profile security system (asking questions such as "What's your mother's maiden name?", "Where did you go on your first date?", etc.) described above, whose main advantage is that users are less likely to reveal personal data than impersonal passwords; additionally, there can be more than one personal profile password -- all of them or a random one can be asked at logon time -- whereas there is only user password. However, the user is still the weakest link in the logon security system, and major steps should be taken to avoid voluntary password disclosure by the user. Thus, an important security rule arises:

* THE USER IS THE WEAKEST LINK IN THE LOGON SECURITY SYSTEM -- DISCOURAGE HIM FROM REVEALING PASSWORDS (by techniques such as personal profile security or even by reprimanding people who reveal passwords -- they seem innocent, but they can lose you millions).

Yet another way in which passwords are often revealed is by having job streams with embedded passwords. First of all, unless you take special precautions (such as altering the job streams so that Read access to them is disallowed, and only Execute -- enough for :STREAMing -- is permitted), anyone who can stream the job stream can also read it and thus see the passwords; in any case, any listing of the job stream (of which plenty are liable to be laying around the computer room) contains this password. More importantly, since changing a password means having to change every single job stream that contains it, these passwords are virtually guaranteed never to be changed. Fortunately, there is a simple way to resolve this problem: there are plenty of programs, contributed and vendor-supported, that take a job stream without embedded passwords, prompt for them, insert them into the job stream, and then stream it.

* PASSWORDS EMBEDDED IN JOB STREAMS ARE EASY TO SEE AND VIRTUALLY IMPOSSIBLE TO CHANGE -- AVOID THEM.

Another way of increasing logon security is by indirectly using another aspect of user identification -- identification by human beings. Actually, this could be the main part of your logon security system: any user who wishes to sign on must first get clearance from a security guard or console operator. Going quite this far is too expensive, but a little bit of this can be obtained for free.

If some 15-year-old high school student walks into your data entry area and starts using the computer, people are bound to notice. It is fear of being identified as a security violator by other human beings that makes most violation attempts come across phone lines, usually at night or on weekends. Thus, another useful security feature is to be able to restrict access by access location (i.e. terminal) and access time. The very fact that someone is trying to run payroll across a phone line at 11 PM on a Saturday is an indication of unauthorized access. Thus, it is worthwile to implement some form of security that prohibits access to certain user id's and accounts at certain times of day, days of week, and/or from certain terminals. Alternatively, you might want to force people to answer an additional password at certain times, or especially when signing on from certain terminals.

This may seem like a poor approach indeed -- after all, if the thief hits the time of day, day of week, or terminal prohibition/password, this means that he has successfully penetrated the rest of your security system, which will never happen -- right? In reality, this is a very potent way of frustrating would-be security violators, especially if the attempted violators are promptly investigated. Thus, another maxim appears:

* SOME FORMS OF ACCESS ARE INHERENTLY SUSPECT (AND THUS REQUIRE EXTRA PASSWORDS) OR ARE INHERENTLY SECURITY VIOLATIONS. THUS, ACCESS TO CERTAIN USER ID'S AT CERTAIN TIMES OF DAY, ON CERTAIN DAYS OF WEEK, AND/OR FROM CERTAIN TERMINALS (SUCH AS DIAL-IN OR DS LINES) SHOULD BE SPECIALLY RESTRICTED.

## ASIDE -- ATTEMPTED VIOLATION REPORTING

Before we go any further with our discussion of various security devices, it is worthwhile to pay particularly close attention to something which should be present in all security devices -- violation reporting.

No security system can cover you 100% -- given enough time, a determined (or even relatively casual) thief can penetrate even the best system. Fortunately, before this one successful penetration, chances are that the thief will make many unsuccessful attempts; if you pay attention to these unsuccessful attempts, you can catch the thief (or at least improve the security system by, say, temporarily shutting down dial-in lines) before he gets in.

This may seem obvious, but few shops really pay attention to unsuccessful penetration attempts -- when was the last time you looked at "INVALID PASSWORD" messages on the system console or in the log files? In reality, every incorrect password entry is indication of a possible attempted security violation, even more so if there are several such errors in a row.

HP doesn't help any either -- the INVALID PASSWORD messages look just like any other console message (no enhancements of any kind); the only place where invalid password entries are logged are in the system log files together with the rest of the console log messages. It would have been far more desirable if the message were logged to a separate log file, and maybe even reported to the line printer or some special device. Additionaly, it might be wise for a terminal on which an invalid password entry occurs to be shut down for some period of time so that it would take more time for a would-be thief to try more passwords.

But, even with the existing HP system, an alert console operator can nip many a potential security violation in the bud by catching the INVALID PASSWORD messages that can be a sign of an attempted violation. In fact, there is a way to highlight some messages so they will be more easily visible. Since most MPE messages are stored in the system file called CATALOG.PUB.SYS, you can do the following:

1. Sign on as MANAGER.SYS.

2. In EDITOR (or TDP), /TEXT CATALOG.PUB.SYS

3. Modify the first line in the file that starts with "65 " and the first line that starts with "68 " to contain an escape sequence such as "escape&dB" (inverse video) right after the blank after the message number and to contain a "escape&d@" (turn off enhancement) at the end of the message. Alternatively, if you have a 263x with expanded character set, insert an "escape&k1S" (enter expanded set) right after the blank after the message number and a "escape&k0S" (exit expanded set) at the end of the message. Similar escape sequences may be put in if you have some other kind of terminal or a voice output device.

