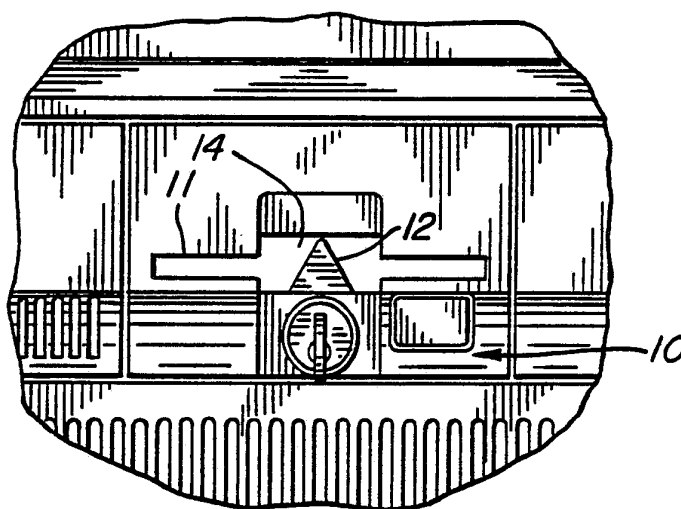




INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁵ : G06F 13/14	A1	(11) International Publication Number: WO 90/05340 (43) International Publication Date: 17 May 1990 (17.05.90)
(21) International Application Number: PCT/US89/04913 (22) International Filing Date: 3 November 1989 (03.11.89) (30) Priority data: 267,265 4 November 1988 (04.11.88) US (71) Applicant: LAMA SYSTEMS INC. [US/US]; 2115 Northland Drive, Austin, TX 78756 (US). (72) Inventors: LAKOSKI, Robert, P. ; 5417 Shoalwood Drive, Austin, TX 78756 (US). VAN CLEVE, Roy, Jr. ; 3935 Lago Vista Drive, Austin, TX 78734 (US). (74) Agent: CAHN, Maurice, U.; Leydig, Voit & Mayer, Suite 300, 700 Thirteenth Street, N.W., Washington, DC 20005 (US).		(81) Designated States: AT (European patent), AU, BE (European patent), BF (OAPI patent), BJ (OAPI patent), BR, CF (OAPI patent), CG (OAPI patent), CH (European patent), CM (OAPI patent), DE (European patent), DK, FI, FR (European patent), GA (OAPI patent), GB (European patent), IT (European patent), JP, KR, LU (European patent), ML (OAPI patent), MR (OAPI patent), NL (European patent), NO, SE (European patent), SN (OAPI patent), SU, TD (OAPI patent), TG (OAPI patent). Published <i>With international search report.</i>

(54) Title: PERSONAL COMPUTER ACCESS CONTROL SYSTEM

**(57) Abstract**

A personal computer security access system incorporating default, boot up, bi-level security access control software (14) which prevents access to the computer without input of the access code and password and a physical disk drive blocking device (12) which can be locked to prevent removal of the software (14) from the computer.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	ES	Spain	MG	Madagascar
AU	Australia	FI	Finland	ML	Mali
BB	Barbados	FR	France	MR	Mauritania
BE	Belgium	GA	Gabon	MW	Malawi
BF	Burkina Fasso	GB	United Kingdom	NL	Netherlands
BG	Bulgaria	HU	Hungary	NO	Norway
BJ	Benin	IT	Italy	RO	Romania
BR	Brazil	JP	Japan	SD	Sudan
CA	Canada	KP	Democratic People's Republic of Korea	SE	Sweden
CF	Central African Republic	KR	Republic of Korea	SN	Senegal
CG	Congo	LI	Liechtenstein	SU	Soviet Union
CH	Switzerland	LK	Sri Lanka	TD	Chad
CM	Cameroon	LU	Luxembourg	TG	Togo
DE	Germany, Federal Republic of	MC	Monaco	US	United States of America
DK	Denmark				

PERSONAL COMPUTER ACCESS CONTROL SYSTEM
TECHNICAL FIELD

This invention relates to computer security and, more particularly, to an access control program coupled with a physical disk drive blocking device prohibiting unauthorized alteration, copying or destruction of programs and data encoded in a computer memory or on computer disks.

BACKGROUND OF THE INVENTION

The presence and use of personal computers in all phases of business and the home is now well established. Also, now well established are a myriad of problems ranging from vandalism including the introduction of computer viruses into computer operating systems and programs to theft or alteration of sensitive business information and data. While actual theft of the computer hardware has occasioned the introduction of numerous devices for locking down the hardware to prevent theft, the great damage occurs when sensitive data, both business and personal, is compromised. The introduction of physical security adjuncts such as internal locking systems and even external portable disk drive locking devices such as that embodied in United States Letters Patent 4,685,312, which is owned by Applicant.

Since the introduction of the personal computer and the full-height disk drive, personal computer design has advanced to where a general computer includes an internal hard disk drive and an external boot diskette drive. Thus, when the computer is turned on, the computer automatically defaults to the external boot diskette drive which,

if empty, causes the computer to reroute its initialization to the internal hard disk. It is this programmed initialization that permits access control to the computer by employing software which will deny access to the programs and information contained on that hard disk drive by an authorized person. In contrast, an access control for main frame and multi-user computer systems are complex and require the likes of an audit trail, multi-permission levels, file and directory restrictions and even encryption. Such security is particularly of great concern to persons dealing with highly-sensitive business information or, even more importantly, to national security information. However, the numerous above-identified safeguards are not particularly useful or productive when employed with a single user personal computer. First, access control, alone, is sufficient to maintain a personal computer personal. Secondly, the introduction of one or more of the numerous safeguards identified above may consume significant system resources thereby denying utilization of the computer's full potential.

Referring to those safeguards, as a practical matter, it leaves the computer operator somewhat at a disadvantage when needing to step away from the computer momentarily for whatever reason. Log-off and log-on procedures will waste considerable time and energy, so much so as to deter the user from employing the safeguards thereby leaving the program and data exposed while the user is absent from the workstation.

The following definitions are set forth for a more complete understanding of the invention:

DOS - disk operating system - permits initialization of the computer for

use with compatible software (MS DOS
PC DOS)

Driver (OR "DEVICE DRIVER" - a
subprogram, often used to control
hardware peripherals, which acts as
an extension of the operating
system

LAMASYS.SYS - the security device
driver, loaded and initialized by
DOS at boot time

LAMA.COM - a DOS command level
program which may be used to invoke
the security driver after boot time
(like hotkey)

LAMAINST.EXF - the DOS program which
installs the LAMASYS.SYS driver so
that it will be initialized at boot
time

BIOS keyboard service interrupts - a
set of functions for reading the
keyboard which are standardized as
part of the IBM standard

Drive A - the floppy disk drive or
other removable medium device
which the system treats as the
first boot device

Hard Disk - a high capacity, fixed
(non-removable) memory storage
device

Hotkey - a user selected combination
of keys used to invoke a program

K equals thousand of bytes - exactly
1024 bytes

RAM - random access memory -
volatile, loses contents when power
is turned off

CGA - color graphics adapter - IBM
standard

EGA - enhance graphics adapter - IBM
standard

<ENTER> - the enter key

<Ctrl> - the control key

<Alt> - the alternate key

 - the delete key

ESCAPE - the escape key

A: - the A drive

B: - the B drive

C: - the hard disk drive

Config.Sys - a special file

containing a list of drivers and
other initialization commands which
DOS reads at boot time

Side Kick - a commercially available
"memory resident" utility program
by Borland, Intl.

Periscope - a commercially available
debugging system used by
programmers .

Mirror - a background communications
package

ATTR.COM - a commercially available
program, distributed by PC
Magazine, to allow a user to hide
and otherwise protect sensitive
files and directories

Steal - to redirect a system function
call (such as keyboard) to one's
own program to allow special
handling before passing it on for
normal system processing

Hook - to redirect a system function
call (such as keyboard) to one's

own program to allow special
handling before passing it on for
normal system processing

Access control means a system which prevents the use of a personal computer by unauthorized personnel (persons who should not be accessing the personal computer). For mainframe and other multi-user computer systems, full security may require audit trails, multiple permission levels, file and directory restrictions, and even encryption. Such safeguards are generally inapplicable to the vast majority of single-user personal computers and even if operable consume excessive system resources. For the single-user system, access control is intended to keep a personal computer personal.

SUMMARY OF THE INVENTION

It is therefore an object of this invention to provide an access control system overcoming the shortcomings of the prior art.

It is another object of this invention to provide an access control system which prevents unauthorized persons from accessing programs and information stored on the hard disks of a personal computer or, further, to prevent access through a personal computer to a mainframe or multi-user network.

It is still another object of this invention to permit full use of the computer, computer programs and data stored on the computer hard disk or within a system by an authorized user but to render use of the computer virtually impossible to a user without authorization.

Still another object of this invention is to provide an access control system which requires

only a very small portion of computer memory for its operation.

Another object of this invention is to provide instant security screen activation upon keystroke command by the authorized user, and to return to the prior state, after the password is given.

Another object of this invention is to provide access control software coupled with a physical security boot drive slot blocking device to prevent introduction or removal of diskettes or cartridges from the drive slot when the physical blocking device is secured.

These and other objects are satisfied by a combination of a computer access control system for use on a personal computer having a diskette boot drive device to deter unauthorized access to the computer data and programs, comprising physical means for blocking removal or insertion of diskette (operator removable medium) containing the computer boot drive device, and computer access control program means for enabling and disabling a personal computer upon input demonstrating authority to access the computer.

Still other objects are satisfied by a method for controlling access to a personal computer with a diskette drive, comprising the following steps: inserting a computer program containing device into the diskette data drive, blocking the drive with a lockable blocking device, installing a computer program including a device driver that is first addressed by the computer during boot up and which requires entry of a proper access code and password prior to moving to another device driver, and entering a selected access code and password.

This invention unites two previously independent access deterrent means, software and hardware, to provide access control that can be employed with confidence to prevent unauthorized access or use of a personal computer. In essence, the software prevents unauthorized entry into the system both at time of initialization and during temporary breaks in the operator's work routine. The combination of the hardware and software creates an intruder barrier prohibiting access, without damage, to the computer and with sounding of audible alarms if attempted surreptitiously.

In brief, the software described is for an embodiment directed for use with a MS-DOS or PC-DOS (2.0 or higher) based computer. When the computer is powered up or reset, it searches for a file called CONFIG.SYS which contains a list of "device drivers". Each of these in turn is loaded into memory and then given temporary control of the computer to allow initialization of the device it controls.

The access control software for purposes of convenience is referred to as LAMASYS. LAMASYS.SYS is a DOS device driver file which is activated at boot time as the first driver invoked by the CONFIG.SYS initialization process. It acts as an extension of the operating system, much like the console driver system included with PC-DOS.

As the system initializes, either by having power applied or by being reset, the LAMASYS.SYS driver is invoked prior to any other drivers. If the correct password and access code are not entered, the LAMASYS.SYS driver will not return control to the operating system, preventing initialization and any use of the computer. In addition, the driver continues to monitor the BIOS

keyboard service interrupts, even after the computer begins normal operations. Any time a key is pressed, the value of the returned keystroke is compared to a user-defined key (hotkey). If they match, the LAMASYS.SYS driver is invoked again.

When security is invoked, LAMASYS.SYS temporarily "steals" whichever DOS and BIOS interrupts it requires in order to maintain system control. When authorization is verified and the LAMASYS.SYS returns control, all interrupts are restored to their previous state and the original screen is restored.

The reliability of the access control governed by the software depends upon maintaining control of the disk drive which the computer uses for initialization. Generally, since personal computers attempt to initialize first from a diskette if one can be found in drive A:, it is essential to block access to diskette Drive A: during the initialization process.

Blocking access is achieved by use of ancillary locking hardware. More particularly, a disk drive lock is employed to block the diskette slot in such a way that a diskette may not be inserted into an empty drive or that a diskette locked into a drive will continue to function but may not be removed without causing visible damage to the computer or the hardware.

The software uses only 7K of memory, 4K of which are used to save the current screen data so that it can be restored later. While this is sufficient to store a normal 80x25 text screen, it does not provide enough memory to save a color graphics screen (16K bytes) or a 110x75 text screen, let alone some EGA graphics screens (256K). In balancing the competing principals, minimizing

resident memory requirements versus complete screen text storage, the former was selected with the intent of saving as much screen area as can be saved in 4K and foregoing tying up memory.

The disk drive lock and software combination, therefore, provides password protected access control at boot-up time. In addition, the computer may be disabled at any time by the press of a user-selected "hotkey". Only entry of the correct password will return the computer to normal operation, and excessive erroneous entries will trip the audio alarm, at which point only the original "hotkey" can turn it off. In order to protect the user's work-in-progress, the computer cannot be re-booted from the keyboard when in the disabled (secured state). The secured state may also be invoked from a batch file or from the DOS command line by use of the appropriate command file (LAMA.COM).

The invention herein provides a computer access control system which does not in any way restrict or monitor the activities of authorized users once the appropriate log-on procedure has been followed. Furthermore, the system neither restricts data files or directories nor does it restrict access to programs or count keystrokes. Indeed, it is a primary objective of the software in this invention to be as transparent as possible for the authorized user, and impossible for the unauthorized user. It is for this very reason that the software only requires 7K of resident memory, including 4K for screen storage..

BRIEF DESCRIPTION OF THE DRAWING

Figure 1 is a schematic representation of a computer secured by the invention.

DETAILED DESCRIPTION OF THE INVENTION

The invention includes hardware component 12 and software component 14 for use together on a multiple drive personal computer 10. Software component 14 is an access control program. The software for personal computers is stored on an insertable/removable medium such as floppy diskette 14 or alternatively, an optical disk or a RAM cartridge. The computer includes disk drive unit 11 configured to receive the diskette (disk or cartridge, if applicable). It is this disk drive unit to which the hardware component is directed.

The hardware component may take the form of a portable disk drive locking unit of the type described in United States Patent 4,685,312 Lakoski et al, incorporated herein by reference. Alternately, the hardware can be of the form of a computer bezel mounted, lock and lock bar assembly described in Applicant's United States Patent Application S.N. 227,129 filed August 2, 1988 incorporated by reference. The primary requirement of any hardware employed in this invention is to secure and otherwise keep the access control program locked into the boot drive thereby preventing unauthorized removal of the program without causing visible damage to the locking unit or computer.

Both of the identified locking units are operated with a key. It is not intended to thus limit the invention to a combination lock or even a more sophisticated electronic locking means. As a

practical matter, especially in a multiunit supervised environment, the supervisor may possess a master key or duplicate key to unlock the subordinate's units. By this means, the supervisor's disk can be substituted for the subordinate's, thus denying the subordinate access to the computer. This concept may prove valuable in the case of a recalcitrant employee.

The second component of the system, in one embodiment, is now described in detail as to its installation, operation and program content. First, addressing the system requirements, the program is configured and designed to operate on an IBM^R personal computer or equivalent, using a conventional Disk Operating System (DOS) such as IBM PC DOS or MS DOS version 2.0, etc. The program supports both monochrome and color monitors and all conventional CGA and EGA graphic modes. Furthermore, the program is capable of supporting non-standard screen dimensions as, for example, 132x44 characters, 110x75 or 80x66 modes of super-EGA boards. Referring to memory requirements, the program for installation requires only 7K bytes of system RAM memory and only a 7.5K byte file on the boot diskette. If the DOS program is in a graphic mode when security is invoked, the security screen (described below) will be compressed. This is because only 4K of memory is used for saving the screen, so only 4K of the screen can be written over. In the high resolution EGA modes this represents only two lines in the upper-left corner of the screen. Finally, as an elementary practical note, the program is only effective if the default boot diskette drive is protected from intruder usage.

