DIDS-92-16
UCRL-MA-103440, Rev. 3

Security Profile Inspector for the UNIX Operating System (SPI/UNIX)

User's Guide

for SPI/UNIX version 2.1

Tony Bartoletti

July 1992

Lawrence Livermore National Laboratory

SPI is sponsored by:
US Air Force Cryptologic Support Center
US Department of Energy

Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

ACKNOWLEDGEMENT

The work reported in the following pages has been sponsored by several organizations. We appreciate the support, advice, and technical direction of:

U. S. Department of Energy
Office of Safeguards and Security
SA-123/GTN
Washington DC

U. S. Air Force
Cryptologic Support Center
AFCSC/SRE
San Antonio, TX

U. S. Department of Defense
Defense Intelligence Agency
ATTN: DSB-4B
Washington, DC

Contents

Figures

Preface

Audience

The audience for this book is a technical reader, for example a UNIX system administrator, who is interested in learning how to use the *System Profile Inspector for UNIX (SPI/UNIX).*

How This Book Is Organized

This book supersedes UCRL-MA-103440, Rev. 2. It contains four chapters, six appendices, a glossary of terms, an index, and a comment form.

Chapter 1, Introduction: An introduction to the SPI/UNIX application that briefly describes the various tools available to assist system managers and computer security officers in maintaining the security of their systems.

Chapter 2, SPI Main Menu Selection: A presentation of the SPI/UNIX main menu.

Chapter 3, SECURITY Functions: A description of the security checks provided by the SECURITY tools.

Chapter 4, ADMINISTRATIVE Functions: A description of various tasks, such as looking at reports, scheduling automatic functions, and managing runtime parameters.

Appendix A, Installation Guide: This appendix contains instructions for installing the SPI/UNIX software onto your system.

Appendix B, Distribution Directory Structure: A list of pre-installation files and directories in the SPI/UNIX package.

Appendix C, Installed Directory Structure: A list of files and subdirectories in the SPI/UNIX directory remaining after installation.

Appendix D, Sample Parameter Files: A list of sample parameter files for the SPI/UNIX security functions.

Appendix E, Sample Output Reports: Sample reports, and their descriptions, generated from running the SPI/UNIX security functions.

Appendix F, ACT Rules: This appendix contains the default rulebase for the ACT utility.

What's New in Version 2.1

SPI/UNIX 2.1 has three new additions. Two new tools, ACT and BIT, and testing for C2 shadow passwords have been added.

Access Control Test (ACT) is a rule-based expert system designed to assess sequential dependencies in UNIX computer access control mechanisms.

Binary Inspector Tool (BIT) identifies known vulnerable system utilities.

The SPI/UNIX Password Inspector now detects the presence of the C2 shadow password file, if it exists. The Password Inspector also automates the processing of these passwords.

Further Information

SPI, or information about it, may be obtained by contacting the SPI/UNIX Team Leader:

ATTN: *Tony Bartoletti* c/o SPI/UNIX Development
Computer Security Research Center, L-419
LAWRENCE LIVERMORE NATIONAL LABORATORY
P.0. Box 808
Livermore, CA 94551-9900
TEL: (510) 422-3881
FTS: 422-3881
EMAIL: spi@irbis.llnl.gov or azb@cheetah.llnl.gov

If you are affiliated with the Air Force, contact:
ATTN: *Lieutenant Dan Teal*
AFCSC/SREC
San Antonio, Texas, 78243-5000
TEL: (512) 997-3134
EMAIL: teal@dockmaster.ncsc.mil

Conventions Used in This Book

This following conventions are used in this book:

- **`bold terminal font`**

  This font indicates text or code typed at the keyboard during an interactive session with the application.

- `terminal font`

  This font indicates information displayed by the computer during interactive sessions. It is also used in code examples and text to indicate filenames, command names, and error names.

- *italic font*

  This font is used in code examples and text to indicate user-specified parameters for insertion into programs or command lines. It is also used to introduce new terms or phrases the first time they are used in the text.

- gray boxes

  > Examples of interactive dialog with SPI are shown in gray boxes.

- Return, Tab, Delete

  Keyboard keys are given an initial capital letter. For example, "To select the 'Run Full Password Profile' option, type **2** and press the Return key."

1

Introduction

1.1 Purpose

This manual explains how to use the *Security Profile Inspector for the UNIX operating system (SPI/UNIX)* to detect potential security problems. This manual provides information for both novice and advanced SPI/UNIX users.

1.2 What Is SPI/UNIX?

SPI/UNIX is a software security package for computers running the UNIX[1] operating system. Its purpose is to assist system managers and computer security officers in providing and monitoring the security of a computer system. SPI inspects various aspects of the computer system and reports on inconsistencies or insecure features. SPI also periodically checks certain security features and produces reports indicating any violated security policies. SPI Version 2.1 provides six major security inspections:

- ❒ The *Quick System Profile (QSP)*,
- ❒ The *Access Control Test (ACT)*,
- ❒ The *Password Security Inspector (PSI)*,
- ❒ The *Binary Inspector Tool (BIT)*,
- ❒ The *File Inode Change Detector (FCD)*, and
- ❒ The *File Data Change Detector (DCD)*

The first four of these are Vulnerability Detection tools, while the last two are Intrusion (Change) Detection tools. Computer system vulnerabilities are weaknesses in a system's defenses against intruders. Examples of vulnerabilities include: improper world permissions, easy-to-guess or outdated passwords, or accounts that no longer have users. SPI helps to detect these vulnerabilities so that you can eliminate or strengthen them.