4. /KEEP the file as INPUT.

5. :RUN MAKECAT.PUB.SYS,BUILD

6. Presto! Your "INVALID PASSWORD" and "MISSING PASSWORD" messages are now much easier to read.

Thus:

* MANY SECURITY VIOLATIONS CAN BE AVERTED BY MONITORING THE WARNINGS OF UNSUCCESSFUL VIOLATION ATTEMPTS THAT OFTEN PRECEDE A SUCCESSFUL ATTEMPT. IF POSSIBLE, CHANGE THE USUAL MPE CONSOLE MESSAGES SO THEY WILL BE MORE VISIBLE.

## LOGOFF SECURITY

Another threat to your system security is, unfortunately, a rather common one. If someone signs on to a terminal and then walks away (for a lunch break, say), a would-be thief can access your computer without even having to log on -- just walk up to the terminal and use it.

You may think this to be a relatively rare occurence, but consider: do your people always sign off when they go to lunch? Haven't there been times when they forgot to sign off even when they leave for the day? Leaving a terminal signed on is a very common mistake, and one that can greatly jeopardize the security of your system.

How can you solve this problem? Well, for one, you can tell your people -- whenever they leave the terminal, they should sign off. Alternatively, if you find that people often leave the terminal when it's in some particular state (say, the main menu of your accounts payable program), set a timeout just before issuing the terminal read (with the FCONTROL intrinsic, mode 4). That way, when the user does not respond for a certain amount of time, the read will abort, and your program will be able to terminate and maybe log off the user. An even better alternative is to use a contributed or vendor-supplied program that automatically aborts all terminals that have been inactive for more than a certain amount of time (such as Boeing's BOUNCER or VESOFT's LOGOFF).

Another, more dangerous, problem occurs when a dial-in user hangs up the phone instead of properly :BYEing off. Then, if the dial-in line is configured with subtype 0, the user will not be automatically :BYEd off, and the next person to call up the computer will be dropped into the still-logged in session. Thus, remember to configure all your dial-in lines with subtype 1 or tell your users in no uncertain terms that they MUST always :BYE off when using the dial-in line.

Thus,

* LEAVING A TERMINAL LOGGED ON AND UNATTENDED IS JUST AS MUCH A SECURITY VIOLATION AS REVEALING THE LOGON PASSWORD. USE SOME KIND OF TIMEOUT FACILITY TO ENSURE THAT TERMINALS DON'T REMAIN INACTIVE FOR LONG; SET UP ALL YOUR DIAL-IN TERMINALS WITH SUBTYPE 1.

## RESTRICTED VS. UNRESTRICTED USER INTERFACE

As was mentioned before, logon security is a very important component of your security system, but it is by no means the only one. Many security violations are committed by people who are allowed to sign on to the computer but manage to get at things that they are not permitted to access.

There are two major ways of forbidding authorized users from doing unauthorized things. One is by permitting them to do only certain specific things (the inclusive approach) and the other is by forbidding them from doing specific things (the exclusive approach). Each has its merits, its uses, and its security strategies.

## THE INCLUSIVE APPROACH

Briefly, the inclusive approach is usually implemented by having an OPTION LOGON, NOBREAK (the NOBREAK is important!) UDC that runs an application program and then, upon exit from the program, immediately BYEs. Thus, the user is only allowed to perform the function or functions of this one program (or, if the program so wishes, only a subset of these functions), and he is forbidden from doing anything else -- accessing files, running programs, or executing MPE commands.

This is, overall, a good approach. Its only real problem is that in some instances, it is too restrictive -- some users (especially programmers) need to have access to the

entire power of MPE. However, when the user does not need to access MPE, it is not only more secure but also more convenient for the user to be automatically dropped into his program when he signs on and be automatically signed off when he exits the program. However, certain technical issues must be kept in mind:

1. Don't forget to make the UDC OPTION LOGON, NOBREAK. If you omit the NOBREAK, the user can hit break, type :ABORT, and get into MPE.

2. A less-known fact is that it is usually essential that you add a CONTINUE line before running your program, thus making your UDC look something like

   LOGONUDC OPTION LOGON, NOBREAK CONTINUE RUN ACCPAY.PUB.AP BYE

   Why? Because otherwise, if the program aborts the entire UDC will be flushed and the BYE will never be encountered. Although it might seem quite improbable that your program will abort, the user can actually make most programs abort by typing a :EOD (or sometimes just a :) when prompted for input. This causes an end of file on $STDIN and makes many programs, including almost all BASIC, COBOL, FORTRAN, and PASCAL programs, abort.

Of course, this approach need not be restricted to running simple applications programs. One of the best uses of this approach is to run some program that displays a menu of allowed MPE commands or constructs and asks the user to choose one. Thus, if you want a user to access the A/P system, EDITOR, or the TELLOP command, you might write a program that displays these three options to the user, asks the user for one, and then executes it (via the COMMAND or CREATE intrinsic). Even better, get a general-purpose menu procesing program that permits you to easily set up various menus by just changing some data files. Thus,

* A USEFUL APPROACH TO SECURING YOUR SYSTEM IS TO SET UP A LOGON MENU WHICH ALLOWS THE USER TO CHOOSE ONE OF SEVERAL OPTIONS RATHER THAN TO LET THE USER ACCESS MPE AND ALL ITS POWER DIRECTLY.