To install the access control program included on the distribution diskette, the diskette is inserted into the computer default drive (A:). The user then types the installation execution program command ("LAMAINST"), and then follows the menus. There are two possible levels for installation, basic and advanced.

The most common method of installation is to copy the program onto the associated (a PC XT, AT, etc.) permanently mounted hard disk where the hard disk is most commonly used as the boot drive. Hence, boot up by the hard disk is the default for the LAMAINST.EXE installation program. Other installation configurations are discussed later, but most users will want to use the quick procedure now described.

After DOS has been loaded, by placing the computer access control program containing diskette in drive A: and typing A: <ENTER>, drive A: becomes the default drive. Next, LAMAINST <ENTER> is typed and the program's Main Menu then appears. Pressing "1" at the Main Menu selects installation onto the hard drive. The user then selects a personal access code of at least six characters which is inputted upon request on the screen. The access code is the first entry of the bi-level security access system provided by this invention.

The next screen requests entry of a personal password, the second entry. This password must not be forgotten or the system can be locked out. The user then has the option of selecting a "hotkey" (a key combination which immediately moves the system into the security mode as detailed below). The "hotkey" initially is assigned the key combination "Ctrl 2". Once selection of the "hotkey" is complete, the user presses "6" to return to the

Main Menu, then "5" to return to DOS. At this point, the user must reboot the computer with the <Ctrl><Alt> keys for the invention's computer access control program to take effect. The computer will now stop in the middle of the boot process and require the user to properly input the selected access code and password in order to continue. Once properly entered, the computer will continue booting in a conventional manner. Once the "hotkey" is entered, the same security screen appearing when booting, is viewed. Once the "hotkey" use is verified, the user is assured that the system has been successfully installed, so the program diskette is removed from drive A: and the diskette drive is locked up.

The foregoing description of the basic installation technique is now supplemented with alternate and more advanced installation instructions. As already noted, the default installation method used by the execute program relies on the computer using the hard disk as the boot disk. The intended "target" for the purpose of this program is identified in the first item in the main menu. This is the simplest of cases, since the same disk is both the "target" and the current "boot" disk. Other computer configurations, such as two floppy boot diskettes will require reconfiguration of INSTALL command line to direct the boot function to the appropriate disk. For successful installation of the control access program onto a floppy boot diskette, the user must specify the target diskette drive. For example, the user would type "LAMAINST B: <Enter>" if the disk is in drive B:.

Once specified, LAMAINST will look for a hard disk drive which is the current "boot" disk in

order to copy the current CONFIG.SYS file and to disable booting from that drive. The end result is that the program will boot only from the specified floppy diskette and will complete booting as though it had booted from the original hard disk (if any). This feature renders the computer access control program installation as transparent as possible. If a hard disk drive is available and specified, the installation program will simply use the specified drive instead of conducting a search for the drive containing the driver files. Unless a special configuration requiring manual override is involved, such specification is not necessary. In any case, the result is to copy the security driver file to the "target" disk and to modify the CONFIG.SYS file.

While almost never necessary, since the current "boot" disk is automatically sensed, the boot disk may be specified by using the following command:

LAINST B: BOOT=C:

This example would specify that drive B: is the target disk and that drive C: is the current "boot disk". Hence, drive C: is disabled from booting.

Selection of a specified drive copies the program on to the indicated drive and modifies the system files as needed. The user will be required to enter a new access code and password at this time. If the software is found to be already installed, a message will appear so indicating and the screen will return to the main menu.

This text now turns to the various menu options involved in installation. One option which the user will find desirable is the ability to change the access code and password. This allows the access code and password to be changed for systems which have previously been installed, but

only after the user enters the current password to demonstrate authority to access the system. If the software is not currently installed, an error message will appear on the screen. If the software has been installed on a floppy diskette rather than an internal hard disk the hardware/drive lock must be removed, the diskette pulled from the slot, the write-protect tab removed from the diskette and the disk reinserted before any changes can take effect. Should it be desired to remove the access control software from the computer and restore the system to its pre-installation state, the disk drive must be unlocked and the write-protect tab removed from the diskette before erasing the software.

Moving to the operational considerations during installation the user invokes the security option sub-menu. This sub-menu appears only after the current password has been entered to demonstrate access authority. The sub-menu options allow the user to customize important operational characteristics.

The first option is the selection of a "hotkey". When the hotkey option is chosen the computer generates a sound and message. In one example, a screen would read:

"Now press the key combination which will activate the security system. A combination of <Alt> or <Ctrl> with some other key is usually best. The bell will sound again when your new security HOTKEY is accepted. If the security screen appears instead, your HOTKEY is already set. You may press <Escape> if you wish to skip this procedure."

As a practical matter, the audible bell sound is

employed to inform the user that a particular key combination has been accepted. Not all keys work as "hotkeys" because some combinations cannot be sensed at this level of keyboard service. For example, ESCAPE cannot be used because it is used to escape from the menus of the access control program.

The user's selection of a hotkey combination should, in part, be governed by possible conflict with functional or essential command keys of the actual programs used. For this reason the use of the <Ctrl> and <Alt> keys and coupled with function or punctuation keys may prove the best combinations. One further aspect of the hotkey is now discussed. The security system will respond to the "hotkey" only when the active program actually requests a keystroke. This feature is to prevent the computer from being interrupted in the middle of critical operations, such as writing to the disk. There is a disadvantage; the system may not respond while involved in some tedious operation such as a spreadsheet recalculation. Also, in operation, the DOS program which can never see the "hotkey" since it is read by the security program may appear to be halted since it is still waiting for a keystroke which has never appeared. Pressing any key will allow the system to continue.

The presence of a hotkey is not required. The program allows selection of a zero option which only invokes security at boot up. This program configuration, therefore, does not use any memory once the boot up security is completed. A command is also provided for the user who may wish to restore the hotkey when the A> appears.

Screen color combination selections provide a highly visible indicator of the presource of a security screen. The screen color selection

option presents the user with an 8x16 grid containing all 128 available color combinations for the security screen. The default color, white on blue, will blink on the screen and the cursor will be located in its block on the grid. The cursor control keys are employed to move the cursor to a new color combination and press <Enter> to accept it. As usual, <Escape> may be used to skip this process. For monochrome monitors, the only useful option here might be to select the black-on-black option for screen blanking purposes.

The next option permits the user to customize the security screen title. The title line appears above the "AUTHORIZED PERSONNEL ONLY" banner. Up to 60 characters may be used for the title line. For convenience, due to internal programming, the resulting line will be properly centered on the screen. Since some CGA modes support only 40 columns of text, it is prudent to limit the title line to 40 characters or less when a color monitor is in use. One further note on this option relating to the EGA and VGA high resolution graphics modes, the title never appears. This is because so little of the graphics screen can be preserved in the 4K of resident memory used for screen storage that there is simply not enough space to display the title.

Probably the most important option available to the user is the selection of the security level from a security level selection menu. The choices are: 1) minimum intrusiveness which may not suppress competing resident programs "SideKick"; 2) controls and overrides such programs and also halts spooling; and finally, 3) maximum security, but halts all background activity.

Level 1, the system default level, is the least intrusive, but least secure level. It is usually adequate to maintain control unless a resident debugging system (such as "Periscope") or other contentious resident programs, such as "SideKick", are present. At level 1, only the BIOS print-screen and the keyboard vectors are "stolen" when security is invoked. Hence, background communications and print spooling are allowed to continue. However, "SideKick" and like programs will try to "steal" the print screen and keyboard vectors back. Thus, a <Ctrl><Alt> reboot at the security screen can be effected when it should be prevented. A reasonable conclusion, therefore, in such a case, a higher security level is required.

The second level "steals" the system timer interrupts when the security screen is invoked. "SideKick" type programs cannot take over but can temporarily disable timer-drive activities such as a print spooler. Level 2 has been reported to be the best security level to use when running both "SideKick" and a background communications package such as "Mirror".

The maximum security level, level 3, will disable even a hardware based debugging program such as "Periscope". When this security level is invoked, all interrupts from 0 .. 1F are restored to their values at CONFIG.SYS entry time, so that no subsequently loaded drivers can be used to "hook" or disable the security system. Caution must be exercised in use, however, since security level 3 has been known to interfere with networking and other background communications, in addition to halting spooling, etc. While level 3 may be extreme, it is extremely reliable.

The final option presented is very practical, especially for the less adept. This option selection of the alarm level, will request the user to input a number between 1 and 9. The inputted number will correspond to the number of incorrect password entry attempts allowed before the alarm is set off. The default value here is three, which gives the authorized user a little latitude for error while providing the intruder with little room to experiment.

The final line in option menu inquires if the user wishes to return to the operating system. Once selected, the currently resident LAMASYS.SYS driver file is updated to reflect any changes to the options above. The computer and screen then returns to the original menu which allows the user to exit the installation program and return to the DOS command line. The user is cautioned since it is only after the computer is rebooted and the newly configured drive is reloaded that the inputted option changes will be fixed.

Given the above installation methods and options, the reader's attention is now directed to the relatively simple operation of the computer access control program. When the security screen is activated, a characteristic trill emanates from the PC audio system. The screen then goes blank and the text, including the user controlled menu message described above, and the request for the user access code, will appear.

The access code must be entered exactly, including capitalization. As is the case with most programs, correction of erroneous input can be effected though the backspace key until the <Enter> key is pressed to complete the entry. Whatever the user types is visible at this point so that the

state of the <Caps Lock> key can be verified, as well as the overall operation of the keyboard itself. Any key strokes beyond the twenty characters allowed will simply be ignored. All entries, right or wrong, are accepted. The screen moves on to request the password.

Password entry is identical to access code entry, except that all typing is invisible. This is to prevent any casual observers from reading the password as it is being typed. If the access code and password are both correct, the screen will be restored to its original state and the boot process will continue. If either the Password or the Access Code is incorrect, the audio system will emit a less-than-encouraging sound. If the allowed number of incorrect entries has not been exceeded, the process starts over again at access code entry. After the programmed number of allowed attempts is surpassed a continuous alarm is sounded which can be terminated only by entry of the "hotkey" or by turning the machine off. If the hotkey is used the security screen is restored. Of course, if vandalism is intended the intruder can pry the physical locking device off of the computer and remove the software diskette or substitute his own diskette. Without further elaboration, such activity is clearly detectable.

The removal of the access control program may be desired or even required at certain times. Since the configuration consists of the program system file in the root directory of the boot disk, the system may be effectively bypassed by removing either the system file or the first line of the CONFIG.SYS program. As a result of the system file normally being hidden, a utility file, such as the commercially available program ATTR.COM, is required

to erase or rename the system file. Hence, it is often easier to erase the first line of the CONFIG.SYS file.

In the event a floppy diskette installation configuration is present, the hard disk booting condition can be restored by renaming the command file (LCOMAND.COM) to its original name, COMMAND.COM. Also the original CONFIG.SYS FILE being hidden and renamed to CONFIG.SAV, it must be released and renamed to restore the computer to its pre-installation state. This requires a file utility program.

Finally, to bypass the floppy diskette boot up altogether, the hardware lock is unlocked, the disk (cartridge) removed and the system rebooted with a proper system disk. Of course, as noted above, to remove the physical hardware lock without damaging the computer requires the proper key.

A programmer may be interested that manipulation of the DOS driver program allows for the invocation of the security screen from any DOS program. The program "LAMA.COM" is included to allow security to be invoked from the DOS command line or from a batch file. It produces exactly the same effect as the "hotkey", a secured state. One use for this program might be to call it at the end of a long print run, when the user will be away from the computer. Another use is as an Application in the Microsoft Windows environment, since resident utilities tend not to work there.

The following assembly language code demonstrates how the security screen can be called from any program by using the device driver mechanisms built into DOS. In effect, the access control program acts as an extension of the operating system. The DOS open command with the

name, 'LAMA1988' is entered. Functions 3, 5, 7, or 10 will activate the security screen. After control is returned, the device is closed in the normal manner. If the open command fails, the access program is not present. As an example, the DOS command line utility, LAMA.COM, included on Applicant's commercially available diskette, uses exactly this method.

```

lamaname db 'LAMA1988', 0
dummybuf db
0
;
Entry    proc    near
    mov     dx, offset lamaname    ; Load the name of
                                   the driver
    mov     ah, 3Eh                ; set to DOS open function
    xor     al, al                ;...in the read mode.
    int     21H                   ;call DOS.Is the driver
                                   present?
    mov     bx, ax                 ;assume so, set bx to new
                                   handle
    jc      LAMAout               ;if fail, not available.
                                   Bye.
    mov     ah, 44h               ;call the DOS IOCTL
                                   function
    mov     al, 7                 ;LAMALOCK+ = functions
                                   3,5,7,10
    mov     dx, offset dummybuf    ;DS:DX = dummy buffer
    xor     cx, cx                ;read no bytes anyway
    int     21H                   ;This will call the
                                   Security Screen
    mov     ah, 3Eh               ;and not return without the
                                   passwords.
    int     21H                   ;close the LAMALOCK+handle in
                                   BX

```



```
LA
MA
out:
    ret                ; return control to the caller
Entry
endp
```

The computer access control program described above can be employed in many environments and adapted to any of the broad range of commercially available personal computers having boot diskette or RAM cartridge input slots. Not to belabor the obvious, but the true utility of the program is achieved only when coupled with a physical access slot blocking lockdown device to prevent removal of the software once loaded.

A copy of the complete system program and installation program are appended to this application and incorporated by reference herein but are not reproduced for purposes of convenience.

