The Philosophy of Computer Systems Management

An Honors Thesis
(ID499, CS499)

by
Paul Frommeyer

Thesis Director
Dr. J. Michael McGrew

Ball State University
Muncie, Indiana
5/20/87

Spring, 1987
This paper was prepared on a VAX-11/785 running 4.3BSD Unix.
It was printed on an Apple Laserwriter using TranScript from Adobe Systems.

VAX, VMS, DCL, and RMS are trademarks of Digital Equipment Corporation.
Unix is a trademark of AT&T Bell Laboratories.
Apple is a trademark of Apple Computer Inc.
TranScript is a trademark of Adobe Systems Inc.

Copyright 1987, Paul Frommeyer
All rights reserved unless otherwise stated
For Digital, who built the better mousetrap
"The more complicated they make the plumbing, the easier it is to stop up the drain."

--Chief Engineer Montgomery Scott, *Star Trek III: The Search for Spock*
Table of Contents

I. .................. Excuses and Disclaimers

II. .................. The Battlefield

III. .................. To Be or Not to Be

IV. .................. Gods and Mortals

V. .................. Mob Rules

VI. .................. Shutdown

A. .................. References
Conventions

In this book I have adopted several conventions which I have seen, but which may not be strictly standard. The first is page numbering for references. Something like "17-2" is misleading, because for a number of my sources this refers to chapter 17, section 2, not pages 17 thru 2. To indicate multiple pages, I use "17-2,6" to refer to pages 2 thru 6.

Second is my citation of references. I did not use footnotes, simply because I feel cluttering up the bottom of the page is foolish. Endnotes are far more preferable, especially since you do not need to keep track of your references, or waste troff(1) time doing so. However, in using end notes I decided to identify each source by a relevant abbreviation, such as VVI for VAX/VMS Internals and Data Structures. This allows a better mnemonic association for the reader than numbers; [5, p. 6] doesn't say much, and the reader must continually flip to the back of the book for source correlation. Instead, the astute reader will know early on that VVI means VAX/VMS Internals, VSH means Vax Software Handbook, etc., with only minimal reference to the back of the paper.

I also chose to use a slightly different method of citation for command language manuals. Instead of something like [VDCL, p. 20] I have chosen rather to use [VDCL, ANALYZE]. I did this because both VMS and Unix documentation are organized alphabetically by command. In fact, in Unix documentation this is the ONLY way to cite the page reference for a command, since pages are numbered individually for each manual section.

Another point is my use of the feminine. Rather than resort to such extremes as s/he, but still finding the Usenet convention of varied gender in pronouns acceptable, I have chosen instead to alternate between the feminine and masculine when need arises to refer to a person holding a position.

Lastly I vary between italics and all caps purely as a matter of style; literary license at work. More literary license is my use of dangling modifiers and sentence fragments, anathema to good English but great for emphasis and something that abounds in creative writing.
"The price one pays for pursuing any profession, or calling, is an intimate knowledge of its ugly side."
--James Baldwin

Chapter I
Excuses and Disclaimers

This is sort of the introduction wherein I will try to anticipate forthcoming criticisms. First, I am going to try and explain what I am trying to do with this paper. I figure that what I am not trying to do will become clear once the criticisms start descending.

It was my aspiration to become a system manager for a computer system as rapidly as was possible, during college if not sooner. As I was soon established in the Digital camp of the great IBM-DEC war, I busied myself reading VAX manuals when I wasn’t "studying" VAX/VMS operating system security. When the time in my studies came when I had to do a paper, systems management seemed the logical choice for a topic. Security systems ran a close second, but systems management won out for several reasons.

I felt that I had railed and complained about bad systems management throughout my undergraduate studies so much that it was
about time I put some money where my mouth was.

I also was tired of railing and complaining and decided that if I could get all my nigglings and annoyances down on paper, perhaps they would leave me alone and I could go back to cursing COBOL.

This is not, as you may suspect, an autobiography of my undergraduate computing experiences. Colorful though they were.

I had read a number of publications on systems security, operating system design, system implementation, etc. I was also exposed to software engineering concepts, and the idea of controlled design and implementation. What I had yet to encounter was something that dealt with systems management policy as opposed to its implementation.

Occasional related articles drifted by me (unfortunately I cannot keep abreast of all trade publications), but there may be a lot of stuff on this topic I am not even aware of, much less haven't read.... However, the only document I had yet run across that attempted to deal squarely with goals and policies of systems management was the VAX/VMS System Manager's Reference Manual.

This elegant tome by Digital expressed concisely and clearly the goals of system management policies, how they should be implemented, etc., etc. No attempt was made to elaborate on personnel
selection, and overall resource management strategies, but rather Digital chose to concentrate solely on the management of the system rather than the management of the organization the system served.