## THE EXCLUSIVE APPROACH

Sometimes, programmers or other users that have to use a wide range of programs, files, and MPE commands must have access to MPE itself. This is a far less controlled environment than a program that is run at logon time, but can still be secured very well.

One approach to securing the system while still allowing people to access MPE is to disable certain MPE commands you find undesirable. For instance, say you do not want your people to :STREAM jobs. You could set up a system or account UDC

```
STREAM !FILENAME="$STDIN", !COLON="!"
OPTION LIST
COMMENT  YOU ARE NOT ALLOWED TO :STREAM FILES.
```

That way, whenever someone types a :STREAM command, he gets the UDC instead.

This approach, however, has a major flaw: Although the command interpreter gives precedence to UDCs over ordinary MPE commands (thus allowing you to block out :STREAM commands by setting up a STREAM UDC), the COMMAND intrinsic does not. Thus, if the user is allowed to access FCOPY, EDITOR, TDP, SPOOK, or even a user-written program that calls the COMMAND intrinsic, he will be able to bypass the UDC restriction. In other words, in the example above, all I need do to bypass the :STREAM command restriction is to run FCOPY, and type the :STREAM command from there!

The only exceptions to the above rule are the commands that can not be directly executed via the COMMAND intrinsic, such as :RUN, :PREP, the compiler commands, :SETCATALOG, and :SHOWCATALOG. But even these commands (all except :SETCATALOG and :SHOWCATALOG) are available through some programs, such as TDP and SPOOK.

Thus,

* BLOCKING OUT MPE COMMANDS VIA UDC'S WITH THE SAME NAME WILL USUALLY FAIL UNLESS THE COMMAND IS :SETCATALOG OR :SHOWCATALOG OR IF YOU ALSO FORBID ACCESS TO MANY HP SUBSYSTEMS AND HP-SUPPLIED PROGRAMS. THIS SEVERELY LIMITS THE USEFULNESS OF THIS METHOD.

Again, I'd like to stress that the :SETCATALOG and :SHOWCATALOG can be blocked out this way, as can (with more difficulty) the :RUN command and some other commands; however, the set of commands still permitted will usually be so small, the method involved so complex, and the chance of penetration so great, that all advantages of the exclusive approach pale in comparison.

By far the best way, in my opinion, of implementing the exclusive approach is by using the existing MPE file, database, and program security features, which is what the next few sections will discuss.

## FILE SECURITY

File security is quite possibly the most sophisticated and the least used and understood security system provided by MPE. If properly handled, it can permit a user to use all MPE commands and all of MPE's power without allowing him to go beyond the confines of his files.

Each file has a so-called "security matrix", an array of information that describes what classes of users can read, write, append, execute, and/or lock a file. Similarly, each group has a security matrix describing the security to be set for its files, and each account also has a security matrix. These security matrices are the things that

LISTDIR2 shows you when you do a LISTSEC (or LISTF, LISTGROUP, or LISTACCT).

When a user tries to open a file, MPE checks to see if the user is allowed to access the file by the account security matrix, by the group security matrix, and the file security matrix. If he is allowed by all three, the file is opened; if at least one security matrix forbids access by this user, the open fails. For instance, if we try to open TESTFILE.JOHN.DEV when logged on to an account other than DEV and the security matrix of the group JOHN.DEV forbids access by users of other accounts, the open will fail (even though both TESTFILE's and DEV's security matrices permit access by users of other accounts).

Each security matrix describes which of the following classes can READ, WRITE, EXECUTE, APPEND to, and LOCK the file:

* CR - File's creator
* GU - Any user logged on to the same group as the file is in
* GL - User logged on to the same group as the file is in and having Group Librarian (GL) capability
* AC - Any user logged on to the same account as the file is in
* AL - User logged on to the same account as the file is in and having Account Librarian (AL) capability
* ANY - any user
* Any combination of the above (including none of the above)

By default, whenever any account is created, access to all its files is restricted to AC (account users only), except for the SYS account, in which Read and Execute is allowed for ANY and Write, Append, and Lock for AC; whenever any group is created, access to all its files is restricted to GU (group users only), except if the group is PUB, in which case access is Read and Execute for AC (all account users) and Write, Append, and Lock for GU (group users) and AL (account librarian); and whenever any file is created, access to it is allowed to everyone. Incidentally, a System Manager can access (in any mode) any file in the system, and an Account Manager any file in his account.

Thus, let us say that you, who build your files in JOHN.DEV, wish other users to be able to read your files. To do this, you have to go to your account manager, get him to allow Read access to the group JOHN.DEV for ANY, and get him to ask the system manager to allow Read access to DEV for ANY. This, needless to say, is rather complicated, and, in fact, most users go the much easier route of just :RELEASEing their files.