Given the foregoing modifications and variations of the invention should now be evident to the person of ordinary skill in the art. Such modifications and variations are intended to fall within the spirit and scope of this invention as defined by the following claims.

```
/* Copyright (c) 1988 Lama Systems Inc. */
```

```
INSTALL
```

```
#include "stdio.h"
#include "dos.h"
#include "stat.h"
#include "fcntl.h"
#include "string.h"

#define FALSE 0
#define TRUE 1
#define PASSWORD 0 /* begin option definition list */
#define HOTKEY 1
#define ATTRIBUTE 2
#define TITLE 3
#define TITLELEN 60
#define SECURITY 4
#define ALARMLEVEL 5

#define BACKSPACE 8
#define ESCAPEKEY 0x1B
#define termch '\r'
#define tabincr 4

char firstfixed, topdisk;
char vidattr = 0x70;
char InMain = TRUE; /* Flag to tell if at top level menu */
union REGS vidregs;
int HideKeys = FALSE;
int lamahandle;
unsigned lamasize;
char *CommStr = "A:\\\\COMMAND.COM";
char *LCommStr = "A:\\\\LCOMAND.COM";
char *ConfStr = "A:\\\\CONFIG.SYS";
char *ConfSav = "A:\\\\CONFIG.SAV";
char *AutoStr = "A:\\\\AUTOEXEC.BAT";
char *LamaStr = "A:\\\\LAMASYS.SYS";
char *DeviceStr = "DEVICE=LAMASYS.SYS\n";
char *autoset1 = ":\nSET COMSPEC=";
char *autoset2 = ":\nLCOMAND.COM\nautoexec\n\n0x1a";
char targetdisk = 'N';
char hardboot = 'N'; /* Initialize to assume no current hard drive boot */
char clineboot = 'N'; /* Assume no hard drive forced by command line option */
#define BUFLen 10000
char lamabuffer[BUFLen];
char *RPLstring = "432 ROL";
char *DevPattern = "device";

/*-----*/
void VMovCur( row, col )
int row, col;
{
    vidregs.h.ah = 2; /* Write character and attribute */
    vidregs.h.dl = col;
    vidregs.h.dh = row; /* initialize the video interface */
    int86( 0x10, &vidregs, &vidregs );
}
/*-----*/
void VPutChr( outchar )
```

SUBSTITUTE SHEET

25

```

    int outchar;
{
    vidregs.h.ah = 9; /* Write character and attribute */
    vidregs.h.al = outchar;
    vidregs.h.bl = vidattr; /* initialize the video interface */
    vidregs.x.cx = 1;
    int86( 0x10, &vidregs, &vidregs );
}
/*-----*/
void VRepChr( outchar, repcount )
    int outchar, repcount;
{
    vidregs.h.ah = 9; /* Write character and attribute */
    vidregs.h.al = outchar;
    vidregs.h.bl = vidattr; /* initialize the video interface */
    vidregs.x.cx = repcount;
    int86( 0x10, &vidregs, &vidregs );
    vidregs.h.ah = 3; /* read cursor position */
    int86( 0x10, &vidregs, &vidregs );
    vidregs.h.ah = 2; /* set cursor position */
    vidregs.h.dl += repcount;
    int86( 0x10, &vidregs, &vidregs );
}
/*-----*/
void VOutChr( outchar )
    int outchar;
{
    VPutChr( outchar );
    vidregs.h.ah = 3; /* read cursor position */
    int86( 0x10, &vidregs, &vidregs );
    vidregs.h.ah = 2; /* set cursor position */
    vidregs.h.dl ++;
    int86( 0x10, &vidregs, &vidregs );
}
/*-----*/
void VOutTTY( outchar )
    int outchar;
{
    vidregs.h.ah = 0x0E; /* write TTY */
    vidregs.h.al = outchar;
    vidregs.h.bl = 7;
    vidregs.h.bh = 0;
    int86( 0x10, &vidregs, &vidregs );
}
/*-----*/
void VOutStr( outstr )
    char *outstr;
{
    while ( *outstr != '\0' ) VOutChr( *outstr + + );
}
/*-----*/
#define HLINE          '\315'
#define VLINE          '\272'
#define ULCORNER       '\311'
#define URCORNER       '\273'
#define LLCORNER       '\310'
#define LRCORNER       '\274'

#define SHLINE         '\304'
#define SVLINE         '\263'

```

SUBSTITUTE SHEET

```

#define SULCORNER '\332'
#define SURCORNER '\277'
#define SLLCORNER '\300'
#define SLRCORNER '\331'

int singline = TRUE;

MakeBox ( orgr, orgc, rows, cols )
int orgr, orgc, rows, cols;
{
    int i;
    char border, saveattr;

    saveattr = vidattr;
    vidattr = 0x70;
    VMovCur( orgr, orgc );
    VOutChr( singline? SULCORNER: ULCORNER );
    VRepChr( singline? SHLINE: HLINE, cols-2 );
    VPutChr( singline? SURCORNER: URCORNER );
    border = singline? SVLINE: VLINE;
    for ( i = 2; i rows; i++ )
    {
        VMovCur( orgr+i-1, orgc );
        VOutChr( border );
        VRepChr( ' ', cols-2 );
        VOutChr( border );
        VPutChr( 219 );
    }
    VMovCur( orgr+rows-1, orgc );
    VOutChr( singline? SLLCORNER: LLCORNER );
    VRepChr( singline? SHLINE: HLINE, cols-2 );
    VOutChr( singline? SLRCORNER: LRCORNER );
    VPutChr( 219 );
    VMovCur( orgr+rows, orgc+1 );
    VRepChr( 219, cols );
    vidattr = saveattr;
}
/*-----*/
clrscreen( fillchar )
int fillchar;
{
    int i;

    for ( i=0; i 25; i++ )
    {
        VMovCur( i, 0 );
        VRepChr( fillchar, 80 );
    }
}
/*-----*/

char *LamaTitle1 = "AUTHORIZED PERSONNEL ONLY";
char *LamaTitle2 = "Installation Program";
char *LamaMAIN = "Main Menu";
char *LamaItem1 = "1. Install the APO software onto Drive ";
char *LamaItem2 = "2. Change the Access Code and Password";
char *LamaItem3 = "3. Customize the Security Options";
char *LamaItem4 = "4. Uninstall the APO software from Drive ";
char *LamaItem5 = "5. Quit and return to DOS";
char *LamaChoose = "Choose a number";

```

SUBSTITUTE SHEET

```

char *LamaCopy = "\\332\\304\\(c) Copyright Lama Systems Inc. 1988\\304\\277";

int MUSTDRAW = TRUE;

int InitMenu()
{
    int i;

    if ( MUSTDRAW )
    {
        clrscr( 177 );
        MakeBox ( 3, 10, 19, 60 );
        VMovCur( 5, 28 );
        VOutStr( LamaTitle1 );
        VMovCur( 6, 30 );
        VOutStr( LamaTitle2 );
        VMovCur( 8, 35 );
        VOutStr( LamaMAIN );
        VMovCur( 10, 19 );
        VOutStr( Lamaltem1 );
        VOutChr( targetdisk );
        VOutChr( ':' );
        VMovCur( 12, 19 );
        VOutStr( Lamaltem2 );
        VMovCur( 14, 19 );
        VOutStr( Lamaltem3 );
        VMovCur( 16, 19 );
        VOutStr( Lamaltem4 );
        VOutChr( targetdisk );
        VOutChr( ':' );
        VMovCur( 18, 19 );
        VOutStr( Lamaltem5 );
        VMovCur( 24, 19 );
        VOutStr( LamaCopy );
        VMovCur( 20, 32 );
        VOutStr( LamaChoose );
        MUSTDRAW = FALSE;
    }
    return( getch() );
}
/* ===== */
VPutSqr( row, col, attr )
    int row, col, attr;
{
    int vrow, vcol;

    vidattr = ((row < 4) + col) | attr;
    vrow = row + 12;
    vcol = col*4 + 9;
    VMovCur( vrow, vcol );
    VOutChr( ' ' );
    VOutChr( row + '0' );
    VOutChr( col + (col > 10? '0': 'A'-10) );
    VOutChr( ' ' );
    VMovCur( vrow, vcol+2 );
}
/* ===== */
int
GetAttr( orgattr )
    char orgattr;

```

SUBSTITUTE SHEET

28

```

{
    int row, col, saveattr, inchar;

    saveattr = vidattr;
    MakeBox( 11, 8, 12, 66 );
    for ( row = 0; row < 7; row++ )
        for ( col = 0; col < 15; col++ )
            VPutSqr( row, col, 0 );
    vidattr = saveattr;
    VMovCur( 20, 8 );
    VOutChr( 195 );
    VRepChr( 196, 64 );
    VOutChr( 180 );
    VMovCur( 21, 9 );
    VOutStr("Use the cursor keys to select your colors and to accept.");
    row = (orgattr & 0xF0) > 4;
    col = orgattr & 0x0F;
    VPutSqr( row, col, 0x80 );
    inchar = 0xFF;    /* null for no action below */
    do
    {
        if ( inchar == 0 ) /* only for function keys */
        {
            VPutSqr( row, col, 0 );
            switch ( getch() )
            {
                case 'K': /* LEFT */
                    if ( col > 0 ) col--;
                    break;
                case 'M': /* RIGHT */
                    if ( col < 15 ) col++;
                    break;
                case 'H': /* UP */
                    if ( row < 7 ) row--;
                    break;
                case 'P': /* DOWN */
                    if ( row < 7 ) row++;
                    break;
            }
            VPutSqr( row, col, 0x80 );
        }
    }
    while ( (inchar=getch()) != termch && inchar != ESCAPEKEY );
    vidattr = saveattr;
    return( inchar == ESCAPEKEY ? -1 : (row < 4) + col );
}

/*=====*/
void
PopBox( row, mstring )
    int row;
    char *mstring;
{
    int boxlen, boxcol;

    MUSTDRAW = TRUE;
    boxlen = strlen(mstring) + 5;
    boxcol = (80 - boxlen) / 2;
    MakeBox( row, boxcol, 5, boxlen );
    VMovCur( row + 1, boxcol + 3 );
    VOutStr( mstring );
}

```

SUBSTITUTE SHEET

```

}

/* ===== */
int
PopMessage( row, mstring )
    int row;
    char *mstring;
{
    int boxlen, boxcol;

    PopBox( row, mstring );
    VMovCur( row + 3, 18 );
    VOutStr( InMain? " Press any key to return to the Main Menu.":
              "Press any key to return to the Options Menu.");

    return( getch() );
}
/* ===== */

int
PromptKey()
{
    int row;
    union REGS keyregs;

    row = 9;

    putch(7); /* Bell */
    MakeBox( row, 2, 7, 75 );
    VMovCur( row + 1, 5 );
    VOutStr(
        "Now press the key combination which will activate the Security Screen.");
    VMovCur( row + 2, 5 );
    VOutStr(
        "A combination of Alt or
        with some other key is usually best.");
    VMovCur( row + 3, 5 );
    VOutStr(
        "The bell will sound again when your new security HOTKEY is accepted.");
    VMovCur( row + 4, 5 );
    VOutStr(
        "If the Security Screen appears instead, your HOTKEY is already set.");
    VMovCur( row + 5, 10 );
    VOutStr(
        "You may press <ESCAPE> if you wish to skip this procedure.");
    keyregs.h.ah = 0; /* Write character and attribute */
    int86( 0x16, &keyregs, &keyregs );
    return( keyregs.x.ax ); /* return whole scan code, not just character */
}
/* ----- */

char *
StrFind( pattern, string, length)
    char *pattern, *string;
    int length;
{
    char *strptr, *retstr, *pattptr;
    int gotmatch = FALSE;

    strptr = string;
    retstr = NULL;
    while ( !gotmatch && length-- 0 )

```

SUBSTITUTE SHEET

```

    if ( *strptr++ == *pattern )
    {
        retstr = strptr-1;
        pattptr = pattern+1;
        while ( *pattptr != 0 && *strptr++ == *pattptr++ ) length--;
        if ( *pattptr == 0 ) gotmatch = TRUE;
    }
    if ( !gotmatch ) retstr = NULL;
    return( retstr );
}
/*-----*/
int
Decode()
{
    return( TRUE );
}
/*-----*/
int
Encode()
{
    return( TRUE );
}
/*-----*/
#define HASHLEN 20

long
HashPass( str )
char *str;
{
    long hashval = 0;
    long *lptr;
    int i, dir, shiftcount, *front, *back;
    char hashstr[HASHLEN+1], *hashptr, *strptr;

    if ( *str == 0 ) return( 0L );
    hashptr = &hashstr[0];
    lptr = (long *) &hashstr[0];
    front = (int *) &hashstr[0];
    back = (int *) &hashstr[HASHLEN-2];
    strptr = str;
    dir = 1;
    for ( i = shiftcount = 0; i < HASHLEN; i++ )
    {
        *hashptr++ = (*strptr) < shiftcount;
        strptr += dir;
        if ( *strptr == 0 || strptr == str )
        {
            dir = -dir;
            strptr += dir;
            shiftcount++;
        }
    }
    for ( i = 0; i < HASHLEN/4; i++, front++, back-- )
    {
        *front += *(front+HASHLEN/4);
        *back -= *(back-HASHLEN/4) + *front;
    }
    hashval = *lptr + *(lptr+1) + *(lptr+2) + *(lptr+3) + *(lptr+4);
    if ( hashval == 0 ) hashval = 0xA55AC387;
    return( hashval );
}

```

SUBSTITUTE SHEET


```

}
/*-----*/
char*
KeyFind( keystr )
    char *keystr;
{
    char *newptr;

    newptr = StrFind(keystr, &lamabuffer, lamasize);
    return( newptr == NULL? newptr: newptr + strlen(keystr) );
}
/*-----*/
int
VerfPass( usernum )
    int usernum;
{
    char *basedata;
    long *lptr;
    int decodeok = TRUE;
    char str[HASHLEN + 1];

    if ( (basedata = KeyFind( RPLstring)) == NULL)
    {
        PopMessage( 12, "The LAMASYS.SYS driver is corrupted and unusable.");
        decodeok = FALSE;
    }
    else
    {
        lptr = (long*) (basedata + 2 + usernum*8);
        HideKeys = TRUE;
        if (!PromptBox(11,20,str,"    Please enter the CURRENT password.  ",
                      "AUTHORIZATION REQUEST"))
            return( FALSE );
        if ( *(lptr + 1) != HashPass( str ) )
        {
            putchar(7);
            PopMessage( 12, "Incorrect! Authority to modify is denied!");
            decodeok = FALSE;
        }
    }
    return( decodeok );
}
/*-----*/
int
PromptBox( row, inplen, inpstr, promptstr, titleprompt )
    int row, inplen;
    char *inpstr, *promptstr, *titleprompt;
{
    int boxlen, boxcol, promptcol, saveattr, retval;

    MUSTDRAW = TRUE;
    saveattr = vidattr;
    boxlen = strlen(promptstr) + 6;
    boxcol = (80-boxlen)/2;
    promptcol = boxcol + (boxlen-inplen)/2;
    if ( titleprompt == NULL )
        MakeBox( row, boxcol, 5, boxlen );
    else
    {
        MakeBox( row + 1, boxcol, 6, boxlen );
    }
}

```

SUBSTITUTE SHEET

32

```

    VMovCur( row, boxcol + ((boxlen-strlen(titleprompt))/2) + 1 );
    VOutStr( titleprompt );
}
VMovCur( row + 1, boxcol + 2 );
VOutStr( promptstr );
VMovCur( row + 3, promptcol );
vidattr = 0x0F;
VRepChr( ' ', inplen + 1 );
VMovCur( row + 3, promptcol );
retval = incurstr( inpstr, inplen );
HideKeys = FALSE;
vidattr = saveattr;
return( retval );
}
/*-----*/
int
SetOption( optnum, usernum )
    int  optnum, usernum;
{
    /*
    Set the given 'optnum' for the given 'usernum' in the LAMASYS.SYS version
    contained in the global 'lamabuffer'. The contents have been decoded and
    the current password verified previously, so no security verification is
    required here.
    */
    char *basedata, attr, offset;
    long *lptr, AccHold;
    int  decodeok = TRUE;
    int  ccount, *iptr, hkey;
    char str[81]; /* Large enough to hold any entry */

    switch( optnum )
    {
        case PASSWORD:
            if ( (basedata = KeyFind( RPLstring)) == NULL )
                decodeok = FALSE;
            else
            {
                {
                    lptr = (long*) (basedata + 2 + usernum*8);
                    if (!PromptBox( 13, 20, str,
                        " Please enter your new ACCESS CODE. ", NULL))
                        return( FALSE );
                    AccHold = HashPass( str );
                    if (!PromptBox( 14, 20, str,
                        " Please enter your new PASSWORD. ", NULL))
                        return( FALSE );
                    *lptr = AccHold;
                    *(lptr + 1) = HashPass( str );
                }
                break;
            }
        case HOTKEY:
            if ( (basedata = KeyFind( "hotkey=")) == NULL )
                decodeok = FALSE;
            else
            {
                {
                    hkey = PromptKey();
                    if ( (hkey & 0xff) == ESCAPEKEY )
                    {
                        PopMessage(10,"ESCAPE from HOTKEY request. No changes made.");
                        return( FALSE );
                    }
                }
            }
    }
}