This seemed to me an admirable achievement, and in addition to being my primary source for this paper the concepts of the manual were the inspiration for its writing. However, in the course of my work and studies, I encountered the Unix operating system. And with it, a radically different approach to systems management than the one ascribed to by Digital. It was obvious that any system manager worth his salt would realize that he was faced with a policy dilemma. A large policy dilemma.

The dilemma, simply stated, is "Just how secure should a computer system be?"

Pick up any book on system security, especially some of the addresses by AT&T and our august Congressional Committees, and the answer to this problem will be clearly stated. Lock it up. The tighter, the better.

On the other hand are the Unix system manuals, which advocate an open, friendly, insecure environment.

Clearly, these are widely different policies. But it also seems
that there are vastly different management philosophies behind the two views. The first seems to be concerned with defining the role of the user in terms of the system. The other seems to be just the opposite, making system management policy serve the user rather than the other way round. These views are not compatible, though they may each be valid in different contexts.

I will try and address this contradiction, and present several key aspects of system management, each from the point of view of the two philosophies. In addition, I will trace the origin of the two philosophies, then show its impact on system management as a whole. I may touch on side topics such as hacking and system security, but my primary emphasis will be on discussion of the two different systems management philosophies, and how they affect systems management. I cannot deal fully with side issues, because to do so would require a rather large book.

Also, like Digital, I am going to avoid entirely the issue of personnel and department management. I will deal with personnel only as far as they are a direct part of the management of the system. How they fit into whatever bureaucratic hierarchy exists is somebody else’s problem.

Chief among this paper’s criticisms, I’m sure, will be my use
of casual style. I've read manuals and books that were stuffy enough to cause anoxia. And I've studied some excellent literature full of wisecracks and tongue-in-cheek. And it seems to me that the latter make much better reading, releasing the reader from the tedium of reams of statistics and buzzwords. Or at least providing a break from the monotony. I can cite several computer science texts, as well as a few trade publications, which freely advocate such style. I am sure there are still those who feel that written language should differ from spoken language, but I feel this sufficient justification of my stylistic choices.

Before launching into the fray of my text, I feel some clarification and definition is necessary. I have used the term "system security" several times so far. (This is one of those words that I'm sure Humpty Dumpty really enjoyed) By system security, I refer to the degree to which confidentiality and privacy of critical information may be maintained in a computer system. Note that this is not the same as reliability, which refers essentially to how easy it is to crash a computer system. I am adopting these definitions for use in this paper. Other professionals (or hackers) often use other terms to refer to these two concepts.

"Hacker" as a word has come to have two different definitions. Originally, the word referred simply to a person who was at
least intensely interested in if not obsessed with computers. A hacker was someone seen simply as grafting themselves to a computer terminal-- for whatever purpose. However, thanks to the efforts of the infamous "414s" and the movie War Games, the word hacker has taken on a more sinister meaning. Now, a hacker is someone who seeks to defeat the security of a computer system for purely malicious reasons.

Unfortunately, this has resulted in individuals interested in the inner workings of a computer being stereotyped as malicious security breakers. The situation is not helped by the inherently flimsy design of some operating systems which will crash at the least provocative system call. Nor are things improved by the paranoia of system administrators who feel that knowledge of system internals are "top secret", and that anyone not authorized who understands how the system's security functions are implemented should be summarily shot.

Bearing these things in mind, I wish to make it clear that my usage of the word hacker in this thesis refers to anyone attempting to compromise the integrity of a computer system, at whatever level. I am not, however, saying that anyone attempting to compromise a system is doing so for malicious reasons. Quite the contrary, I hold that such activity can be essentially innocent. I feel quite strongly that there is a serious flaw in the perspective on attempts at system compromise. Far
from punishing the successful hacker, she should be thanked for pointing out such a serious flaw in the system. Anyone who knows that much about computer systems is clearly an asset, and should be hired at once. Indeed, several large firms have already followed this example.

Do not misconstrue this as support of illegal activities. The successful, benign hacker is quite different from the malicious variety, or its related species, the software pirate. Laws must be enforced. Neither the rampant destruction of crucial data nor the flagrant violation of copyright laws can be tolerated. (Leaving the question of the inanity of such laws for another time.) I merely wish to distinguish between malicious computer assault and essentially "harmless" attempts to compromise a system simply to see if it can be done.

In this discussion, then, hackers are considered to be the benign variety (probably students in a university environment), interested in pitting their knowledge of a computer system against its claimed security just to see if they can compromise the system. But to be sure all bases are covered, I'll also clarify the role of a system manager. I use the term as set forth by Digital: "...the management of a system has as its ultimate goal delivering efficient economical service to all users" [VSH, p. 434]. In this context, the system manager is the person responsible for achieving this end. Her place in the management
hierarchy of her employer is, again, beyond the scope of this paper.