However, the problem with :RELEASEing a file is that when you do it, ANYBODY is

allowed to do ANYTHING to the file -- this means read it, write to it, even purge it! And, since doing this is so easy, many files are :RELEASEd and never re-:SECUREd, thus leaving them open for easy tampering by anyone; another contributing factor to this is that ordinary MPE :LISTF does not show whether or not the file has been :RELEASEd, so many people don't even know which of their files are :RELEASEd.

However, if getting the access restrictions on your group and account loosened is so difficult, but :RELEASEing the file makes it wide-open for any kind of access, what is to be done? Unfortunately, the solution is by no means easy.

The first step is to set up all your accounts with all forms of access allowed to ANY; i.e. alter them with a command such as

```
:ALTACCT
accountname;ACCESS=(R,W,A,L,X:ANY)
```

This still leaves a level of security (group security) that will by default protect the file (except for PUB groups, which should thus be built with Read and Execute access for AC instead of ANY), while making the security much easier to waive -- one would need to lift group security only instead of group and account security.

Next, when building each group, consider closely the security that you would wish to put on it. If, for instance, this group consists mostly of files that should be readable by anybody, build it with Read access allowed to ANY. Files can then be protected individually by :ALTSECing them to a more restrictive security level.

Finally, if you :RELEASE a file so that someone can access it, be sure to :SECURE it immediately after the other person is done (unless you don't care about security for that file). It's even better if you have some global file manipulation utility (such as VESOFT's MPEX) with which you can :SECURE all the files in some fileset that have been :RELEASEd.

Thus, some important file security guidelines exist:

* REMEMBER THAT :RELEASE'ING A FILE LEAVES IT WIDE OPEN FOR ANY KIND OF ACCESS; :RELEASE FILES CAUTIOUSLY, AND RE-:SECURE THEM AS SOON AS POSSIBLE.

* TRY TO MAKE IT AS EASY AS POSSIBLE FOR PEOPLE TO ALLOW THEIR FILES TO BE ACCESSED BY OTHERS WITHOUT HAVING TO :RELEASE THEM. THUS, BUILD ALL ACCOUNTS WITH (R,W,X,A,L:ANY) SO THAT ALLOWING ACCESS TO A GROUP WILL BE EASIER.

* IF A GROUP IS MOSTLY COMPOSED OF FILES THAT SHOULD BE ACCESSIBLE BY ALL USERS IN THE SYSTEM OR BY ALL ACCOUNT USERS, BUILD IT THAT WAY. THIS WILL ALSO REDUCE :RELEASE'S.

* THE :ALTSEC COMMAND IS USEFUL FOR RESTRICTING ACCESS TO FILES IN A GROUP TO WHICH ACCESS IS NORMALLY LESS RESTRICTED.

One more aspect of file security that bears mentioning is the file lockword. With it, you could conceivably restrict access to a file to only those users (or programs!) who know the file lockword, even if the file's security matrix says that they have complete access to the file. However, the problem with lockwords is the same as the problem with passwords -- they don't stay secret for long. In my opinion, other security approaches (better use of the security matrices, user id checks in programs being protected, etc.) are superior.

* LOCKWORDS AREN'T ALL THEY'RE CRACKED UP TO BE. OTHER APPROACHES SHOULD BE PREFERRED.

## ASIDE -- ALLOWING PROGRAMS TO READ :SECURE'D FILES

Say that you want your accounts payable program to ask the user for a password and then check the user's input against a password stored in a file. Now, you naturally can't store the password in a :RELEASEd file, for then the password would be readable by anybody; however, if it is stored in a :SECUREd file, then the program won't be able to access it, either, since the program is run by ordinary users.

One solution is to :RELEASE a file, but put a lockword on it. Then, the program could open the file specifying a lockword, but users will not be able to open a file because they won't know the lockword. This is a relatively good solution; however, its flaw is that, like all passwords, the lockword is likely to become known sooner or later. Then, the entire advantage of storing the password in a file, namely that the password can be easily changed,

will be nullified by the fact that the file's lockword can not be easily changed.

A different approach uses an undocumented feature of the FOPEN intrinsic. If FOPEN is called in privileged mode, and the 4 low-order bits of the "aoptions" parameter (third from the left) are set to 15, the file is opened for read access IGNORING ALL SECURITY. This is not a security violation because it requires PM capability (see the CAPABILITIES section); however, since PM need only be granted the program and the group and account in which it resides (which could be PUB.SYS), the program will be able to access the file regardless of who is running it, but most users will not (since the file can thus be :SECUREd).

## CAPABILITIES

There are some MPE capabilities that have a bearing on system security.

Of these, SM and AM are simple to explain and relatively well understood -- they allow one to access (in any way) all files in the system, and the account, respectively.

Some others -- AL and GL -- allow one to establish classes of users (Librarians) that are allowed to access files that other users may not because they can be explicitly allowed access by the security matrices (see FILE SECURITY).

However, the security effects of two other capabilities -- OP and PM -- are often not properly appreciated, much to the detriment of system security.

## OP CAPABILITY

OP capability, which stands for System Supervisor (NOT Operator!), has one capability that has a very large bearing on system security: a user with OP capability can :STORE and :RESTORE any file on the system. This might not mean much, but this really means that

A USER WITH OP CAPABILITY CAN READ AND WRITE ANY FILE IN THE SYSTEM

After all, to read it, all he has to do is to :STORE it and then FCOPY the tape to the line printer; and to write to it, he can store it, move it to a system on which he has write access to the file's group and account, modify it, store it again, and restore it on the original system. Can you trust your operators (who are usually given this capability) with this kind of power?