```

SUBSTITUTE SHEET

33

```

    }
        putch(7); /* Bell */
        putch(7); /* Bell */
        putch(7); /* Bell */
    iptr = (int*) (basedata);
    *iptr = hkey;
}
break;
case ATTRIBUTE:
    if ( (basedata = KeyFind( "attribute=")) == NULL)
        decodeok = FALSE;
    else
    {
        if ( (attr = GetAttr( *basedata )) == -1 )
        {
            PopMessage(15,"ESCAPE from COLOR request. No changes made.");
            return( FALSE );
        }
        *basedata = attr;
    }
    break;
case TITLE:
    if ( (basedata = KeyFind( "MString=")) == NULL)
        decodeok = FALSE;
    else
    {
        if ( !PromptBox( 13, TITLELEN, str,
            "Please enter the text which you wish to appear on your Security Screen."
            , NULL))
            return( FALSE );
        *basedata += offset = (TITLELEN - strlen(str))/2;
        strnset( basedata, ' ', TITLELEN );
        basedata += offset;
        strcpy( basedata, str );
    }
    break;
case SECURITY:
    if ( (basedata = KeyFind( "hotkey=")) == NULL)
        decodeok = FALSE;
    else
    {
        basedata += 3;
        do
        {
            MakeBox( 15, 11, 9, 60 );
            VMovCur( 16, 27 );
            VOutStr("Security Level Selection Menu");
            VMovCur( 18, 14 );
            VOutStr("1. Minimum intrusiveness, but can't suppress SideKick.");
            VMovCur( 19, 14 );
            VOutStr("2. Controls SideKick, etc., but also halts spooling.");
            VMovCur( 20, 14 );
            VOutStr("3. Maximum security, but halts ALL background activity.");
            VMovCur( 22, 17 );
            VOutStr("Press 1 or 2 or 3 to select your security level.");
        }
        while ( (((hkey = getch()) != '1' || hkey != '3') && hkey != ESCAPEKEY);
        if ( hkey == ESCAPEKEY)
        {
            PopMessage(17," ESCAPE from LEVEL request. No changes made. ");
        }
    }
}

```

SUBSTITUTE SHEET

34

```

        return( FALSE );
    }
    *basedata = hkey - '0';
}
break;
case ALARMLEVEL:
    if ( (basedata = KeyFind( "maxatt=" )) == NULL)
        decodeok = FALSE;
    else
    {
        do
        {
            MakeBox( 17, 2, 5, 75 );
            VMovCur( 18, 4 );
            VOutStr(
                "Enter the number of incorrect attempts required to set off the alarm.");
            VMovCur( 20, 24 );
            VOutStr("Press a number between 1 and 9.");
        }
        while (((hkey=getch()) != '1' || hkey != '9') && hkey != ESCAPEKEY);
        if ( hkey == ESCAPEKEY)
        {
            PopMessage(18,"ESCAPE from ALARM setting. No changes made.");
            return( FALSE );
        }
        *basedata = hkey - '0';
    }
    break;
}
if ( !decodeok )
    PopMessage( 14,
        "The driver could not be decoded properly and may be corrupted!");
return( decodeok );
}
/*-----*/
int
SetPass()
/*
Given a file, which has been loaded into 'lamasize' bytes of 'lamabuffer':
1. Allow the password to be changed
2. re-encode the file
3. write and close the file
*/
{
    if ( !SetOption(PASSWORD, 0) ) return( FALSE );
    _chmod(LamaStr, 1, 0); /* UNHIDE */
    if ( (lamahandle = open(LamaStr, O_CREAT | O_RDWR | O_BINARY,
        S_IRREAD | S_IWRITE)) 0)
    {
        /* then the LAMA driver can't be written on the diskette. */
        PopMessage( 12,
            "The LAMASYS.SYS file can not be written. Please check the drive.");
        return(FALSE);
    }
    Encode();
    lseek( lamahandle, 0L, 0 );
    write( lamahandle, &lamabuffer[0], lamasize );
    close( lamahandle );
    _chmod(LamaStr, 1, FA_RDONLY | FA_HIDDEN | FA_SYSTEM); /* HIDE FOR LATER */
    return( TRUE );
}

```

SUBSTITUTE SHEET

35

```

/*-----*/
int
ReadLAMA( usetarget )
int usetarget;
{
    int orghandle;

    if ( (orghandle = open( usetarget? LamaStr: "LAMASYS.SYS",
                           O_RDONLY | O_BINARY)) 0)
    { /* then the LAMA driver isn't on the current drive */
        PopMessage( 12,
                    "The LAMASYS.SYS file is not on the current drive. Please correct.");
        return( FALSE );
    }
    lamasize = (unsigned) filelength(orghandle); /*LAMASYS.SYS must be */
    if ( lamasize BUFLen )
    {
        close( orghandle );
        PopMessage(12,"The LAMASYS.SYS file is invalid. Try to Install again.");
        return( FALSE );
    }
    read( orghandle, &lamabuffer[0], lamasize );
    close( orghandle );
    return( Decode() );
}
/*-----*/
PopOKInst()
{
    PopMessage(12,
               "The installation was successful. It will take effect after you reboot.");
}
/*=====*/
void
FixRefs( filename, orgstr, newstr )
char *filename, *orgstr, *newstr;
{
    FILE *fp;
    int c, pattchar, pattlen;
    long fileloc;
    char *copy;

    if( (fp = fopen( filename, "r+" )) == NULL ) return;
    pattchar = *orgstr;
    pattlen = strlen( orgstr );
    do
    {
        while( (c=getc(fp)) != EOF && c != pattchar && tolower(c) != pattchar);
        if( c != EOF )
        {
            fileloc = ftell( fp );
            copy = orgstr;
            while(*++copy && (*copy == (c=getc(fp)) || *copy == tolower(c))) ;
            if ( !*copy ) fileloc -= 2;
            fseek( fp, fileloc, SEEK_SET );
            if ( !*copy ) fwrite( newstr, pattlen, 1, fp );
        }
    } while( c != EOF );
    fclose( fp );
}
/*=====*/

```

SUBSTITUTE SHEET

36

```

int
FixDevices( orgfile, destfile, drive, dofix )
char *orgfile, *destfile;
int drive, dofix;
{
    FILE *orgfp, *destfp;
    int c, nextc, pattchar, pattlen;
    char *copy;

    orgfp = fopen( orgfile, "rt" );
    if( (destfp = fopen( destfile, "wt" )) == NULL )
    {
        if ( orgfp != NULL ) fclose(orgfp);
        return(FALSE);
    }
    fputs( DeviceStr, destfp );
    if ( orgfp != NULL )
    do
    {
        while( (c=getc(orgfp)) != EOF && c != 'd' && c != 'D') putc(c, destfp);
        if( c != EOF )
        {
            /* we have found a 'D', DEVICE? */
            putc( c, destfp ); /* must output character in any case */
            if ( dofix )
            {
                copy = DevPattern;
                while(*++copy && (*copy == (c=getc(orgfp)) || *copy == tolower(c)))
                putc( c, destfp );
                if ( *copy )
                putc( c, destfp ); /* must still put out last character */
            }
            else
            {
                /* We have found the word DEVICE */
                while( (c=getc(orgfp)) == ' ') putc( ' ', destfp );
                putc( c, destfp );
                if ( c == '=' ) /* then we have a device specification */
                {
                    while( (c=getc(orgfp)) == ' ') putc( ' ', destfp );
                    if ( (nextc = getc(orgfp)) != ':' ) /* no drive here */
                    {
                        putc( drive, destfp );
                        putc( ':', destfp );
                    }
                    putc( c, destfp );
                    putc( nextc, destfp ); /* complete stream */
                }
            }
        }
    } while( c != EOF );
    if ( orgfp != NULL ) fclose(orgfp);
    fclose( destfp );
    return( TRUE );
}

/*=====*/
int
RestConf()
{
    FILE *orgfp, *destfp;
    char *lineptr, inbuff[250];

```

SUBSTITUTE SHEET

```

if( (orgfp = fopen( ConfSav, "rt" )) == NULL ) return(FALSE);
if( (destfp = fopen( ConfStr, "wt" )) == NULL )
{ fclose(orgfp); return(FALSE); }
while ( (lineptr = fgets(inbuff, 250, orgfp)) != NULL )
if ( StrFind( "LAMASYS.SYS", lineptr, strlen(lineptr) ) == NULL &&
    StrFind( "lamasys.sys", lineptr, strlen(lineptr) ) == NULL )
    fputs( lineptr, destfp );
fclose( orgfp );
fclose( destfp );
return( TRUE );
}
/*-----*/
int
TargetStat( line )
int line;
{
    char buffer[512];
    int retval, target;
/* Returns 0x80 if not ready, 3 if write-protected */
    target = targetdisk - 'A';
    retval = biosdisk( 2, target, 0, 0, 1, 1, &buffer ); /* read twice! */
    if ( (retval = biosdisk( 2, target, 0, 0, 1, 1, &buffer )) == 0 )
        retval = biosdisk( 3, target, 0, 0, 1, 1, &buffer );
    if ( retval == 0x80 )
        PopMessage( line, "The target diskette is not ready. Please correct.");
    else
        if ( retval == 3 )
            PopMessage( line,
                "The target diskette is write protected. Please correct.");
    return(retval);
}
/*-----*/
void
Finstall()
{
/*
    Finstall is called by the installation process to configure the system to
    boot from a floppy disk in drive A:. This copies and modifies the required
    files from the current boot drive.
*/
    int orghandle, tstat;
    char bootchar;
    FILE *fp;

    if ( TargetStat(12) ) return; /* 3 == write-protected, 0x80 if not ready */
    *CommStr = targetdisk;
    if ( access( CommStr, 0 ) != 0 )
    { /* then the target disk isn't a boot disk */
        PopMessage(12, "The target diskette is not a boot disk. Please correct.");
        return;
    }
    *LamaStr = targetdisk;
    if ( access( LamaStr, 0 ) == 0 ) /* the system is already installed */
    {
        PopMessage( 12,
            "LAMASYS.SYS is already installed on the target diskette. Try Uninstall.");
        return;
    }
    if ( !ReadLAMA(FALSE) || !SetPass() ) return;
/* Now look for the hard boot disk, use its config.sys, and anti-boot it. */

```

SUBSTITUTE SHEET

38

```

if ( clineboot != 'N' ) /* boot disk on command line */
    hardboot = clineboot;
else
{
    hardboot = 'N'; /* assume no boot disk to start */
    *LCommStr = *CommStr = bootchar = firstfixed;
    while ( bootchar < topdisk && hardboot == 'N' )
    {
        if ( access( LCommStr, 0 ) == 0 ) /* Previously installed, quit */
        {
            PopMessage( 12,
                "The hard disk has already been modified. Try Uninstall first.");
            return;
        }
        if ( access( CommStr, 0 ) == 0 )
            hardboot = bootchar;
        else
            bootchar ++;
        *LCommStr = *CommStr = bootchar; /* Set next Drive */
    }
}
if ( hardboot != 'N' )
{
    *AutoStr = *CommStr = *LCommStr = *ConfStr = *ConfSav = hardboot;
    if ( hardboot == firstfixed ) /* Floppy-only installation */
    { /* The user normally boots from the floppy drive 'hardboot' */
        rename( ConfStr, ConfSav ); /* rename config.sys - config.sav */
        *ConfStr = targetdisk;
        FixDevices( ConfSav, ConfStr, hardboot, FALSE );
        unlink( ConfSav );
    }
    else
    { /* The user normally boots from the drive 'hardboot' */
        rename( CommStr, LCommStr );
        rename( ConfStr, ConfSav ); /* rename config.sys - config.sav */
        *ConfStr = targetdisk;
        FixDevices( ConfSav, ConfStr, hardboot, TRUE );
        unlink( ConfSav );
        FixRefs( ConfStr, "command.com", "LCOMAND.COM" ); /* */
        *AutoStr = hardboot;
        FixRefs( AutoStr, "command.com", "LCOMAND.COM" );
        *AutoStr = targetdisk;
        if ( (fp = fopen( AutoStr, "wt" )) == NULL )
            if ( (lahandle = open( AutoStr, O_CREAT | O_RDWR | O_BINARY,
                S_IRUSR | S_IWUSR )) == 0 )
            { /* then AUTOEXEC.BAT file can't be written on the diskette. */
                PopMessage( 12,
                    "The new AUTOEXEC.BAT file can not be written. Please check the drive.");
                return;
            }
        putc( hardboot, fp );
        fputs( autoset1, fp );
        putc( hardboot, fp );
        fputs( autoset2, fp );
        fclose( fp );
    }
}
}
PopOKInst();
}
/*-----*/

```

SUBSTITUTE SHEET


```

void
Install()
{
    if ( targetdisk firstfixed )
    {
        Finstall(); /* only if a floppy drive is explicitly given */
        return;
    }
    *LamaStr = *ConfStr = *ConfSav = targetdisk;
    if ( access( LamaStr, 0 ) == 0 ) /* the system is already installed */
    {
        PopMessage( 12,
            "LAMASYS.SYS is already installed on the target disk. Try Uninstall.");
        return;
    }
    if ( !ReadLAMA(FALSE) || !SetPass() ) return;
    rename( ConfStr, ConfSav ); /* rename config.sys - config.sav */
    if ( FixDevices( ConfSav, ConfStr, targetdisk, FALSE ) )
    {
        unlink( ConfSav );
        PopOKInst();
    }
    else
    {
        rename( ConfSav, ConfStr ); /* rename config.sys - config.sav */
        PopMessage( 12,
            "The CONFIG.SYS file cannot be modified. Installation failed.");
    }
}
/*=====*/
PopOKUninst()
{
    PopMessage( 15,
        "The Uninstall process was successful. You should reboot to complete it.");
}
/*=====*/
int
CheckLAMA()
{
    *LamaStr = targetdisk;
    if ( access( LamaStr, 0 ) != 0 )
    {
        /* then the system is not installed on the target disk */
        PopMessage( 15, "APO is not installed on the target disk.");
        return(FALSE);
    }
    _chmod(LamaStr, 1, 0 ); /* UNHIDE */
    unlink( LamaStr );
    return( TRUE );
}
/*=====*/
void
FUninstall()
{
    char bootchar;

    if ( TargetStat(15) ) return; /*3 if write-protected, 0x80 if not ready*/
    if ( !CheckLAMA() ) return; /* Can't proceed, not installed */
    /* Find the modified boot disk and restore its config.sys and command.com */
    if ( clineboot != 'N' ) /* boot disk on command line */
        hardboot = clineboot;
}