These sundry items out of the way, we move on to examine the security policy conflict.
"No man can serve two masters."
--Matthew 6:24

Chapter II
The Battlefield

"Effective system management depends to a large degree on effective user management. The control of system functions on a user-by-user basis plays a vital role in managing system resources and thus enhancing system efficiency and security." [VSM, p. 5-1] This is the stated goal of system management policy as presented by Digital Equipment Corporation in the System Manager's Reference Manual: control of system functions on a user-by-user basis. And, indeed, the VAX/VMS operating system supports this goal to the fullest. When one examines the attention to detail and the precision with which the facets of VMS are brought together to provide a unified, secure, reliable system, one can only admire Digital for their effective synthesis of an operating system with system management functions.

The VMS operating system provides extensive functions for system managers to provide resource and user control for the system. The AUTHORIZE utility, which maintains the User Authorization File (UAF), containing each user's account information, is an extensive
program allowing manipulation not only of the UAF, but also of a rights access database and a network access control file. In addition, such utilities as ACCOUNTING and DISKQUOTA provide means for monitoring and limiting user resource consumption not otherwise under the jurisdiction of AUTHORIZE.

Just what was the design goal that resulted in such a secure, reliable system in the midst of the birth of so many patchwork operating systems? "The demands of software were central to the design of the architecture... We made sure that the VAX architecture enhanced the efficacy of the VAX/VMS operating system..." [VSH, p. vii] says the Vax Software Handbook. Clearly, then, the design of VMS was a carefully planned and well-executed project. But what was the idea behind this piece de resistance in operating systems?

Examining these perspectives, it seems that a secure, reliable system was the goal of the VMS design effort. And it was a successful design, for VMS has attained a rank as an industry standard for ease-of-use and reliability. Providing standardized command format and extensive recovery features, VMS stands as a leading operating system [VSH, p. vii].

However, there seems to be another, underlying consideration throughout these methods. Namely, how such design goals will be
achieved, and how they will be carried to the user. Digital says that "A computer installation exists to provide its users with efficient and economical service. The challenge of system management is to provide such service on a day-to-day basis" [VSM, p. 1-1]. How, then, is this carried out?

All Digital system management, and the VAX/VMS operating system itself, is based on the principle of least access. This states that no system user shall have any access or privileges that are not necessary to the performance of her task [CSH, p. 216]. VMS incorporates thirty-five privileges allowing scaled access to the system. In addition, there exist numerous quotas and other resource controls [VSM, Ch. 6].

It is the actualization of the principle of least access that makes VAX/VMS such an effective and efficient operating system. By maintaining maximum data security, disruption of the system's operation or the activities of the institution it serves is minimalized. By enforcing strict resource and privilege control, forbidding casual users to affect system operation, operating system integrity is maintained, as is the reliability of the system. Lastly, by employing extensive resource control, VMS ensures that unneeded resources are not consumed, preventing the disruption of computer activity and the wastage of system
resources by careless or incompetent users. [VSM, p. 6-7]

These features of VMS are admirable, creating a highly secure, highly reliable computing environment suitable for business or industry use. Because of its attention to these details, VMS has become an industry standard. With several million dollars behind its development, the quality of VAX/VMS is unquestionable.

I'll turn now to the underlying philosophy behind this approach to systems management. It seems clear from the VAX system design goals as well as the design of the VAX/VMS operating system itself that the central issue here is control. The better the control over the system, and the users, the better the system management and hence the better the performance of the system. Fatal events are anticipated and provided for; the system is designed to "gracefully degrade" when placed in high demand or crash environments. Such a design philosophy assumes a priori that there will be problems, and it attempts to provide facilities for handling them as best it can [VSH, p. 1].

Furthermore, from the principle of least access alone it is apparent that this philosophy is highly distrustful of the user. Again, it assumes a priori that every user is out to cause trouble. For whatever reason, users will attempt to defeat security, consume system resources, and generally create havoc. The principle of least access is designed to
minimize, if not prevent entirely, any damage the user may (intentionally or inadverdently) cause. [VSM, p.6-7]

In addition, the "Big Brother" image characterized in George Orwell's 1984 seems to be the set role for the system manager. Such extensive user auditing and control facilities are provided that it seems that VMS was written with the system manager in mind, and the user as a secondary consideration.

But to some degree this is true. It may be one thing for academicians to rail at these strictures, but in a business computing environment, these features are not merely nice, they are a necessity! Perhaps Frank and Irene's Florist Shoppe isn't too concerned about data integrity, security breaches, and authorized access, but you can just bet the United States Armed Forces are. (Digital has numerous major military accounts, no doubt also a design factor.) Moreover, such flexibility in user accounting and control is desirable also on Wall Street, or indeed in any going concern in which sensitive data are manipulated with the computer. [CSH, p. 7,8]

It seems, however, Digital recognized that at times users would have to perform special functions not normally available to them. To this end, VMS has been designed with gradiated control and access, the degree of each being determined by the system manager. This
allows a fine granularity in the implementation of the least access principle, providing for a wide variety of user needs.