* YOU SHOULD ONLY GIVE OP CAPABILITY TO USERS WHO YOU TRUST AS MUCH AS YOU WOULD A SYSTEM MANAGER, TO USERS WHO HAVE NO ACCESS TO MAGNETIC TAPES OR SERIAL DISCS, OR TO USERS WHO HAVE A LOGON UDC THAT DROPS THEM INTO A MENU WHICH FORBIDS THEM FROM DOING :STORE'S OR :RESTORE'S

## PM CAPABILITY

No capability has been feared, discussed, or maligned quite as much as PM capability. In this paper, I will only discuss the the security ramifications of PM capability; for a discussion of PM and system crashes, see my paper "Privileged Mode: Use and Abuse".

What does PM capability give you? Quite simply, it allows you to obtain SM capability as follows:

```
:DEBUG
?MDL-'DL-1'+2
DL-NNN     MMMMM := -1
?E
```

Once you do this, you are (at least partially) a system manager until you log off. You can access any file and even execute system manager commands like :ALTACCT and :ALTGROUP to give yourself SM or any other capability permanently.

Obviously, PM capability is not something you want to give to every Tom, Dick, and Harry.

* YOU SHOULD ONLY GIVE PM CAPABILITY TO USERS WHO YOU TRUST AS MUCH AS YOU WOULD A SYSTEM MANAGER.

However, there are other ways in which users can get PM capability.

For one, for a program to have PM capability (and thus use various privileged operating system functions), the program must reside in a group and account which have PM capability. This is very good -- that way, programs like DBUTIL and SPOOK, which use privileged mode, can be run by plain vanilla users who do not have to be given PM. However, this means that if a privileged program does something to circumvent normal MPE security (see the ASIDE -- ALLOWING PROGRAMS TO READ :SECURE'D FILES), it'll do it for anybody who runs

it, unless it explicitly checks who is running it.

More importantly, this means that a user does not need to have PM capability to write privileged programs -- only the ability to build files in a privileged group (i.e. S [Save] access to that group) or to overwrite a program file in that group with his own file (i.e. W [Write] access to any program file in that group) and then run them (i.e. X access to the program file being overwritten or any access if he has S access to the group -- then he can just release the file).

For instance, say that I work out of EUGENE.DEV and the group PROG.DEV has PM capability and Save access for all account users. I can just write a program that uses privileged mode to access a file that I shouldn't be able to access or to grant myself all the capabilities (like in the :DEBUG example above), :PREP it without CAP=PM (since :PREPping with CAP=PM requires PM capability), then change the program file to have PM capability (a task that does not require privileged mode), and copy it into PROG.DEV. Although while the program was in EUGENE.DEV, I couldn't run it (since it is required that the group in which the program is have PM capability), once it is in PROG.DEV, I could run it. If I don't have execute access to PROG.DEV, I can :RELEASE the program before running it, since I'm the creator of the file.

Or, say that somebody :RELEASEd any program file in PUB.SYS, thus giving me write and execute access to it. Then, I can write a program that uses privileged mode to bypass system security, :PREP it without CAP=PM, change the program file to have PM capability, and copy it on top of that program file in PUB.SYS. Then, since PUB.SYS has PM capability and I have execute access to the file I just overwrote, I can run the program.

Thus,

* IF ANY USER HAS SAVE ACCESS TO A GROUP WITH PM CAPABILITY, OR WRITE AND EXECUTE ACCESS TO ANY PROGRAM FILE THAT RESIDES IN A GROUP WITH PM CAPABILITY, HE CAN WRITE AND RUN PRIVILEGED CODE.

And, since :RELEASEing a file gives everyone write and execute access to it,

* *NEVER* :RELEASE A PROGRAM FILE THAT RESIDES IN A GROUP WHICH HAS PM CAPABILITY!

As if this wasn't enough, there are some other potential security violations that can occur with privileged mode. Consider the following circumstance:

Two HP 3000s, which we will call O-machine (intended for OPEN access) and S-machine (which the system management wants SECURED) are linked via DS/3000. A person has a userid and a group with PM capability on O-machine and a plain vanilla userid and group with only default capabilities on S-machine. S-machine management thinks that their machine is secure, since only MANAGER.SYS and PUB.SYS have PM capability on their machine.

Now, there are several file system operations that bypass system security and thus require privileged mode; for instance:

* FOPEN with the 4 low-order bits of aoptions set to 15 (see ASIDE -- ALLOWING PROGRAMS TO READ :SECURE'D FILES), when called from within privileged mode, lets you read a file even when you have no access to it.

* FOPEN with EXECUTE access (4 low-order bits of aoptions set to 6; document in System Intrinsics manual), when called from within privileged mode, lets you read and write a file if you have only execute access to it.

* MUSTOPEN, a procedure identical to FOPEN in all respects except that, when called in privileged mode, it ignores a file's lockword.

* FOPEN of a privileged file (a file with a negative filecode, such as an IMAGE database).