```

```

else
    {
        hardboot = 'N'; /* assume no boot disk to start */
        *LCommStr = bootchar = firstfixed; /* Start with C: */
        while ( bootchar < topdisk && hardboot == 'N' )
        {
            if ( access( LCommStr, 0 ) == 0 ) /* Previously installed, OK */
                hardboot = bootchar;
            else
                bootchar++;
            *LCommStr = bootchar; /* Set next Drive */
        }
    }
if ( hardboot != 'N' )
    { /* The drive 'hardboot' is the modified boot disk */
        *AutoStr = *CommStr = *LCommStr = hardboot;
        if ( hardboot == firstfixed ) /* must restore original hard diskfiles */
        { /* The user normally boots from the floppy drive 'hardboot' */
            rename( LCommStr, CommStr );
            remove( LCommStr ); /* Get rid of LCOMMAND.COM one way or another */
            FixRefs( AutoStr, "lcomand.com", "COMMAND.COM" );
            *AutoStr = targetdisk;
            unlink( AutoStr );
        }
        *ConfStr = *ConfSav = targetdisk;
        if ( rename( ConfStr, ConfSav ) == 0 ) /* Success */
        {
            *ConfStr = hardboot;
            RestConf();
            unlink( ConfSav );
            if ( hardboot == firstfixed ) /* must restore original files */
                FixRefs( ConfStr, "lcomand.com", "COMMAND.COM" );
        }
    }
PopOKUninst();
}
/* ===== */
void
Uninstall()
{
    *LamaStr = targetdisk;
    if ( targetdisk == firstfixed )
        if ( TargetStat(14) ) /* 3 if write-protected, 0x80 if not ready */
            return;
    if ( access( LamaStr, 0 ) != 0 )
    { /* then the system is not installed on the target disk */
        PopMessage( 14, "APO is not installed on the target disk." );
        return;
    }
    if ( !ReadLAMA(TRUE) || !VerfPass(0) ) return;
    if ( targetdisk == firstfixed )
    {
        FUninstall(); /* only if a floppy drive is explicitly given */
        return;
    }
    if ( !CheckLAMA() ) return; /* Can't proceed, not installed */
    *ConfStr = *ConfSav = targetdisk;
    if ( rename( ConfStr, ConfSav ) == 0 ) /* Success */
    {
        RestConf();
    }
}

```

SUBSTITUTE SHEET

41

```

        unlink( ConfSav );
    }
    PopOKUninst();
}
/*-----*/
void
DispOptions()
{
    clrscr( 177 );
    MakeBox ( 3, 10, 19, 60 );
    VMovCur( 5, 28 );
    VOutStr( LamaTitle1 );
    VMovCur( 6, 30 );
    VOutStr( "Security Options Menu" );
    VMovCur( 8, 20 );
    VOutStr( "1. Select the Security Hotkeys" );
    VMovCur( 10, 20 );
    VOutStr( "2. Select the Security Screen Colors" );
    VMovCur( 12, 20 );
    VOutStr( "3. Enter the Security Screen Title" );
    VMovCur( 14, 20 );
    VOutStr( "4. Select the Security Level" );
    VMovCur( 16, 20 );
    VOutStr( "5. Select the Alarm Level" );
    VMovCur( 18, 20 );
    VOutStr( "6. Save and return to the Main Menu" );
    VMovCur( 20, 32 );
    VOutStr( LamaChoose );
}
/*-----*/

void OptionMenu()
/*
    Using the handle of the read/write, binary file, which is loaded
    into 'lamasize' bytes of 'lamabuffer':
    1. Verify password authority
    2. Decode the file
    3. Set any options
    4. re-encode the file
    5. write and close the file
*/
{
    int          inchar, DONE;

    MUSTDRAW = TRUE;
    *LamaStr = targetdisk;
    if ( targetdisk firstfixed )
        if ( TargetStat(14) ) /* 3 if write-protected, 0x80 if not ready */
            return;
    if ( access( LamaStr, 0 ) != 0 )
    { /* then the system is not installed on the target disk */
        PopMessage( 14, "APO is not installed on the target disk." );
        return;
    }
    if ( !ReadLAMA(TRUE) || !VerfPass(0) || !Decode() ) return;
    InMain = FALSE;
    DispOptions();
    DONE = FALSE;
    while ( !DONE )
    {

```

SUBSTITUTE SHEET

42

```

do inchar = getch(); while ( inchar '1' && inchar '6');
switch( inchar )
{
case '1': SetOption(HOTKEY, 0);
break;
case '2': SetOption( ATTRIBUTE, 0 );
break;
case '3': SetOption( TITLE, 0 );
break;
case '4': SetOption( SECURITY, 0 );
break;
case '5': SetOption( ALARMLEVEL, 0);
break;
case '6': DONE = TRUE;
break;
}
if ( !DONE ) DispOptions();
}
InMain = TRUE; /* back at main menu level */
_chmod(LamaStr, 1, 0); /* UNHIDE */
if ( (lamahandle = open(LamaStr, O_CREAT | O_RDWR | O_BINARY,
S_IRREAD|S_IWRITE)) 0)
{ /* then the LAMA driver can't be written on the diskette. */
PopMessage( 12,
"The LAMASYS.SYS file can not be written. Please check the drive.");
return;
}
Encode();
lseek( lamahandle, 0L, 0 );
write( lamahandle, &lamabuffer[0], lamasize );
close( lamahandle );
_chmod(LamaStr, 1, FA_RDONLY | FA_HIDDEN | FA_SYSTEM); /* HIDE FOR LATER */
return;
}
/* ===== */
void
Password()
{
*LamaStr = targetdisk;
if ( targetdisk firstfixed )
if ( TargetStat(14) ) /* 3 if write-protected, 0x80 if not ready */
return;
if ( access( LamaStr, 0 ) != 0 )
{ /* then the system is not installed on the target disk */
PopMessage( 14, "APO is not installed on the target disk.");
return;
}
if ( ReadLAMA(TRUE) && VerfPass(0) ) SetPass();
}
/* ===== */
void main(argc,argv)
int argc;
char *argv[];
{
int DONE = FALSE;
int i;
char *parmptr, inchar, bootchar;

vidregs.h.bh = 0;
topdisk = 'A' + setdisk( getdisk() ); /* Limit for hard boot disk */

```

SUBSTITUTE SHEET

43

```

firstfixed = (biosequip() & 0x00C0) > 6;
if ( firstfixed == 0 ) firstfixed = 1; /* 1 floppy systems boot from C: */
firstfixed += 'B';
i = 1;
*RPLstring = '3'; /* Just to be obscure */
while ( argc-- 1 )
{
    if ( EquivStr( argv[i], "?" , 999 ) || EquivStr( argv[1], "/H", 999 ) )
    { /* Give some brief command-line help, then exit */
        printf( " This program is used to install the APO high security\n" );
        printf( "software onto either the hard disk which has is used to boot\n" );
        printf( "or else onto a floppy disk which has already been formatted\n" );
        printf( "as a boot disk. Any valid disk drive may be entered as the\n" );
        printf( "command line option. If no option is given, drive C: will be\n" );
        printf( "assumed for the target disk.\n" );
        exit(0);
    }
    parmptr = argv[i];
    inchar = toupper(*parmptr);
    if ( EquivStr( parmptr, "boot=", 5 ) )
    {
        for ( parmptr += 5; *parmptr == ' '; parmptr++ );
        inchar = toupper(*parmptr); /* force upper case */
        if ( inchar == 'A' && inchar != 'topdisk' ),
            clineboot = inchar; /* force boot choice */
    }
    else
    if ( *(parmptr+1) == ':' && inchar == 'A' && inchar != 'topdisk' ),
        targetdisk = inchar; /* if == "A:", leave as 0 */
    else
    { /* The command-line option wasn't recognized */
        printf( "An option was not recognized. Try ? for help" );
        exit(1); /* exit with error code == 1 */
    }
    i++;
}
*(RPLstring+5) = 'P'; /* Also to be obscure */
if ( targetdisk == 'N' ) /* no target/boot disk on command line */
{
    bootchar = firstfixed;
    while ( bootchar != 'topdisk' && targetdisk == 'N' ),
    {
        *CommStr = bootchar; /* Set next Drive */
        if ( access( CommStr, 0 ) == 0 )
            targetdisk = bootchar; /* use first booting hard disk */
        else
            bootchar++;
    }
}
if ( targetdisk == 'N' )
{
    PopMessage( 12, "No default boot disk could be found. Exiting!" );
    exit(1);
}
while ( !DONE )
    switch( InitMenu() )
    {
        case '1': Install();
            break;
        case '2': Password();
    }

```

SUBSTITUTE SHEET

44

```

        break;
    case '3': OptionMenu();
        break;
    case '4': UnInstall();
        break;
    case '5': DONE = TRUE;
    }
    vidattr = 7;
    clrscr();
    exit(0);
}
/*-----*/

int
EquivStr( str1, str2, complen )
    char *str1, *str2;
    int complen;
/*
    Compare strings, "str1" and "str2" for equivalence of the first
    "complen" characters. This function allows case insensitivity by mapping
    all alphabetic characters to upper case.
    Accepts if both are equivalent but less than the given length.
*/
{
    char char1;
    int i;

    for ( i = 1; i < complen; i++ ),
        if ( (char1 = toupper(*str1++)) != toupper(*str2++) )
            return( FALSE );
        else
            if ( char1 == '\0' ) i = complen;
    return( TRUE );
}

/*=====*/
/*
    THIS FUNTION PREVENTS TYPING OUT OF THE INPUT FIELD AND SOUNDS THE BELL
    WHEN AN ATTEMPT IS MADE TO DO SO. IT TERMINATES BY THE 'termch' CHARACTER.
    IT ALSO HANDLES BACK-SPACES, BUT NO OTHER INTRA-LINE EDITING IS IMPLEMENTED.
    Returns FALSE only if the escape key is pressed.
*/
int
incurstr( str, ssize)
    int      ssize;
    char     *str;
{
    int  n, savn, backout;
    char inchar;

    n = 0;
    while ( (inchar = getch()) != termch && inchar != ESCAPEKEY )
        if ( inchar == '\0' )
            getch();
        else
            if ( inchar != BACKSPACE )          /* if not back-space.... */
            {
                savn = n + 1;
                *str++ = inchar;
                if ( inchar == '\t' ) n += tabincr - 1;
            }

```

SUBSTITUTE SHEET

```

else
    if (inchar == '\n')
    {
        n++;
        if (n == ssz && !HideKeys) VOutChr('^');
        inchar |= 64;
    }
    if (n == ssz)
    {
        VOutTTY("\007");
        n = savn;
        str--;
    }
else
    if (!HideKeys) VOutChr(inchar);
}
else /* have backspace */
    if (n > 0)
    {
        str--;
        if (!HideKeys)
        {
            backcount = 1;
            if (*str == '\t') backcount = tabincr;
            else
                if (*str == '\n') backcount = 2;
            n -= backcount;
            for (; backcount > 0; backcount--)
            {
                VOutTTY(BACKSPACE);
                VPutChr(' ');
            }
        }
    }
}
*str = '\0';
return( inchar == ESCAPEKEY? FALSE: TRUE );
}
/* ===== */

```

SUBSTITUTE SHEET

46

; Copyright (c) 1988 Lama Systems Inc.

EXECUTE

```

        name      LAMA
        page      55,132
        title     'LAMA -- Lama security software driver: password only '

ZERORAM EQU 0          ; set to 1 for no-hotkey, min memory model, else 0

code      segment      public 'CODE'

LAMA      proc          far
          assume        cs:code,ds:code,es:code
          org           0

;
Max_Cmd   equ          16          ; MS-DOS command code maximum:
;                                     ; this is 16 for MS-DOS 3.x
;                                     ; and 12 for MS-DOS 2.x.
;
; device driver header
;
Header     dd          -1          ; link to next device, -1 = end of list
          dw           8000h      ; device attribute word:
;                                     ; bit 15 = 1 for character devices
;                                     ; bit 14 = 1 if driver can handle IOCTL
;                                     ; bit 13 = 1 if block driver & non-IBM format
;                                     ; bit 12 = 0
;                                     ; bit 11 = 1 if OPEN/CLOSE/RM supported (DOS 3.x)
;                                     ; bit 10 = 0
;                                     ; bit 9 = 0
;                                     ; bit 8 = 0
;                                     ; bit 7 = 0
;                                     ; bit 5 = 0
;                                     ; bit 4 = 0
;                                     ; bit 3 = 1 if CLOCK device
;                                     ; bit 2 = 1 if NUL device
;                                     ; bit 1 = 1 if Standard Output
;                                     ; bit 0 = 1 if Standard Input

          dw           Strat      ; device "Strategy" entry point

          dw           Intr       ; device "Interrupt" entry point

          db           'LAMA1988' ; ???
;                                     ; character device name, 8 char, or if block
;                                     ; device, no. of units in first byte followed by
;                                     ; 7 don't care bytes

;
; Double word pointer to Request Header
; Passed to Strategy routine by MS-DOS
;
RH_Ptr     dd          ?
          page

;
; MS-DOS Command Codes dispatch table.
; The "Interrupt" routine uses this table and the
; Command Code supplied in the Request Header to
; transfer to the appropriate driver subroutine.
;

```

SUBSTITUTE SHEET

48

; The strategy Routine is passed the address of the Request
 ; Header in ES:BX, which it saves in a local variable and then
 ; returns to MS-DOS.

```
Strat      proc      far
           ; save address of Request Header
           mov       word ptr cs:[RH_Ptr],bx
           mov       word ptr cs:[RH_Ptr+2],es
           ; etc.
           ret
Strat      endp      ; back to MS-DOS
```

; Device Driver "Interrupt Routine"
 ; This entry point is called by MS-DOS immediately after the
 ; call to the "Strategy Routine", which saved the long address
 ; of the Request Header in the local variable "RH_Ptr".