This whole philosophy I will henceforth call the "VMS philosophy", though this is intended to say nothing about Digital nor the authors of VMS. I use it only as a distinguishing handle, since in this paper I am using VMS to exemplify the philosophy. IBM was not chosen as I felt that both IBM design and implementation were a good example of how not to construct an operating system. A view born out, I think, by the fact that Digital was the only computer industry member to show a profit for FY86.

However. What happens if you want an open system, what happens if you don't really care about extensive security and operating system efficiency? What if all that is desired of the computer is that it provide an extensive program development environment, with easy access to the operating system?

VMS can fulfill this goal, but that is throwing the baby out with the bath water. The whole purpose of VMS is to provide high security and efficiency; indeed, it becomes difficult to convince the proprietary Digital operating system to permit free file access! You can't, for instance, create files across volumes if you have no disk quota on the volume being accessed. In addition, the scaled privileges are
really only good for program images; it turns out that if you have one privilege for system access, you can generally get the others. (As an example, a person with PHY_IO privilege, which allows reading and writing to physical disk blocks, can toddle on in and change the page or swap files, effectively altering suspended processes. It is a simple matter to create a process, suspend it, then alter its workspace such that when it resumes execution it has complete access to the entire system.)

It was without such restraints and restrictions that the Unix operating system was created. First designed in 1969 by Ken Thompson, Dennis Ritchie, and Brian Kernighan at Bell Labs, the Unix system has risen in popularity until it may well become an industry standard. Written mostly in the medium-level language C, Unix provides an open, friendly environment conducive to programming research and development. It does not distinguish among process classes, nor does it provide extensive operating system optimization. Yet its innate design of terse elegance is visible at every level of the system. [OS, p. 481,483]

Unlike VMS, the simple design goal of the Unix system was program development. It was never intended as a mass-market, wide-distribution operating system. It got its start as an in-house system tailored to the needs of a select few. That the select few grew and grew until Unix is now an industry giant is tribute to the genius and insight of
the original programming team. [BY, p. 162]

That the design goal was programming is sometimes painfully obvious. Unix to the uninitiated is cryptic, terse, and totally unhelpful. To a new user, Unix can be an impossible Everest; often, it is no less an obstacle to the experienced programmer. "Because Unix was not developed for the uninstructed user, a beginner cannot take advantage of much of its computing power" [BY, p. 162]. Yet once the initial unfamiliarity is overcome, Unix proves to be truly fertile ground for not only programming development, but also (with its growth and slow maturity) for a host of other MIS and office automation tasks [BY, p. 163].

Unix, however, was written with no efficiency considerations, nor any thorough attention to security. It was not designed in committee, but by that best of all programming agencies, a few gifted people. Some of the system data structures are cumbersome, and the command syntax is cryptic at best. These are common criticisms of Unix, but it must be kept in mind that efficiency and user friendliness were never part of the design goal of the system. There is no 'make' command in VMS, nor is there any syntax checking in VAX C. It is clear where the design effort lay. [OS, p. 482,483]

It seems to be the general consensus, however, that Unix is an
inherently insecure, unreliable system. This is anything but the case. Unix, though designed ten years earlier with no direction, provides security exceeding the earlier versions of VMS (which employed several Unix design concepts), as well as proprietary security features that have left Digital at a loss. Unix security is not to be contested, though it is far more at the discretion of the system manager than is VMS. [USS, p.1]

In reliability, though, VMS clearly has the lead. Unix was designed with no graceful degradation. There is an all or nothing philosophy that pervades all the aspects of the system. If there is a fatal error, the kernel doesn’t thrash several thousand lines of code; it just quits. [USM, all] Debugging of the puzzle is left to the systems personnel. Similarly, disk file system corruption is a plague in Unix; the sophisticated error handling of VMS minimizes the risk of file loss. Nonetheless, a good Unix system manager can turn cartwheels around his VMS counterpart in numerous situations. The tools and capabilities of Unix, while simple, can often be combined into a formidable array far outperforming any native VMS software.

The all or nothing philosophy also extends to control of the Unix system. Just as there is no provision for graceful degradation, similarly there is no provision for scaled access. There is one process,
root, called the superuser. A person logged in as root can do anything it is possible to do on the system. There are no privileges; you either are the superuser, or you aren't. All privileged system software runs as root; a user who hacks the system into giving him a root ID can do anything he wishes. [USS, p. 106]

Usually only the system manager has the root password; only a specified group of users are allowed to change their process ID to root at all. The theory is that since there is no way to lock the system manager out, and since he needs all privileges anyway, why bother with complicated protection schemes? In the light of the former note on VMS privilege integrity, this seems to be a valid point.