These are not inherently security violations -- in fact, as the ASIDE -- ALLOWING PROGRAMS TO READ :SECURE'D FILES section shows, they can be used to actually INCREASE your security. However, they are not security violations only because they require PM capability to be executed.

Now, consider our would-be security violator. He has his eyes on the S-system file FOO.JOB.SYS, which he knows is a job stream that contains an embedded password (it could just as well contain any other kind of sensitive data). He signs on to O-system as a privileged user, and then to

the S-system via DS as a plain vanilla user. Now, because, DS allows a program on one system to open a file on another system (by specifying the file's device to be the dsline device followed by a "#", e.g. "60#"), our user writes a program on O-system that opens file FOO.JOB.SYS in the "ignore security mode" (aoptions 4 low-order bits = 15) on S-system. Since the program is running in privileged mode (remember, our O-system user is privileged), the open succeeds, and the user can read the file!

Now note that the file system does not check that the user on S-system must have PM capability to use this security-bypassing mode; the program need merely be running in PM capability, regardless of which system it is on!

This is one of the few genuine flaws in MPE's security system, and it's nothing to sneeze at. What it means is that

* IF TWO HP3000'S ARE CONNECTED VIA DS, AND A USER HAS PM CAPABILITY ON ONE AND AN ORDINARY LOGON ON THE OTHER, HE CAN VIOLATE THE OTHER'S SECURITY. THUS, IF ANY HP3000 IN A DS NETWORK IS BROKEN INTO OR LEFT OPEN, ALL OTHERS ARE IN GRAVE DANGER.

Thus, if you want to keep one system secure, you must keep all systems hooked up to it via DS secure as well.

One other issue, somewhat more arcane but nonetheless relevant arises when using privileged mode.

If a program which has PM capability calls DEBUG when the user running it does not have PM capability, even though the user will be dropped into non-privileged DEBUG, he can use it to break system security.

Briefly, the user can modify some data in the program's stack or the program's P pointer (which points to the current instruction being executed) to cause the program to do something other than what is supposed to do when it performs its privileged operations. One thing that actually happened to one of my programs is that it called the WHO intrinsic, figured out the logon user, account, and group, put them into global arrays, and then went into privileged mode and got the logon user, account, and group passwords and wrote them to a stream file. This was perfectly kosher -- if a user managed to sign on, he already knows his logon passwords; however, the program allowed the user to enter DEBUG even if he was non-privileged. Although the program did not call DEBUG when privileged, and the user was not put into privileged debug, the user could modify the user, account, and group id arrays in the stack to read, say, "MANAGER", "SYS", and "PUB". Then, the next stream the program built would contain MANAGER.SYS's passwords!

This is, as I said, a rather arcane and relatively infrequent problem; however, it is a possible security flaw nonetheless, and should not be ignored. In fact, I'd like to ask HP to correct their DBDRIVER program, which is privileged and has a "/D" command which drops the user into DEBUG whether or not he is privileged.

In the same vein, dynamically loading (via the LOADPROC intrinsic) a procedure from a user's group or account SL and then calling it should also be forbidden to privileged programs -- the called procedure, even though it resides in a non-privileged SL, can call GETPRIVMODE because the program calling it is privileged. Again, rather arcane but still worth noting.

Thus,

* PRIVILEGED PROGRAMS MUST NEVER CALL DEBUG UNLESS THEIR USER IS PRIVILEGED, AND MUST NEVER DYNAMICALLY LOAD AND CALL PROCEDURES FROM A USER'S GROUP OR ACCOUNT SL UNLESS THE USER IS PRIVILEGED.

Now, I do not intend to unfairly malign PM capability. It has its uses, and in fact, some programs must have it (such as the HP system utilities in PUB.SYS or many very useful contributed and vendor-supported programs). However, and I can not stress this enough, use of PM must be watched very carefully if you wish to keep your system secure.

## IGNORANCE SECURITY

Many techniques of violating system security described herein may appear rather complicated and improbable; in fact, they are. It is all too easy to say: "Well, my users aren't so smart -- they'd never think of pulling all those tricks". Unfortunately, it is of such complacency that insecure systems are born. After all, if we could think of these tricks, why can't some smart guy in your shop? What if he reads this paper? What if one of his friends is a sophisticated HP user? The assets of your

company are far too precious a thing to entrust to the presumed ignorance of your users; you should rather improve the security of your system, so that even a smart user will not be able to penetrate it -- and if your users aren't that smart, all the better.

## DATABASE SECURITY

IMAGE/3000's security system is probably one of its most complex features and also one of its least used. My first impulse was to chastise the HP user community for not using this wonderful security feature more, and to blame 99.44% of all security violations on their failure to do so, but then I realized that this is not such a wonderful facility after all.

IMAGE/3000 security permits the database creator to restrict access to each individual data item and data set to only those users who specify a certain password when opening the database. Admittedly, this is a very useful feature when you expect the database to be accessed via QUERY -- then you can define exactly what a user can do by what password you give him. However, most databases are accessed by application programs, not through QUERY, and most of the time it is the program, not the user, that specifies the password. So, unless you intend to reveal certain database passwords to only certain programmers and thus protect your database against your programmers, not your users, you are probably far better off implementing application security, i.e. having your application figure out what a certain user is authorized or not authorized to do, rather than using IMAGE security.