; The "Interrupt Routine" uses the Command Code passed in the
 ; Request Header to transfer to the appropriate device handling
 ; routine. Each Command Code routine must place any necessary return
 ; information into the Request Header, then perform a Near Return
 ; with AX = Status.

```
Intr       proc      far

           push      ax      ; save general registers
           push      bx
           push      cx
           push      dx
           push      ds
           push      es
           push      di
           push      si
           push      bp

           push      cs      ; make local data addressable
           pop       ds

           les       di,[RH_Ptr] ; let ES:DI = Request Header

           ; get BX = Command Code
           mov       bl,es:[di.Command]
           xor       bh,bh
           cmp       bx,Max_Cmd ; make sure it's legal
           jle       Intr1      ; jump, function code is ok.
           mov       ax,8003h    ; set Error bit and "Unknown Command" code
           jmp       Intr2

Intr1:     shl       bx,1        ; form index to Dispatch table and
                                   ; branch to driver routine
           call      word ptr [bx+Dispatch]
                                   ; should return AX = status
           les       di,[RH_Ptr] ; restore ES:DI = addr of Request Header

Intr2:     or        ax,0100h    ; merge Done bit into status and
           mov       es:[di.Status],ax
                                   ; store into Request Header
           pop       bp          ; restore general registers
           pop       si
           pop       di
```

SUBSTITUTE SHEET

49

```

pop     es
pop     ds
pop     dx
pop     cx
pop     bx
pop     ax
ret

```

; back to MS-DOS

page

; Command Code subroutines called by Interrupt Routine

; These routines are called with ES:DI pointing to
; the Request Header.; They should return AX = 0 if function was completed successfully,
; or AX = 8000H + Error code if function failed.

```

Media_Chk proc      near          ; function 1 = Media Check
; etc.
xor     ax,ax
ret
Media_Chk endp

```

```

Build_Bpb proc      near          ; function 2 = Build BPB
; etc.
xor     ax,ax
ret
Build_Bpb endp

```

```

Read proc      near          ; function 4 = Read
; etc.
xor     ax,ax
ret
Read endp

```

```

Inp_Stat proc near          ; function 6 = Input Status
; etc.
xor     ax,ax
ret
Inp_Stat endp

```

```

Write proc      near          ; function 8 = Write
; etc.
xor     ax,ax
ret
Write endp

```

```

Write_Vfy proc      near          ; function 9 = Write with Verify
; etc.
xor     ax,ax
ret
Write_Vfy endp

```

```

Outp_Flush proc      near          ; function 11 = Flush Output Buffers
; etc.
xor     ax,ax
ret
Outp_Flush endp

```

SUBSTITUTE SHEET

```

                                50
IOCTL_Wrt proc                 near        ; function 12 = I/O Control Write
    ; etc.
    xor                         ax,ax
    ret
IOCTL_Wrt endp

Dev_Open proc                 near        ; function 13 = Device Open
    ; etc.
    xor                         ax,ax
Dev_Open endp

Dev_Close proc               near        ; function 14 = Device Close
    ; etc.
    xor                         ax,ax
    ret
Dev_Close endp

Rem_Media proc               near        ; function 15 = Removable Media
    ; etc.
    xor                         ax,ax
    ret
Rem_Media endp

Out_Busy proc                 near        ; function 16 Output Until Busy
    ; etc.
    xor                         ax,ax
    ret
Out_Busy endp
page
;=====

```

```

videocall    macro
    pushf
    call dword ptr cs:VidVect
endm

```

```

keybdcall    macro
    pushf                ; Call BIOS keyboard routine,
    call dword ptr cs:KbVect ; but directly to avoid traps
endm

```

```

diskcall     macro
    int        13h                ; call BIOS DISK function
endm

```

```

TRUE         EQU        -1
FALSE        EQU        0

```

```

; ASCII definitions
;

```

```

bel          equ        07h
backsp equ 08h
cr           equ        0dh
esckey      equ        1bh
home        equ        1eh
lf          equ        0ah
soh         equ        01h
eot         equ        04h

```

```

; start of header
; end of text

```

SUBSTITUTE SHEET

```

                                51
eom equ 0 ; DOS end-of-message
ack equ 06h
nack equ 15h
cancel equ 18h
xon equ 11h ;
Q
xoff equ 13h ;

IDkey db 'LamaID='
IF ZERORAM db 'ZR' ; ZeroRam version
ELSE db 'HK' ; Hot Key version
ENDIF db 'LamaID='

scankey db 'hotkey='
scan_code dw 0300h ;(Ctrl-2) scan code to start/stop program.
KbVect dw 2 DUP (0) ;Holds old Interrupt 16h BIOS vector.
inLAMA db FALSE ;Busy flag, for our stack
;----- Intercept keystroke reading -----
KeyInt16 proc far
    sti ;Turn interrupts back on.
    cmp ah,0 ;See if was request for an actual key.
    je come_back ;If so, we'll check it first.
    skip_us: jmp dword ptr cs:KbVect ;Otherwise, let go straight to caller.
    come_back:
        cmp cs:inLAMA, FALSE ;Is LAMA routine busy?
        jne skip_us ;If so, pass all keys along.
    getkey: keybdcall
        cmp ax,cs:scan_code ;Is scan code correct?
        je now_busy ;If so, go to work.
        iret ;Else just pass on key to caller.
    now_busy:
        call LAMATEST ;Go to LAMA.
        mov ah,0 ;Reset AH to BIOS key read function.
        jmp getkey ;Go get new key from BIOS.
KeyInt16 endp

flag_str db '332 RPL' ; marks boundary of password
numusers db 1 ; used by INST utility
maxusers db 1 ; used by INST utility
;
U1Access1 dw 318Bh ; "LAMA"
U1Access2 dw 363Bh
U1Password1 dw 0B35Dh ; "LOCK"
U1Password2 dw 0F7A2h
hashstr db 21 DUP (0) ; HASHLEN + 1
hashptr dw 0
lptr dw 0
front dw 0
back dw 0
dir dw 1
shiftcount dw 0
icount dw 0
;
HashPass PROC near
    CMP BYTE PTR pw_buf,00H
    JNE L0021

```

SUBSTITUTE SHEET

52

```

XOR      DX,DX
MOV      AX,DX
JMP      L00DA
L0021:   MOV      AX, offset hashstr
MOV      hashptr, AX
MOV      lptr, AX
MOV      SI, ax
MOV      AX, offset hashstr + 18      ; hashstr[HASHLEN-2]
MOV      back, AX
mov      DI, offset pw_buf
MOV      dir, 0001H
XOR      AX,AX
MOV      shiftcount, AX
MOV      icount, AX
JMP      SHORT L0074
L0048:   MOV      AL,[DI]
MOV      CX, shiftcount
SHL      AL,CL
MOV      BX, hashptr
MOV      [BX],AL
INC      hashptr
ADD      DI, dir
CMP      BYTE PTR [DI],00H
JE       L0064
cmp      di, offset pw_buf
JNB      L0071
L0064:   MOV      AX, dir
NEG      AX
MOV      dir, AX
ADD      DI,AX
INC      shiftcount
L0071:   INC      icount
L0074:   CMP      icount, + 14H
JL       L0048
MOV      icount,0000H
JMP      SHORT L009A
L0081:   MOV      AX,[SI + 0AH]
ADD      [SI],AX
MOV      BX, back
MOV      AX,[BX-0AH]
ADD      AX,[SI]
SUB      [BX],AX
INC      icount
ADD      SI, + 02H
SUB      back, + 02H
L009A:   CMP      icount, + 05H
JL       L0081
MOV      BX, lptr
MOV      DX,[BX + 02H]
MOV      AX,[BX]
ADD      AX,[BX + 04H]
ADC      DX,[BX + 06H]
ADD      AX,[BX + 08H]
ADC      DX,[BX + 0AH]
ADD      AX,[BX + 0CH]
ADC      DX,[BX + 0EH]
ADD      AX,[BX + 10H]
ADC      DX,[BX + 12H]
MOV      CX,DX
OR       CX,AX

```

SUBSTITUTE SHEET

```

JNE      L00DA
MOV      DX,0A55AH
MOV      AX,0C387H
L00DA:   ret
HashPass ENDP
;*****
;
DummyIRET proc far
iret
DummyIRET endp
;*****
; DISPlay Character in AL
disp_ch proc near
    push    di
    push    si
    mov     ah, 0Eh      ; BIOS write tty
    mov     bh, DispPage
    mov     bl, LamAttr
    videocall
    pop     si
    pop     di
    ret
disp_ch endp
;*****
; DISPlay STRing from memory
disp_str proc near
dislp:   lodsb
        cmp     al, 0
        je      dispout
        call    disp_ch
        jmp     dislp

dispout: ret
disp_str endp
;*****
; clr_line --- clears line (0 at top, 24 at bottom)
; preserves affected registers
clr_line PROC NEAR
    push    ax      ; save general registers
    push    bx
    push    cx
    push    dx
    push    di
    push    si
    mov     ah, 6    ; init window function
    mov     al, 0    ; # lines to scroll
    mov     bh, LamAttr ; attribute for blanked area
    cmp     splitscr, 0
    je      doclr
    mov     bh, 0
doclr:   mov     ch, dh ; y coordinate, upper left coordinate of window
        mov     cl, 0  ; x coordinate, upper left coordinate of window
        mov     dl, MaxCol ; x coordinate, lower right coordinate of window
        cmp     dl, 79
        jbe     dockmod
        mov     dl, 79
dockmod: cmp     VMODE, 7
        jbe     doVcall
        mov     dl, 32 ; x coordinate, lower right coordinate of window
doVcall: videocall
        pop     si

```

SUBSTITUTE SHEET

54

```

        pop        di
        pop        dx
        pop        cx
        pop        bx
        pop        ax
        ret

clr_line endp
;*****

; CLRSCR --- clears screen
; preserves affected registers;
; assumes variable "column" set to # columns on screen - 1
clrscr    proc near
        push        ax                ; save general registers
        push        bx
        push        cx
        push        dx
        push        di
        push        si
        mov         ax,0600h          ; AH == 6 for "scroll or initialize window"
                                         ; AL == 0 for # lines to scroll
        xor         cx,cx            ; CL,CH == x,y coordinates of upper left corner
        xor         bh,bh
        cmp         VMODE, 7
        jbe         CGAclr
        mov         dx, 0120h        ; x,y lower right
        jmp         doclrs
CGAclr:
        mov         dl, MaxCol
        cmp         dl, 79
        jbe         dosetdh
        mov         dl, 79
dosetdh: mov     dh, 05h                ; x,y lower right
        cmp         splitscr, 0
        jne         doclrs
        mov         bh, LamAttr        ; attribute for blanked area
        mov         dh, 18h            ; x,y lower right
doclrs:   videocall
        pop         si
        pop         di
        pop         dx
        pop         cx
        pop         bx
        pop         ax
        ret
clrscr endp
;*****
; Used by LAMATEST to steal vectors from debuggers and hidden here
; just to be obtuse. Note that CLI and STI are done elsewhere.
StealVects proc near
        les         di, ObscureZ2
        mov         ax, es:[14h]        ; Save current PRINTSCREEN vector
        mov         SavePscrn, ax
        mov         ax, es:[16h]
        mov         SavePScrn+2, ax
        mov         ax, es:[20h]        ; Save original BIOS timer vector
        mov         SaveTimer, ax
        mov         ax, es:[22h]
        mov         SaveTimer+2, ax
        mov         ax, es:[70h]        ; Save original BIOS TIC vector

```

SUBSTITUTE SHEET


```

                                55
mov     SaveTic,ax
mov     ax, es:[72h]
mov     SaveTic + 2,ax
cmp     stealv, 2             ; security level 2?
jb      nosteal               ; skip if less than 2
mov     si, offset InitVects
mov     cx, VectLenVar
rep     movsw
mov     di, ObscureNMI
mov     ax, offset cs:DummyIRET ; make dummy printscreen vect.
stosw   ; mov     es:[14h], ax
mov     ax, cs
stosw   ; mov     es:[16h], bx
nosteal:
mov     di, ObscureV
mov     ax, offset cs:DummyIRET ; make dummy printscreen vect.
stosw   ; mov     es:[14h], ax
mov     ax, cs
stosw   ; mov     es:[16h], bx
mov     di, 24h               ; point to 0:24h
mov     ax, offset cs:INT_9 ; use local to defeat ctl-alt-del
stosw
mov     ax, cs
stosw
cmp     stealv, FALSE ; steal timer unless minimum security level
je      stealout
mov     di, 20h               ; point to INT 8 - Timer
mov     ax, TimerVect
stosw   ; mov     es:[20h], ax
mov     ax, TimerVect + 2
stosw   ; mov     es:[22h], bx
mov     di, 70h               ; point to INT 1C - Timer
mov     ax, TicVect
stosw   ; mov     es:[70h], ax
mov     ax, TicVect + 2
stosw   ; mov     es:[72h], bx
stealout:ret
StealVects endp
;*****

```

; Set cursor position to (0,y). (0,0) is upper-left corner.

```

setc_row proc near
push     ax                    ; save general registers
push     bx
push     dx
push     di
push     si
mov     ah,2
mov     bh, DispPage
xor     dl, dl
cmp     VMODE, 7
ja      doset
cmp     dh, LAMA1_LINE
jne     dostd                 ; always go column 10 in title line
mov     dl, 10
cmp     MaxCol, 39
ja      doset
mov     dl, TitleOff
sub     dl, 10                ; offset-20 + 10
jge     doset

```

SUBSTITUTE SHEET

56

```

                xor        dl, dl
                jmp        doset
dostd:
                mov        dl, 4
                cmp        MaxCol, 39
                jbe        doset
                mov        dl, 18H
doset:
                videocall
                pop        si
                pop        di
                pop        dx
                pop        bx
                pop        ax
                ret

```

```

setc_row endp

```

```

;=====

```

```

;               WaitMilli()
;   Waits 1/1000 second ( + or - ??% ) and returns
;

```

```

SETVAL equ 0

```

```

TPort equ 40h

```

```

;

```

```

WaitMilli proc near

```

```

                push        ax
                push        bx
                push        cx
                push        dx
                mov        dx, 60 ; Loop used in case timer is disabled. GAMES, etc!!
                mov        al, SETVAL
                out        43h, al ; read timer 0
                in         al, TPort
                mov        bl, al
                in         al, TPort
                mov        bh, al
miloop1:
                dec        dx
                je         waitout
                mov        al, SETVAL
                out        43h, al
                in         al, TPort
                mov        cl, al
                in         al, TPort
                mov        ch, al
                mov        ax, bx
                sub        ax, cx
                cmp        ax, 1000
                jle        miloop1
waitout:pop     dx
                pop        cx
                pop        bx
                pop        ax
                ret

```

```

WaitMilli endp

```

```

;=====

```

```

; description   Sounds the speaker according to values freq and period
;

```

```

TIMODE      EQU          0B6H

```

```

;

```

```

; LOCATION OF PORTS

```

SUBSTITUTE SHEET

57

```

;
PORT_B EQU 61H ;8255 port B addr
TIMER EQU 40H ;8253 timer port addr
FREQ dw 1000 ; Must always be = 13
PERIOD dw 1
CUTOFF dw TRUE
saveset db 0
;
sndtone PROC near
    MOV AL, TIMODE ;Select TIMER 2,LSB,MSB
    OUT TIMER+3, AL ;Write the timer mode register
    MOV CX, FREQ ; get reques
    MOV DX, 12H ; UPPER NUMERATOR
    MOV AX, 34DEH ; LOWER NUMERATOR
    DIV CX ; the quotient is in AX
    OUT TIMER+2, AL ;Write timer 2 cnt - LSB
    MOV AL,AH ;Write timer 2 cnt - MSB
    OUT TIMER+2, AL ;Write timer 2 cnt - MSB
    IN AL,PORT_B ;Get current setting of port
    MOV saveset, AL ;Save the setting
    OR AL,03 ;Turn speaker on
    OUT PORT_B, AL
    MOV CX, PERIOD ;Set delay count
DELOOP: call WaitMilli
    LOOP DELOOP ;If not, continue beeping speaker
    CMP CUTOFF, FALSE ; turn off tone at end?
    JE TONEXIT ; if not, just quit
    MOV AL, saveset ;Recover value of port
    AND AL, 0FCH ; TURN OFF BOTH GATES
    OUT PORT_B, AL
TONEXIT:RET
sndtone ENDP

```

```

TONEXIT:RET
sndtone ENDP

```

```

;=====

```

```

; name sndbell()
;
; description Sounds the speaker according to values of EQU's used
;
;

```

```

sndbell PROC near
    MOV CUTOFF, FALSE
    MOV PERIOD, 60
    MOV FREQ, 1000
    CALL sndtone
    MOV CUTOFF, TRUE
    MOV FREQ, 1500
    CALL sndtone
    RET

```

```

sndbell ENDP

```

```

;=====

```

```

; name Whoops()
;
; description Sounds the whooping alarm and hangs up the machine.
;
;

```

```

Whoops PROC near
    MOV CUTOFF, FALSE
    MOV PERIOD, 30
PreLp: MOV FREQ, 100
WhoopLp:CALL sndtone
    add FREQ, 100