The Unix system design and management philosophy is now clear. (I will henceforth refer to this as the "Unix philosophy" for the same reasons stated above for the VMS philosophy.) First is the idea that rather than prevent disaster, the system should be recoverable from a disaster. Why try and stop the inevitable? Just make sure it doesn't kill you.

A second and major point is that Unix philosophy is trustful. It is a routine matter for Unix users to submit software for installation on the system. Development of operating system utilities is encouraged; traditional Unix provides the operating system source code on-line (for
VMS it’s on microfiche), as well as giving extensive documentation of operating system internals in the manuals (which are also on-line). The Unix philosophy respects the users; it acknowledges that they are smart enough to know that if they mess up the system, they are not just inconveniencing other users, but they are also inconveniencing themselves. In short, it treats the users responsibly.

In addition, it assumes at least a limited camaraderie among the user community. You are not going to delete X’s files, because X is your friend (or at least your associate). Whether academic or business, you are all in this together. Again, Unix philosophy centers on user-consciousness, the idea that the users realize it is their system, and for that reason will take good care of it.

All this is perfect for a development or research environment. In such a setting comingling of ideas and thoughts flourishes, and indeed is promoted by the Unix system. Such secure data as there is can be protected; otherwise, users are free to roam and explore the system. Creativity and self-discovery are encouraged.

This, of course, is exactly what you don’t want if you are a stock broker, a bank, or the Federal Government. The last thing you want is a secretary figuring out how the accounts payable is processed, or an operator puzzling out the intricacies of the nation’s Strategic
Defense Initiative. You not only want unauthorized personnel to be ignorant of what goes on in areas of the system that are not their concern, you want to make sure they can’t figure it out even if they want to. [USS, p. 5] For this end, Unix is a problematic system, as it still possesses enough bugs and nuances even at this late date to pose a security threat. (Not to say that VMS doesn’t, but it’s not as easy to find them...).

These then are the two system management philosophies. On the one hand is a highly secure, reliable environment designed to protect the system from the user. On the other hand is an open, friendly environment which makes the user himself, in a sense, responsible for the system. Which of these two philosophies predominates in the management of a given system will, clearly, have maximal impact on how that system is configured.

In particular, the following areas of system management will be dramatically affected: Access to the system, and the provision of user accounts. System security, and the control of the system by the users and systems personnel (resource management), at all levels. I'm going to discuss the question of access first, in terms of both philosophies, so we can kind of get a feel for them. Then, I'll trace both philosophies to their root, and show how they each arose and what method should be
used to implement each. Lastly, I'll present each philosophy in terms of its key component... resource management.
Open here I flung the shutter, when, with many a flirt and flutter,  
In there stepped a stately Raven of the saintly days of yore;  
Not the least obeisance made he; not a minute stopped or stayed he;  
But, with mien of lord or lady, perched above my chamber door—  
Perched upon a bust of Pallas just above my chamber door—  
Perched and sat and nothing more.  
-- Edgar Allen Poe, *The Raven*

Chapter III
To Be Or Not To Be

Of paramount importance in any computer system is the question of who has access to it. If you cannot get to the system, you cannot damage it. Similarly, if you can’t get to it you can’t use it. The question of access is a central one; before any discussion of operating system security or resource management can take place, it must first be determined who is going to have access to the computer system in question, and, more importantly, just how much access they will have.

There are three main areas of access that are affected by systems management philosophy: User access, system personnel access, and, lastly, availability of old accounts.

I’ll address the question of user access first. I define a user here as anyone who is employing the system to accomplish some directed end. He differs from system personnel in that his directed end is unrelated to the maintenance and operation of the system itself. There
are exactly two kinds of access the user may have to the system: hardware access and software access.

Software access refers to those aspects of the operating system and associated files the user may manipulate. In both systems, all access is determined by one file. In Unix it is the system file /etc/passwd (the password file); in VMS it is the file SYSUAF.DAT (User Authorization File). Both files contain the information that tells the operating system how to configure the user process when it is created. Data relevant to each file will be introduced as needed. Suffice it to say that the files contain the user's username (his name on the system), his password (to get him into the system), his home directory of files, the group of users he belongs to, and what program is used to interpret operating system commands. Such access can take two forms under both VMS and Unix: the restricted account and the normal account.

A restricted account is one in which the user may log in on the system, but is given access to only a limited part of the system, say one accounting package or a database. These accounts are, generally, prohibited from program development. (Under both VMS and Unix, if you can write a program you can defeat the restrictions.) There may be several reasons for restricting a user's access to the system. First of all is security. To keep the secretary from getting to the payroll file, you
lock him into the word processing software. Another point, however, might be simplicity. The Vice President wants financial quotes; she doesn’t want to be bothered with the details of a command language interface, or have a plethora of system options available.