* IMAGE/3000 DATABASE SECURITY IS NOT PARTICULARLY USEFUL EXCEPT FOR PROTECTING DATABASES AGAINST UNAUTHORIZED QUERY ACCESS. IN FACT, SOME DEGREE OF PROTECTION AGAINST UNAUTHORIZED QUERY ACCESS CAN BE GIVEN BY USING DBUTIL'S "SET SUBSYSTEM" COMMAND TO DISALLOW ANY QUERY ACCESS OR QUERY MODIFICATION OF A DATABASE.

## DATA ENCRYPTION

If you want to secure your data against unauthorized reading, you need not prevent anybody from accessing it if, even if they manage to access it, they won't be able to understand it. This is the principle of encryption -- change the format of your data so that nobody except for the authorized people will be able to understand it.

Usually, encryption algorithms involve the use of so-called "keys". Say that I want to encrypt the phrase "NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR COUNTRY". I could do this by choosing some number (say, 7) and adding it to each letter of the sentence, so that A would become H, B would become I, C would become J, R (#18) would become Z, S would become A, etc. Then, the phrase would become "UVD PZ AOL APTL MVY HSS NVVK TLU AV JVTL AV AOL HPK VM AOLPY JBVUAYF", an unreadable jumble of letters to anyone who doesn't know that to decrypt it, one must subtract 7 from each character. Thus, 7 is the key and the encryption algorithm is to add the key to each character.

Unfortunately, things are a bit more complicated than that, primarily because with some work, one can realize that the letters A and V occur quite often, the combination AO occurs frequently as well, and that there are only so many possible two-letter words (some of which must correspond to PZ, AV, and VM). Thus, we could find out what key letters correspond to, and thus decode the entire sentence.

Fortunately, there are more sophisticated encryption algorithms that are far harder to decrypt. And, since the key need not be stored on the computer, but only in the user's mind or some other safe place, encrypted data can only be decrypted by an authorized person.

Another, less general but nonetheless useful technique for encrypting passwords is called "one-way encryption". Say that you wish a user to enter a password into your program when he is first set up, and then have your program ask him for the password every time he subsequently logs on. You do not need to actually decrypt the password -- just encrypt it once at user set-up time, store it in encrypted form, and then, every time the user tries to log on, ask him for a password, encrypt his answer, and compare it against the encrypted real password.

Thus, your encryption algorithm can map the entire password into a single number (by, say, adding the squares of all the letters, each multiplied by the cube of its position in the password string), thus making it impossible to decrypt; and, the encryption algorithm is much simpler than two-way encryption algorithms that need to have a corresponding decryption

algorithm. Unfortunately, this technique is limited to applications in which decryption is never necessary, such as when passwords are stored.

## SECURITY

One-way encryption is easy to do; good two-way encryption is harder -- I know of no HP programs that do it, but hopefully that will be remedied soon.

* IN GENERAL, ENCRYPTION IS ANOTHER GOOD WAY OF PROTECTING SENSITIVE DATA FROM UNAUTHORIZED READING.

## CONCLUSION

It is all too easy to get involved in the implementation and perfection of an application system, putting "little things" like security on the back burner; unfortunately, this is precisely what accounts for the alarming amount of computer crime that is threatening us today. What is best is that with application of some simple guidelines and a little time and effort, you could dramatically decrease your chances of becoming a victim. No security system will cut these chances to zero, but if you have as much valuable data in your machine as the average HP user has in his, doing nothing can literally cost you millions.

## APPENDIX A: SUMMARY OF USEFUL HINTS

* THE USER IS THE WEAKEST LINK IN THE LOGON SECURITY SYSTEM -- DISCOURAGE HIM FROM REVEALING PASSWORDS (by techniques such as personal profile security or even by reprimanding people who reveal passwords -- they seem innocent, but they can lose you millions).

* PASSWORDS EMBEDDED IN JOB STREAMS ARE EASY TO SEE AND VIRTUALLY IMPOSSIBLE TO CHANGE -- AVOID THEM.

* SOME FORMS OF ACCESS ARE INHERENTLY SUSPECT (AND THUS REQUIRE EXTRA PASSWORDS) OR ARE INHERENTLY SECURITY VIOLATIONS. THUS, ACCESS TO CERTAIN USER ID'S AT CERTAIN TIMES OF DAY, ON CERTAIN DAYS OF WEEK, AND/OR FROM CERTAIN TERMINALS (SUCH AS DIAL-IN OR DS LINES) SHOULD BE SPECIALLY RESTRICTED.

* MANY SECURITY VIOLATIONS CAN BE AVERTED BY MONITORING THE WARNINGS OF UNSUCCESSFUL VIOLATION ATTEMPTS THAT OFTEN PRECEDE A SUCCESSFUL ATTEMPT. IF POSSIBLE, CHANGE THE USUAL MPE CONSOLE MESSAGES SO THEY WILL BE MORE VISIBLE.

* LEAVING A TERMINAL LOGGED ON AND UNATTENDED IS JUST AS MUCH A SECURITY VIOLATION AS REVEALING THE LOGON PASSWORD. USE SOME KIND OF TIMEOUT FACILITY TO ENSURE THAT TERMINALS DON'T REMAIN INACTIVE FOR LONG; SET UP ALL YOUR DIAL-IN TERMINALS WITH SUBTYPE 1.