```

SUBSTITUTE SHEET

58

```

        cmp      FREQ, 2000
ja       PreLp
        mov     ah,1                ; read keyboard
        keybdcall
        jz      WhoopLp            ; have no key waiting, loop
        mov     ah, 0              ; read keyboard
        keybdcall
        cmp     ax, scan_code      ;Is scan code correct?
        jne     WhoopLp
        MOV     CUTOFF, TRUE
        CALL    sndtone
        ret
Whoops ENDP
;=====
; name                Oops()
; description Builds ominously from low to high and returns.
;
Oops PROC near
        MOV     CUTOFF, FALSE
        MOV     PERIOD, 25
        MOV     FREQ, 20
OopLp:   CALL    sndtone
        mov     ah,1                ; read keyboard
        keybdcall
        jz      OopsChk            ; have no key waiting, loop
        mov     ah, 0              ; read keyboard
        keybdcall
        cmp     ax, scan_code      ;Is scan code correct?
        je      OopsOut
OopsChk: add    FREQ, 5
        cmp     FREQ, 3000
        jbe     OopLp
OopsOut:
        pushf
        MOV     CUTOFF, TRUE
        CALL    sndtone
        popf
        ret                        ; if returns 'equal' ( ie. JE ), then hotkey was pressed
Oops ENDP
;=====
; name                Blatt()
; description Sends an insulting failure sound
;
Blatt PROC near
        MOV     CUTOFF, TRUE
        MOV     PERIOD, 20
        mov     al, 50
        mul     entry_count
        mov     cx, ax
Bloop:   cmp     FREQ, 60
        jne     setlo
        mov     FREQ, 103
        jmp     sendit
setlo:   mov     FREQ, 60
sendit:  push    cx
        CALL    sndtone

```

SUBSTITUTE SHEET

59

```

                                pop      cx
                                loop     Bloop
                                ret
Blatt ENDP
;-----
INT_9      PROC      FAR
; (Flags saved by INT)
; Allow interrupts
; Save used register
; are we in LAMA input?
; If not, continue on.
; Get key scan code
; Check if CTL-Break key
; If so, break out
; Get shift status fn
; ignore ANY CTL-ALT combo
; If not, continue on.
; Check if DEL key
; then is our signal
; Restore register
; Process key as normal
;-----
; Reset the keyboard interrupt controller (forget the key stroke)
;-----
StopReset:
    IN      AL,61H
    MOV     AH,AL
    OR      AL,80H
    OUT     61H,AL
    MOV     AL,AH
    JMP     SHORT $+2
; I/O delay for fast AT's
    OUT     61H,AL
; Disable interrupts and
; reset the int controller
    MOV     AL,20H
    OUT     20H,AL
; Allow interrupts
; Restore register
; Go back where we came from
    STI
    POP     AX
    IRET
INT_9      endp
;=====
SCRNSIZE equ 2000
screenbuf dw SCRNSIZE DUP (0)
endbuff   label near
VMode     db 5Eh ; dummy value, must be initialized
VideoSeg dw 0B800h
savecurs dw 33c9h ; dummy value
splitscr db 0
DispPage db 0
;
; line numbers (0 == top) for display
LOGIN_LINE db 5 ; 13
PW_LINE    db 7 ; 14
ENTRY_LINE db 10
LAMA1_LINE db 19 ; 6
LAMA3_LINE db 21
;

```

```

                                60
SaveScrn    proc near                ; screen
            push    ds
            push    es
            push    di
            push    si
            push    bx
            push    cx
            mov     cx, cs
            mov     es, cx
            mov     ds, cx

            call    sndbell
            call    sndbell
            call    sndbell
            call    sndbell
            call    sndbell
            mov     ah, 0Fh                ; read current mode
            videocal
            and     al, 7Fh                ; correct top-bit error
            mov     VMode, al
            mov     DispPage, bh
            dec     ah
            mov     MaxCol, ah
            mov     ah, 3                ; read cursor position
            videocal
            mov     savecurs, dx
            mov     di, offset screenbuf
            mov     al, VMode
            mov     VideoSeg, 0B000h
            cmp     al, 7                ; monochrome adapter?
            je      getcursor
            cmp     al, 0Fh                ; monochrome graphics adapter?
            je      getcursor
            mov     VideoSeg, 0B800h
            cmp     al, 6                ; CGA graphics?
            jle     getcursor
            jmp     pixmode

;
getcursor:
            mov     LOGIN_LINE, 5 ;13
            mov     PW_LINE, 7 ;14
            mov     ENTRY_LINE, 10
            mov     LAMA1_LINE, 19 ;6
            mov     LAMA3_LINE, 21
            mov     splitscr, 0
            cmp     al, 4
            jb      getcur1
            cmp     al, 6
            ja      getcur1
            mov     LOGIN_LINE, 1 ;2
            mov     PW_LINE, 2 ;3
            mov     ENTRY_LINE, 3 ;4
            mov     LAMA1_LINE, 4 ;0
            mov     LAMA3_LINE, 5
            mov     splitscr, 1

getcur1:
            mov     ds, VideoSeg
            mov     si, 0
            cmp     cs:splitscr, 0
            je      singlscr

```

SUBSTITUTE SHEET

61

```

        mov     cx, SCRNSIZE/2
        rep     movsw
        mov     si, 8192
        mov     cx, SCRNSIZE/2
        rep     movsw
        jmp     saveout

singlscr:
        mov     cx, SCRNSIZE
        cmp     CS:MaxCol, 79
        jbe     stdscr
        mov     bx, 25
        mov     dx, word ptr CS:MaxCol
        sub     dx, 79
        shl     dx, 1
        ; always do 25 lines

saverow:mov     cx, 80
        rep     movsw
        dec     bx
        je      saveout
        add     si, dx
        jmp     saverow

stdscr:
saveout:pop     cx
        pop     bx
        pop     si
        pop     di
        pop     es
        pop     ds
        ret

pixmode:
        push    di
        mov     ah, 8
        videocal
        ; BIOS read char/attr
        ; to reset video, MUST HAVE!
        pop     di
        mov     dx, 0
        mov     cx, 281
        ; start at row 0
        ; set column counter to 281

pixloop:
        push    di
        mov     ah, 0Dh
        mov     bh, DispPage
        videocal
        ; save buffer pointer
        ; read pixel
        and     al, 0FH
        mov     bl, al
        dec     cx
        ; save LS nibble
        mov     ah, 0Dh
        videocal
        ; read pixel
        shl     al, 1
        shl     al, 1
        shl     al, 1
        shl     al, 1
        add     al, bl
        pop     di
        ; restore buffer pointer
        stosb
        ; save 2 pixels
        cmp     di, offset endbuff
        ; end of storage space?
        jb     deccol
        inc     cx
        ; This is needed to reset the video into char mode
        mov     al, bl
        ; get LS nibble
        mov     ah, 0Ch
        ; write pixel
        videocal
        mov     PW_LINE, 0
        mov     LOGIN_LINE, 0

```

SUBSTITUTE SHEET

```

                                62
                                ENTRY_LINE, 1
                                saveout
deccol:  mov     cx, 281          ; decr. column
                                pixloop   ; continue if 0
                                mov     dx, 281      ; set column counter to 281
                                inc     dx          ; increment row
                                mov     LOGIN_LINE, 1 ;2
                                mov     PW_LINE, 2   ;3
                                mov     ENTRY_LINE, 3 ;4
                                mov     LAMA1_LINE, 4 ;0
                                mov     LAMA3_LINE, 5
                                mov     splitscr, 1
                                jmp     pixloop
;
SaveScrn  endp
;*****
RestScrn  proc near                ; clear screen
            push     es
            push     di
            push     si
            push     bx
            mov     si, offset screenbuf
            cmp     VMODE, 7
            jbe     CGArest
            mov     dx, 0          ; start at row 0
            mov     cx, 281        ; set column counter to 281
restloop:  mov     bl, [si]
            inc     si
            push     si
            mov     al, bl          ; get LS nibble
            mov     bh, DispPage
            and     al, 0FH
            mov     ah, 0Ch        ; write pixel
            videocall
            dec     cx
            mov     al, bl          ; get LS nibble
            shr     al, 1
            shr     al, 1
            shr     al, 1
            shr     al, 1
            mov     ah, 0Ch        ; write pixel
            videocall
            pop     si
            cmp     si, offset endbuff ; end of storage space?
            jae     restout
            dec     cx              ; decr. column
            jg     restloop        ; continue if = 0
            mov     cx, 281        ; set column counter to 281
            inc     dx              ; increment row
            jmp     restloop
;
CGArest:  mov     es, VideoSeg
            mov     di, 0
            cmp     splitscr, 0
            je     singlrst
            mov     cx, SCRNSIZE/2
            rep     movsw
            mov     di, 8192
            mov     cx, SCRNSIZE/2

```

SUBSTITUTE SHEET


```

                                63
singlrst:    jmp                reststd
            mov                cx, SCRNSIZE
            cmp                MaxCol, 79
            jbe                reststd
            mov                bx, 25
            mov                dx, word ptr MaxCol
            sub                dx, 79
            shl                dx, 1
restrow:
            mov                cx, 80
            rep                movsw
            dec                bx
            jz                 restout
            add                di, dx
            jmp                restrow
reststd:rep  movsw
restout:mov  ah, 2                ; set cursor position
            mov                bh, DispPage
            mov                dx, savecurs
            videocall
            pop                bx
            pop                si
            pop                di
            pop                es
            ret
RestScrln   endp

;*****
ObscureZ dd 0                ; Constant used to obscure code
ObscureV dw 14H              ; Constant used to obscure code
ObscureNMI dw 08H            ; Constant used to obscure code
keycount db 0
editecho db FALSE

editline proc near
            mov                di, offset pw_buf
InputLp:    mov                ah, 0                ; read keyboard
            keybdcall
            cmp                al, backsp            ; backspace?
            jne                testfunc
            cmp                keycount, 0
            je                 InputLp
            dec                keycount
            dec                di
            mov                byte ptr [di], 0
            cmp                editecho, FALSE
            je                 InputLp
            push                cx
            push                si
            push                di
            call                disp_ch
            mov                ax, 0920h            ; write char == space
            mov                bl, LamAttr           ; set attr
            mov                cx, 1                ; write 1 char
            videocall
            pop                di
            pop                si
            pop                cx
            jmp                InputLp

```

SUBSTITUTE SHEET

```

testfunc:                                64
        cmp     al,cr
        jne     addchar
        mov     keycount, 0
        jmp     editcout

addchar:
        cmp     keycount, PWLEN           ; don't over-run space
        je      editcout
        inc     keycount
        mov     byte ptr [di],al
        inc     di
        cmp     editecho, FALSE
        je      editcout
        call    disp_ch

editcout:
        cmp     al,cr
        jne     InputLp
        ret

editline endp

```

```

;*****

```

```

ClearBuf proc near

```

```

; clear buffer for another try    ??? make into macro

```

```

        push    ax
        push    es
        push    ds
        pop     es
        mov     cx,PWLEN + 1           ; initialize loop
        mov     di,offset pw_buf
        xor     al,al
        rep     stosb
        pop     es
        pop     ax
        ret

```

```

ClearBuf endp

```

```

;*****

```

```

DispPW proc near

```

```

        MOV     DH,PW_LINE
        call    clr_line               ; clear line for another try
        MOV     DH,PW_LINE
        call    setc_row
        mov     si, offset pw_pmpt
        call    disp_str
        ret

```

```

DispPW endp

```

```

;*****

```

```

DispEnt

```

```

        proc near
        mov     dh,ENTRY_LINE
        call    setc_row
        mov     si, offset entry_msg
        call    disp_str
        xor     ax,ax
        mov     al,entry_count
        mov     cx,1
        call    bin2dec
        ret

```

```

DispEnt endp

```

```

;*****

```

```

LAMATEST proc near

```

SUBSTITUTE SHEET

65

```

; notes:
; if 0 is read, nextread returns extended ASCII code
; should adapt to host's graphics mode
; NOTE: Must be called with CLI active
;
push    bx
cli                      ;Avoid interrupt right now.
mov     cs:inLAMA,1      ;Set busy flag, to protect our stack.
mov     cs:savss,ss      ;Save caller's stack segment.
mov     cs:savsp,sp      ;save caller's stack pointer.
mov     bx,cs
mov     ss,bx            ;Reset stack to our code segment.
mov     sp,offset cs:topstack ;Start our stack below KbVect addr.
push    cx
push    dx
push    bp
push    si                ;Save all registers.
push    di
push    ds
push    es
push    cs
pop     es
mov     cx,CS:VectLenVar ; save all vectors now, restore later
lds     si,CS:ObscureZ
mov     di,offset CS:SaveVects
rep     movsw
push    cs
pop     ds
call    StealVects       ; assume CLI active now, take over vectors
sti
call    SaveScrn ; save screen, as much as 4k will hold
; put up screen
DoScreen:
call    clrscr           ; clear screen area
mov     entry_count,0
cmp     VMODE,7          ; not enough room in hires, skip
ja      main_loop
mov     dh,LAMA1_LINE
call    setc_row
mov     si,offset TitleStr
cmp     MaxCol,39
ja      dtitle
xor     ah,ah
mov     al,TitleOff
add     si,ax
dtitle: call disp_str
mov     dh,LAMA3_LINE
call    setc_row
mov     si,offset lama3
call    disp_str
main_loop:
cmp     entry_count,0    ; display count of entry attempts
je      put_login
call    DispEnt
; prompt for login
put_login:
cmp     AccessToo,FALSE
je      put_prompt
mov     dh,LOGIN_LINE    ; clear line for another try
call    clr_line
MOV     DH,LOGIN_LINE

```

SUBSTITUTE SHEET

```

                                66
        call setc_row
        mov     si, offset login_pmpt
        call disp_str
; retrieve login
        MOV     editecho, 1
        call    ClearBuf
        call    editline
        call    HashPass
; retrieve password without echo
put_prompt:                                ; prompt for password
        push    ax
        push    dx
        call    DispPW
        call    ClearBuf
        MOV     editecho, FALSE
        call    editline
        pop     dx
        pop     ax
; don't bother decrypting password if login is invalid
        cmp     AccessToo, FALSE
        je      dohash
        cmp     ax, U1Access1
        jne     bad_pw
        cmp     dx, U1Access2
        jne     bad_pw
dohash:   call    HashPass
        call    ClearBuf
        cmp     ax, U1Password1
        jne     bad_pw
        cmp     dx, U1Password2
        je      good_pw
; keep count of bad tries
bad_pw:   inc     entry_count
        call    Blatt
        mov     al, maxatts
        cmp     al, 0
        je      main_loop                ; try again
        cmp     al, entry_count
        ja      main_loop                ; try again
        call    clrscr                    ; clear screen area
        call    Oops
        je      recover
        call    Whoops                    ; alarm and hang up
recover:  jmp     DoScreen
; NOW WE MUST RESTORE THE SCREEN
good_pw:  call    RestScr                  ; restore screen
        cli
        mov     cx, CS:VectLenVar
        mov     si, offset SaveVects
        les     di, ObscureZ2
        rep     movsw
        mov     di, ObscureV
        mov     ax, SavePSCrn            ; replace dummy printscreen vect.
        stosw                                     ; mov     0:[14h], ax
        mov     ax, SavePSCrn + 2
        stosw                                     ; mov     0:[16h], bx
        mov     di, 20h                    ; point to INT 8 - Timer
        mov     ax, SaveTimer
        stosw                                     ; mov     0:[20h], ax