Whatever the reason, both VMS and Unix provide features for limiting user access to the system while still allowing some access to the system. Unix has rsh, restricted shell, which limits users to a command subset determined by the system manager (The activation of rsh is determined in /etc/passwd). This is innately powerful, since it permits the system manager to determine with fine granularity what the user can and cannot do. Unfortunately, it looks better on paper than in the flesh. Much as with VMS privileges, it turns out that many useful Unix commands, due to the inherently permissive nature of Unix, allow the user to escape from the restricted shell and run free in the system. [USS, p. 150]

VMS provides no such facility for simple alteration of the user’s selection of commands. (This can, however, be accomplished by modification of the system’s CLI tables. A subsequent change in the user’s UAF record specifies an alternate CLI, or Command Language Interpreter.) Instead, a series of login flags in the UAF is available, permitting the system manager to determine a number of things about
user logins.

The easiest way to turnkey a VMS account is to set the flags DISCTLY and NOCAPTIVE. These prevent the user from generating a system interrupt to cause exit from a program, and disable the user's ability to specify alternate login options or spawn other processes (and thus escape the restricted environment). Then the user is provided with a special login initialization file (LOGIN.COM), which starts up the turnkey application. This method seems to be something of an inversion of the Unix way of dealing with things. [VSM, p. 5-19]

The login flags also allow the system manager to specify login times, both days and hours. This is a powerful feature, and one which greatly enhances system security. By limiting users to certain hours, tight control of both system use and system security may be maintained. (System penetration attempts at any but the specified times will fail; if the specified times are work hours, during which the system manager is resident at the site, an intruder is taking considerable risk.) This is a simple and flexible system. Note that Unix can also be made to give similar performance; however, such use involves running of cron batch jobs and is considerably more cumbersome than the VMS implementation of hours-restricted access. [VSM, p. 5-20]

We see then that while Unix makes some provision for
restricting user use of the system, making possible turnkey accounts
simplifying system use, VMS supports a much stronger structure which
reinforces the severe user control found in VMS. Both systems are
thoroughly consistent with their respective philosophies; Unix is nearly
so concerned with restricting user actions as is VMS, since a high-
security environment was not part of the original design goals. There
remains, then, the question of normal accounts.

A normal user is one who is able to do anything possible on
the system, with the exception of functions relegated to system operators
and system managers. Normal accounts in Unix usually run under one
of two shells, or command interpreters: sh (Bourne shell) and csh (C
shell). Both are specified in /etc/passwd. A user may switch to a
different shell during a login session. Normal Unix users have no
restricted command set, except for commands limited to root or special
groups. Unix philosophy favors the assignation of normal accounts;
restricted shell is handy in some cases, but is not, like captive VMS
accounts, considered to be something that will be used often.

Normal VMS accounts also allow the user access to all
functions but those reserved for systems personnel. Also, as with Unix,
different CLIs are available, such as DCL (Digital Command Language)
and MCR (Monitor Console Routine, an RSX-11 compatible
interpreter). In addition, however, the set of commands available to the normal VMS user may be modified by alteration of the CLI’s command tables (what commands it recognizes), as well as the locking or deinstallation of privileged software. This is consistent with VMS philosophy and the principle of least access; great effort is made to allow fine and careful control of what the user can do, even down to his capabilities upon and during login. This, then, is how software account control operates under VMS and Unix philosophies.

Next at hand, then, is the hardware access available to a user. Hardware access refers to what physical pieces of the system the user can get at; Terminals, printers, tapes, disks, consoles, etc. Hardware access, far more than software, is a matter of grave concern. A user’s capability to use and abuse the system in software can be controlled, and at the least monitored. However, there are no such control or audit features for hardware, other than those provided by the security systems guarding the devices. A user in the midst of a computer room filled with security guards and a SWAT team could still fire a clip of 9mm shells into the CPU and disks before anyone could stop her.

Similarly, data integrity may be guaranteed by the software, but if anyone can walk up to a printer and kite off with someone’s mail or a list of authorized users on the system, there may be problems. In
short, "The security of any computer system is only as good as the physical security of the hardware." A computer cannot defend itself, unless programmed to. And even then, "You can make physical access difficult. You cannot make it impossible." [CSH, p. 57]

Access to terminals may or may not be a matter of concern. On both VMS and Unix, terminals are treated logically rather than physically. If the system manager wished, she could configure the system such that it would only allow accounts payable to be run from terminals 5, 7, 10, and 20, and to automatically start up on those terminals when the system was bootstrapped. This, however, would be foolish. It is far better to require a password, even one which the entire accounting department knows, which at least adds an additional level of security. Simply allowing access on the basis of terminal ID opens up all sorts of security holes. Other personnel may gain access to the terminal, or truly malicious intruders may tap into physical terminal lines. [CSH, p. 69] The best way to insure security is, of course, to lock the terminal up in its own room.