* A USEFUL APPROACH TO SECURING YOUR SYSTEM IS TO SET UP A LOGON MENU WHICH ALLOWS THE USER TO CHOOSE ONE OF SEVERAL OPTIONS RATHER THAN TO LET THE USER ACCESS MPE AND ALL ITS POWER DIRECTLY.

* BLOCKING OUT MPE COMMANDS VIA UDC'S WITH THE SAME NAME WILL USUALLY FAIL UNLESS THE COMMAND IS :SETCATALOG OR :SHOWCATALOG OR IF YOU ALSO FORBID ACCESS TO MANY HP SUBSYSTEMS AND HP-SUPPLIED PROGRAMS. THIS SEVERELY LIMITS THE USEFULNESS OF THIS METHOD.

* REMEMBER THAT :RELEASE'ING A FILE LEAVES IT WIDE OPEN FOR ANY KIND OF ACCESS; :RELEASE FILES CAUTIOUSLY, AND RE-:SECURE THEM AS SOON AS POSSIBLE.

* TRY TO MAKE IT AS EASY AS POSSIBLE FOR PEOPLE TO ALLOW THEIR FILES TO BE ACCESSED BY OTHERS WITHOUT HAVING TO :RELEASE THEM. THUS, BUILD

ALL ACCOUNTS WITH (R,W,X,A,L:ANY) SO THAT ALLOWING ACCESS TO A GROUP WILL BE EASIER.

* IF A GROUP IS MOSTLY COMPOSED OF FILES THAT SHOULD BE ACCESSIBLE BY ALL USERS IN THE SYSTEM OR BY ALL ACCOUNT USERS, BUILD IT THAT WAY. THIS WILL ALSO REDUCE :RELEASE'S.

* THE :ALTSEC COMMAND IS USEFUL FOR RESTRICTING ACCESS TO FILES IN A GROUP TO WHICH ACCESS IS NORMALLY LESS RESTRICTED.

* LOCKWORDS AREN'T ALL THEY'RE CRACKED UP TO BE. OTHER APPROACHES SHOULD BE PREFERRED.

## APPENDIX A: SUMMARY OF USEFUL HINTS

* YOU SHOULD ONLY GIVE OP CAPABILITY TO USERS WHO YOU TRUST AS MUCH AS YOU WOULD A SYSTEM MANAGER, TO USERS WHO HAVE NO ACCESS TO MAGNETIC TAPES OR SERIAL DISCS, OR TO USERS WHO HAVE A LOGON UDC THAT DROPS THEM INTO A MENU WHICH FORBIDS THEM FROM DOING :STORE'S OR :RESTORE'S

* YOU SHOULD ONLY GIVE PM CAPABILITY TO USERS WHO YOU TRUST AS MUCH AS YOU WOULD A SYSTEM MANAGER.

* IF ANY USER HAS SAVE ACCESS TO A GROUP WITH PM CAPABILITY, OR WRITE AND EXECUTE ACCESS TO ANY PROGRAM FILE THAT RESIDES IN A GROUP WITH PM CAPABILITY, HE CAN WRITE AND RUN PRIVILEGED CODE.

* *NEVER* :RELEASE A PROGRAM FILE THAT RESIDES IN A GROUP WHICH HAS PM CAPABILITY!

* IF TWO HP3000'S ARE CONNECTED VIA DS, AND A USER HAS PM CAPABILITY ON ONE AND AN ORDINARY LOGON ON THE OTHER, HE CAN VIOLATE THE OTHER'S SECURITY. THUS, IF ANY HP3000 IN A DS NETWORK IS BROKEN INTO OR LEFT OPEN, ALL OTHERS ARE IN GRAVE DANGER.

* PRIVILEGED PROGRAMS MUST NEVER CALL DEBUG UNLESS THEIR USER IS PRIVILEGED, AND MUST NEVER DYNAMICALLY LOAD AND CALL PROCEDURES FROM A USER'S GROUP OR ACCOUNT SL UNLESS THE USER IS PRIVILEGED.

* IMAGE/3000 DATABASE SECURITY IS NOT PARTICULARLY USEFUL EXCEPT FOR PROTECTING DATABASES AGAINST UNAUTHORIZED QUERY ACCESS. IN FACT, SOME DEGREE OF PROTECTION AGAINST UNAUTHORIZED QUERY ACCESS CAN BE GIVEN BY USING DBUTIL'S "SET SUBSYSTEM" COMMAND TO DISALLOW ANY QUERY ACCESS OR QUERY MODIFICATION OF A DATABASE.

* IN GENERAL, ENCRYPTION IS ANOTHER GOOD WAY OF PROTECTING SENSITIVE DATA FROM UNAUTHORIZED READING.

*BIOGRAPHY*

*Eugene Volokh was born February 29, 1968 in Kiev, USSR. His family moved to the U.S.A. in 1975. In 1979 Eugene started his work with HP equipment. Eugene is now a senior consultant at VESOFT, Inc., a company he cofounded with his father, Vladimir Volokh, in 1980 (their products are MPEX/3000 and SECURITY/3000). This year Eugene graduated from UCLA with a B.S. degree in computer science.*