```

```

                                67
                                mov     ax, SaveTimer + 2
                                stosw
                                mov     di, 70h
                                mov     ax, SaveTic
                                stosw
                                mov     ax, SaveTic + 2
                                stosw
                                pop     es
                                pop     ds
                                pop     di
                                pop     si
                                pop     bp
                                pop     dx
                                pop     cx
                                mov     ss,cs:savss
                                mov     sp,cs:savsp
                                sti
                                mov     cs:inLAMA, FALSE
                                pop     bx
                                ret
LAMATEST endp

```

page

```

*****
; bin2dec: binary to decimal conversion. Displays the contents of AX
; on the screen as a signed decimal number. Finds the rightmost digit by
; division, repeating until all found. The CH register contains the minimum
; # of digits to be displayed. AX is destroyed by this call.
bin2dec proc near
    push    bx
    push    cx
    push    dx
    mov     cl,0
    mov     bx,10
; check for negative number. If negative, make number positive.
    or      ax,ax
    jnl     more_hex
    neg     ax
    push    ax
    mov     al,'-'
    call    disp_ch
; main division loop - get decimal digit until no more remain
more_hex:
    xor     dx,dx
    div     bx
    push    dx
    inc     cl
    or      ax,ax
    jnz     more_hex
; main digit print loop - reverse order
    sub     ch,cl
    jle     morechr
    xor     dx,dx
morezero:
    push    dx
    inc     cl
    dec     ch
    jnz     morezero

```

SUBSTITUTE SHEET

68

```

morechr:
    pop     dx                ; restore last digit
    add     dl,30h           ; converts to ASCII
    push    ax
    mov     al,dl
    call    disp_ch
    pop     ax
    dec     cl                ; digits countr == 1
    jnz     morechr         ; continue if more
    pop     dx
    pop     cx
    pop     bx
    ret

bin2dec    endp
;*****
xxx        db      'MaxCol='
MaxCol     db      0        ; maximum row for the current video mode.
MaxCol2    db      0        ; allow use as word
ObscureZ2  dd      0        ; Constant used to obscure code
entry_count db      0        ; # entry attempts

PWLEN      equ      20      ; size of password
pw_buf     db      (PWLEN+1) dup (0) ;typed password buffer (extra space for $)

MsgKey     db      'MString=' ; let MString be customized with 60 chars
TitleOff   db      20      ; offset of text from start of string
; counter- |012345678901234567890123456789012345678901234567890123456789|
;TitleStr db '
;          LAMALOCK+
TitleStr db '
;          Anchor Pad Plus
db eom

WPFstring  db      'Write-protect error. Drive A: is not protected.',7,0
login_pmpt db      'Access Code:',eom
pw_pmpt    db      'Password: ',eom
lama3      db      ' AUTHORIZED PERSONNEL ONLY',cr,lf,eom
entry_msg  db      ' Entry Attempts: ',eom

;*****
stackspac  db      150 DUP (0)
topstack   dw      0
VectLength equ      40H
InitVects  dw      VectLength DUP (0)
SaveVects  dw      VectLength DUP (0)
SavePScrn  dw      2 DUP (0);Holds old Interrupt 8 BIOS vector.
TimerVect  dw      2 DUP (0);Holds old Interrupt 8 BIOS vector.
SaveTimer  dw      2 DUP (0);Holds old Interrupt 8 BIOS vector.
TicVect    dw      2 DUP (0);Holds old Interrupt 1ch BIOS vector.
SaveTic    dw      2 DUP (0);Holds old Interrupt 1ch BIOS vector.
VidVect    dw      2 DUP (0);Holds old Interrupt 10h BIOS vector.
OrgINT9    dw      2 DUP (0);Holds old Interrupt 9 BIOS vector.
savss      dw      ?        ;Holds caller's stack segment
savsp      dw      ?        ;Holds caller's stack pointer
VectLenVar dw      VectLength
attrkey     db      'attribute='
LamAttr     db      07H
maxakey     db      'maxatt=' ; keyword for maximum attempts
maxatts     db      3        ; alarms after three attempts
;alarmkey    db      'alarm='
;alarmlevel  db      0
writetest   db      FALSE    ; assume don't test drive A: write protect

```

SUBSTITUTE SHEET

```

                                69
stealv      db      0          ; Level of security, vector stealing. 2 = = max
AccessToo   db      TRUE       ; BOTH Access Code and Password
;*****
;
;page

;int23      dw      0,0
;int1b      dw      0,0

ND_Read:    ; function 5 = Non-Destructive Read
Inp_Flush:  ; function 7 = Flush Input Buffers
Outp_Stat:  ; function 10 = Output Status

IOCTL_Rd proc      near          ; function 3 = I/O Control Read
    cli
    call LAMATEST          ;Go to LAMA.
    xor      ax,ax
    ret
IOCTL_Rd endp

;
EndByte     db 5Ah          ; last byte, not XORed in sweep decode
;*****
doscall     macro
    int      21h            ;; call MS-DOS function
endm

;
LamaInit    proc      near          ; function 0 = Initialize Driver
    push     ds
    push     es              ; save address of Request Header
    push     di
    push     ax

;
    push     cs
    pop      ds
    xor      ax, ax
    mov      es, ax
    mov      ax, es:[40h]    ; Save original BIOS video vector
    mov      VidVect,ax     ; BEFORE calling disp_str!
    mov      ax, es:[42h]
    mov      VidVect+2,ax
    mov      writetest, FALSE
    cmp
    je

readloop:
    mov      ax, 0201H      ; BIOS diskette read.
    push     cs
    pop      es
    mov      bx, offset screenbuf
    mov      cx, 0001H
    mov      dx, 0000H
    diskcall
    jb      readloop        ; loop until read passes
    mov      ax, 0301H      ; BIOS diskette write back
    diskcall
    and      ah, 02H        ; write protect error
    jnz      LTest
    mov      si, offset WPFstring
    call disp_str
    jmp      WPfail
WPfail:     jmp      WPfail
; XXXXX Proper DOS calls cannot be made during INIT, must cheat. XXXX
LTest:      xor      ax, ax
            mov      es, ax

```

SUBSTITUTE SHEET

70

```

        mov     ax, es:[24h]      ; Save entry hardware keyboard vector
        mov     OrgInt9, ax
        mov     ax, es:[26h]
        mov     OrgInt9 + 2, ax
    IF ZERORAM
        mov     di, 24h          ; point to 0:24h
        mov     ax, offset cs:INT_9 ; use local to defeat ctl-alt-del
        stosw
        mov     ax, cs
        stosw
    ENDIF
    push     cs
    pop      es
    mov     cx, VectLenVar
    lds     si, ObscureZ
    mov     di, offset InitVects
    rep     movsw
    push     cs
    pop      ds
    xor     ax, ax
    mov     es, ax
    mov     ax, es:[58h]          ; Save original BIOS keyboard vector
    mov     KbVect, ax
    mov     ax, es:[5Ah]
    mov     KbVect + 2, ax
    mov     ax, es:[20h]          ; Save original BIOS timer vector
    mov     TimerVect, ax
    mov     ax, es:[22h]
    mov     TimerVect + 2, ax
    mov     ax, es:[70h]          ; Save original BIOS timer vector
    mov     TicVect, ax
    mov     ax, es:[72h]
    mov     TicVect + 2, ax
    cli
    IF ZERORAM
        mov     ax, offset KeyInt16
        mov     es:[58h], ax
        mov     ax, cs
        mov     es:[5Ah], ax
    ENDIF
    call     LAMATEST             ;Go to LAMA.
    ;
    IF ZERORAM
        mov     ax, cs
        mov     es, ax
        mov     ax, offset CS:Media_Chk
        mov     di, offset CS:Dispatch
        mov     cx, 17             ; 17 functions in the table
        rep     stosw
    ENDIF
    ; set first usable memory address as start of Lamalnit to reclaim its memory
    pop     ax
    pop     di                     ; restore Request Header address
    pop     es
    pop     ds
    ;
    IF ZERORAM
        mov     word ptr es:[di.address], offset CS:Header
        mov     word ptr es:[di.address], offset CS:Build_Bpb
    ELSE

```

SUBSTITUTE SHEET

71

```

    ENDIF      mov      word ptr es:[di.address],offset CS:LamaIntr
                mov      word ptr es:[di.address + 2],cs
                xor      ax,ax          ; return status
                ret
LamaIntr      endp
;
Intr          endp
LAMA          endp
code          ends
end
```

SUBSTITUTE SHEET

WE CLAIM:

1. A computer access control system for use on a personal computer having an external boot drive device to deter unauthorized access to the computer data and programs, comprising:

physical means for blocking removal or insertion of a computer program code containing device from the computer boot drive device, and

computer access control program means for enabling and disabling a personal computer upon input demonstrating authority to access the computer.

2. The control system according to claim 1 where the computer access control program comprises a priority device driver which is first addressed during bootup.

3. A computer access control system according to claim 1 including a program featuring a bi-level entry access input requirement where an access code and password must both be properly inputted to demonstrate authority and obtain control of the computer.

4. A computer access control system according to claim 1 comprising means for immediately disabling a computer upon single input without destroying any data.

5. A computer access control system according to claim 2 where the program includes means for selecting the installation configuration;

a plurality of security levels offering differing degrees of security;

means for selecting a desired one of said security levels; and

means for selecting an appropriate alarm level.

6. A personal computer access control system for a computer with a default boot drive, comprising:

a) computer software featuring a program including program means to override system initialization during boot up to require entry of an access code and a password before initialization of the operating system, program means to permit interruption of computer operations by entry of a user defined key and restoration upon entry of the password, and

b) means for maintaining said software in the default boot drive of the computer, said means being positionable between a drive blocking mode which locks said software in said boot drive and an unblocking mode which permits removal of said software from said boot drive.

7. A system according to claim 6 where said program requires only approximately a 7.5K byte file on the disk.

8. A system according to claim 6 where said program further comprises a security driver file to modify the system configuration program of the computer.

9. A system according to claim 8 where said program further comprises at least three different security levels and means to select a particular level of security desired.

10. A method for controlling access to a personal computer with an external data drive, comprising the following steps:

- inserting a computer program containing device into the external data drive,
- blocking the external data drive with a lockable, blocking device capable of physically locking the program into the computer,
- installing a computer program including a device driver that is first addressed by the computer during boot up,
- selecting a proper access code and password,
- rebooting the computer, and
- entering a selected access code and password.

11. A method according to claim 10 further comprising the steps of selecting the desired level of security, selecting a user defined hotkey for immediate interrupting of computer operations, and selecting the security screen color.

12. A method for personal computer access control, comprising the steps of:

- a) providing computer software on a medium removable from the computer including a system configuration drive-command to supersede the computer's disk operating system's configuration to prevent access to the disk operating system without demonstrating authority to so access the computer,

b) locking the computer software into the computer, and

c) inputting an access code and a password to demonstrate authority to access the computer.

1/1

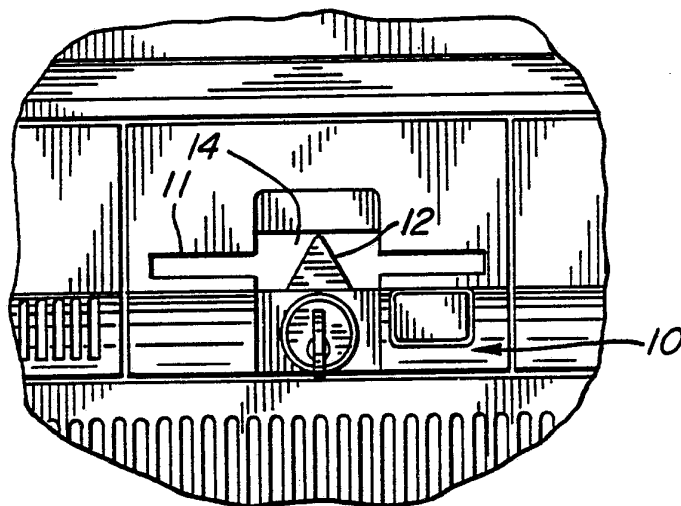
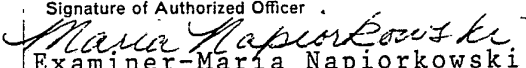


FIG. 1
SUBSTITUTE SHEET

INTERNATIONAL SEARCH REPORT

International Application No. PCT/US89/04913

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) ⁶		
According to International Patent Classification (IPC) or to both National Classification and IPC INT. Cl.(5): G06F 13/14 U.S.Cl. 364/200		
II. FIELDS SEARCHED		
Minimum Documentation Searched ⁷		
Classification System	Classification Symbols	
U.S.	364/200,900; 70/14,57,58	
Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched ⁸		
III. DOCUMENTS CONSIDERED TO BE RELEVANT ⁹		
Category *	Citation of Document, ¹¹ with indication, where appropriate, of the relevant passages ¹²	Relevant to Claim No. ¹³
Y, E P	US,A, 4,870,591 (CICCIARELLI et al.) 26 SEPTEMBER 1989, see column 11, line 45- column 13, line 19.	5,9,11
Y	US,A, 4,719,566 (KELLEY) 12 JANUARY 1988, see column 1, line 65 - column 2, line 45.	1,6,10,12
Y	US,A, 4,685,312 (LAKOSKI et al.) 11 AUGUST 1987, see the entire document.	1,6,10,12
Y	US,A, 4,652,990 (PAILEN et al.) 24 MARCH 1987, see the entire document.	1-12
Y	US,A, 4,621,321 (BOEBERT et al.) 04 NOVEMBER 1986, see the entire document.	1-12
Y	US,A, 4,439,830 (CHUEH) 27 MARCH 1984, see the entire document.	1-12
Y	US,A, 4,218,738 (MATYAS et al.) 19 AUGUST 1980, see the entire document.	1-12
<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>* Special categories of cited documents: ¹⁰</p> <p>"A" document defining the general state of the art which is not considered to be of particular relevance</p> <p>"E" earlier document but published on or after the international filing date</p> <p>"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)</p> <p>"O" document referring to an oral disclosure, use, exhibition or other means</p> <p>"P" document published prior to the international filing date but later than the priority date claimed</p> </div> <div style="width: 45%;"> <p>"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention</p> <p>"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step</p> <p>"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.</p> <p>"&" document member of the same patent family</p> </div> </div>		
IV. CERTIFICATION		
Date of the Actual Completion of the International Search		Date of Mailing of this International Search Report
24 JANUARY 1990		23 FEB 1990
International Searching Authority		Signature of Authorized Officer
ISA/US		 Examiner-Maria Napiorkowski

III. DOCUMENTS CONSIDERED TO BE RELEVANT (CONTINUED FROM THE SECOND SHEET)		
Category *	Citation of Document, with indication, where appropriate, of the relevant passages	Relevant to Claim No
Y	APS Text Search and Retrieval-Classroom Manual, Planning Research Corporation, Virginia, Revision 11/87, pp. 1-14 and 1-26 through 1-36.	1-12
A	US,A, 4,655,057 (DERMAN) 07 APRIL 1987, see the entire document.	1,6,10,12
A	US,A, 4,640,106 (DERMAN) 03 FEBRUARY 1987, see the entire document.	1,6,10,12
A	US,A, 4,527,405 (RENICK et al.) 09 JULY 1985, see the entire document.	1,6,10,12
A	US,A, 4,131,001 (GOTTO) 26 DECEMBER 1978, see the entire document.	1,6,10,12