Printers may seem beneath notice in the academic environment, but more than one system has been compromised from unearthed printouts in trash bins. Printers should be monitored by authorized system personnel, as neither VMS nor Unix provide an
inherent method for preventing a file from being printed. (Though it is possible to deny printing privileges to both VMS and Unix users, and some scheme using SUID might be worked out in Unix.) Unix philosophy may adopt its usual "don't care" approach to the dumping of sensitive data. After all, /etc/passwd itself is traditionally world readable, on the theory that the encrypted passwords it contains are safe. However, woe to the naive Unix system manager who allows a user to print out the operating system source, and carry it off. A multimillion dollar lawsuit from AT&T is not something to be unconcerned about.

Printers, then, should be at least monitored under BOTH philosophies. (A good method is to assign one person to handle printouts, and if high security is necessary enforce user identification and the shredding of errant printouts.)

The next device of concern is the tape drive, and its associated volumes. VMS philosophy would restrict all user tape access, requiring volume mounting and dismounting by qualified systems personnel. Indeed, in a business environment user tape submissions must be prohibited; an investment analyst cannot be allowed to carry off the corporation's assets for dissemination to high-paying competitors. (And for both Unix and VMS, theft of the system distribution is a problem...)

A Unix philosophy, however, encourages user tape use. Tape
is a cheap, effective storage medium. In addition, it adds a degree of portability to the user's work on the system they might not otherwise have. Actual tape drive access may be restricted for hardware considerations. (Not many people know how to mount a tape.) However, tape submissions should be encouraged.

One note, however, is that there is no file write protection on the tape drive when it is in operation. If the system manager is doing backups, a user can step up and write right over the backup tape. This is a perilous situation, even for an open environment. With the file system corruptions found in Unix, maintaining backup integrity is imperative regardless of the management philosophy in effect. Perhaps the best method would be to run backups through a script run SUID as root; the protection on the tape drive could be raised, then lowered after the backup. Nonetheless, this world-write attitude enforces the philosophy of encouraged tape use.

Disks are devices for which access considerations are heavily dependent on what the disk is used for. Unix philosophy would certainly advocate user submission of disk packs for mounting and unmounting, if drives are available for that purpose. VMS philosophy, for the reasons mentioned for tapes, must necessarily restrict this access. However, neither philosophy should allow a user direct physical access to the disk
drives, as these will normally reside in the computer rooms (tapes may be remotely located). Access to the user drive then would also permit access to the system drive; VMS philosophy would concern itself with possible damage to the system by the device being powered down with no software dismounting. Unix must again worry about a user taking off with the licensed system... (Though here too this is a problem for VMS.) In a research environment, however, normal user access to the disk drives is not only possible, but a frequent occurrence.

Lastly is the CPU. This is one piece of equipment that no user should have access to, regardless of which system management philosophy is extant. VMS realizes that with the CPU the user can gain control of the entire system. Unix sees that the normal user has no need to manipulate the CPU or console directly. If they are loading console floppy programs or booting alternate systems, then they are no longer normal users, but special systems personnel. Even in a research situation, only those users knowledgeable enough not to damage the CPU should be given access to it.

User access concerns dealt with, I'll turn now to access to the system by systems personnel. There are, as I see it, three classes of systems person: Operators, system programmers, and system managers.

"(Operators) are the users who respond to the requests of
ordinary users, who tend to the needs of the system’s peripheral devices (mounting reels of tape and changing printer forms), and who attend to all the other day-to-day chores of system operations." [VSM, p. 6-13]

Taking this as the definition of an operator’s duties, we can determine what access they need, and perhaps shall be granted, under the two different system management philosophies.

An operator must have access to the tape drives, printers, and any other peripherals he may need in the course of his work. This may, if his duties include system startup and shutdown, extend to the CPU itself. There is, however, the question of whether he is allowed any additional access.

VMS philosophy says, flatly, no. (Recall the principle of least access.) The last thing desired is that an operator have programming capability. If security is a major factor, a captive operator account could be created in which all the system commands unrelated to operator functions were disabled. The operator would then be entirely unable to either affect system operation or gain knowledge of the system outside his field of work.

Unix philosophy, of course, would simply give the operator a normal account and a set of commensurate duties. Unix philosophy realizes that there is little point in restricting the activities of the
operator, when in all probability he has access to the console. And can, at any rate, take off with the system distribution in the form of a dump. On a research system, the system manager(s) are probably the ones doing the backups; in this case, the question of operator access is moot.

Systems programmers are a step above operators. They must, of necessity, be given a high degree of access to the software portion of the computer system. In VMS, the assigning of any privileges above the norm is, as we have seen, an ultimate compromise of the entire software system. In Unix, too, a system programmer running SUID root programs has access to the totality of the system. In addition, the system programmer is given opportunity to know far more about the system than the average user (which includes the average applications programmer). She will probably have a good working knowledge of the system internals, be they VMS or Unix. To a large extent, then, the system programmer has as much control over the software portion of the system as the system manager himself. Any security enforcement for the system programmer, then, must proceed directly from the system manager.

In Unix, however, there is evidence of a more open attitude. VMS attempts the illusion of scaled access with privileges. Unix makes no such attempt; you either are the super-user, or you aren’t. If a
programmer has super-user access, he has the run of the system. If he does not, he is an ordinary user. (Recall that my discussion in this paper is independent of the bureaucratic hierarchy.) The implicit trust of the system programmer seems to be much higher in Unix than in VMS. [CSH, p. 93]

System programmers are concerned with the software, not the hardware of the system. Under VMS philosophy they would be denied all access but that of a terminal and perhaps a printer. Even under Unix, though, unless their duties carry them into an area requiring physical tape, disk, or console access (writing CPU or console code would be such a case), there is really no reason to provide them with physical access. On the other hand, if programming activity runs the risk of a system crash, console and/or CPU access may be provided so that the system programmer can pick up the pieces. There is really no visible difference between Unix and VMS philosophy on this point of hardware access.

We come at last, then, to the access permitted to system managers. Obviously very little need be said about this. The system manager, by the virtue of his duties alone, must be provided with full and complete access to the system, regardless of which philosophy is in force. Failure to do so will impede him in performance of his job. His
account will always be the most privileged, his position one of unquestionable integrity. Obviously, such a position should not be assigned lightly; even in a research environment (perhaps, especially in a research environment...!), the system manager is responsible for all that occurs on the system. He has in his trust the entire user community. Not only must he be technically competent... he must be legally and ethically competent as well.

I conclude my discussion of system access with the issue of old accounts. An old account is, simply, one that is no longer used towards the directed end(s) of the system as a whole. Examples are the test account created for the new disk drive, or the old account that Dick and Jane used before they moved on to greener pastures at Digital Equipment Corporation. There are three things you can do with an old account: disable it, purge it, or do nothing at all. Which option is chosen depends on which system management philosophy is in force.

Disabling involves setting the access file entry for the account in question to no access. In VMS the DISUSER flag is set; in Unix, the password field can be set to "nologin." VMS philosophy would probably disable if the user was on vacation; for a permanent leave of absence, more drastic action would be taken. Unix, however, uses disabling as a standard practice; the distribution comes with a number of disabled
accounts from the Unix system creators (Kernighan, Thompson, et. al.)

Relevance of access, it seems, is not a concern under Unix philosophy. Disabling has another purpose, however, in that it is often used to restrict access to a powerful account. Under VMS, FIELD and SYSTEST are often disabled until needed. (It is through the default passwords for these accounts that intruders often gain access.)

In Unix, however, disabling of root can cause problems during a system crash or other unlikely system event. The system must be rebooted single user, and the /etc/passwd entry changed to reenable root login. However, there may be other accounts, such as daemon or ingres, which require existence for UID purposes but which should not be available for general access; disabling in these cases is appropriate.

The next alternative, of course, is purgation. If an employee was on vacation, it would make no sense to remove his account entry and delete his files, regardless of the philosophy. However, his transfer to another employer would constitute sufficient cause for housecleaning under VMS. Unix, as we have noted, apparently does not believe in purgation. (Though an individual system manager might purge as a matter of disk space, or simply to "unclutter" the distribution /etc/passwd.)

Purgation can range from removal of the account entry to
deletion of all files, and even to deletion of all that user's files on backup tapes! Such an extreme program might be necessary in the event of a trojan horse, logic bomb, or virus code in the user's account somewhere (whether put there by him or not).

VMS philosophy clearly favors account deletion; if there is no reason for a user to have access to the system, that user's account should be deleted.

Lastly, then, is the "do nothing" option. This, for VMS, is anathema, as it provides an open door for system compromise. Under Unix it is problematic. Leaving accounts belonging to the system manager's friends active, even after they have left the installation, is tempting; in the end, the level of security necessary for the Unix site will determine this. For if the Unix system is its open, friendly, compromisable self, the possibility of outside intrusion through so easy a medium is not to be born.

Thus we see that access is the first concern of both philosophies. I will now show the system manager's place in the system, and reveal the origin of the two philosophies...
"If you aren’t in complete control of a situation anything you do can make it worse."
--Howard Leary

Chapter IV
Gods and Mortals

One of a system manager’s highest responsibilities is the security and control of the computer system’s resources. His domain extends from filesystems to batch job queues, from access control lists to network protocols. In the context of the computer system his word is law; his system access is most privileged; for on the system, the system manager is god.

The users are the mortals. Their access exists only through the power of the system manager; their files are backed up against recovery through her policies. She installs the users’ new software, she protects them and guides them through a maze of technology which all but the most adventuresome systems programmers could never hope to traverse. The users are dependent on the system manager in to an extent that few users, or system managers, realize.

We now come to the crux of the issue of system management philosophy. In the foregoing chapters I have laid the groundwork for the