[1] The following UNIX systems are currently supported: Sun 3/80 running SunOS 4.x, Sun 4 (Sparcstation) running SunOS 4.x, DEC running Ultrix 4.1, and Cray running Unicos 6.1.

Computer intrusions are detrimental changes to a system. SPI helps to protect against these changes by taking an initial "snapshot" of the system configuration, and then using this as a baseline for comparison when you need to check what elements have been affected by an intrusion. Intrusions are things such as: a changed CRC value on a file that should have been left alone, unexpectedly changed device or file permissions or ownerships, and corrupted log data.

SPI is undergoing constant improvement and input from users is always welcome.

Ease of Use

SPI was designed to be easy to use. All of SPI's security functions, as well as other administrative functions, are accessible through a menu-driven user interface. This interface also checks that the data you enter is within expected limits. Defaults are set for most functions. The parameters of each inspection can be modified by entering or changing values on easy-to-use forms. SPI's user interface also provides online help. For more experienced users, all of SPI's security functions are available as programs which you can run from the UNIX shell.

SEE ALSO the accompanying *UNIX man pages*. The online location of these pages is shown in *Appendix C—Installed Directory Structure.*

Portability

SPI is portable to many different versions of UNIX. While there is no guarantee that it will run on all UNIX platforms, we have tested it on standard AT&T System V and Berkeley 4.x, as well as SunOS operating systems. We have included an extensive configuration script (a derivative of Larry Wall's "Metaconfig") which will try to configure SPI to as many different versions of UNIX as possible.

Available Formats

SPI/UNIX Version 2.1 is currently available from the Computer Security Research Center at Lawrence Livermore National Laboratory. The Department of Energy and other Government sites can obtain SPI on 9-track magnetic tape or 1/4-in. cartridge tape (Sun format). Others can obtain SPI via Internet `ftp` access. We prefer to set up a redistribution contact point for larger sites. These contact points will be responsible for redistribution within an organization. Please indicate if you will be redistributing this software for your organization.

1.3 SPI/UNIX Operational Directories

The following is a list of directories required for proper operation:

Directory	Description
`parameters`	Location of the parameter files for the SPI security functions.
`database`	Location of the database files for the SPI security functions.
`results`	Location of the output reports generated by the SPI security functions.
`screen`	Location of the form descriptions for the user interface and text for the online HELP windows.
`man`	Location of the UNIX-style online *manual (man)* pages for command-line operation of the SPI security functions.

1.4 SPI/UNIX Installation Directories

The following is a list of directories not required after installation:

Directory	Description
`quisp`	Source code to the Quick System Profile Inspector.
`acid`	Source code to the Access Control Test.
`pwd`	Contains a database of dubious passwords.
`bitsrc`	Source code to the Binary Inspector Tool.
`sum`	Source to the File Data Change Detector.
`inode`	Source to the Fast Inode Change Detector.
`pass`	Source to the Password Security Inspector.
`ui`	Source to the user interface.
`doc`	Source of manual pages.

2

SPI Main Menu Selection

2.1 Menu Selections

The SPI main menu is divided into SECURITY and ADMINISTRATIVE functions, respectively. The six SECURITY functions, also called INSPECTORS, are:

(1) The Quick System Profile (QSP)
(2) The Access Control Test (ACT)
(3) The Password Security Inspector (PSI)
(4) The Binary Inspector Tool (BIT)
(5) The File Inode Change Detection (FCD)
(6) The File Data Change Detection (DCD)

Figure 2-1 SPI/UNIX main menu

```
SECURITY PROFILE INSPECTOR / UNIX                    VERSION 2.1

   SECURITY FUNCTIONS                  ADMINISTRATIVE FUNCTIONS

   1. Run Quick System Profile         A. Active Process Status
   2. Run Access Control Test          B. View Output Reports
   3. Run Full Password Profile        C. Schedule Run Times
   4. Run Binary Inspector Tool        D. Parameter Management
   5. Run File Inode Change Detector   X. Exit
   6. Run File Data Change Detector



     Enter selection number or letter:

                                Help  <?>
```

To run a SECURITY function, enter its number and press Return. The functions run in the background, allowing you to continue to perform other tasks in the SPI application.

Each function runs according to its ACTIVE parameter file. Default parameter files have been supplied for each function. You may generate alternate parameter files for each SECURITY function.

There are four ADMINISTRATIVE functions:
(A) Active Process Status
(B) View Output Reports
(C) Schedule Run Times
(D) Manage Runtime Parameters
Details about each SECURITY and ADMINISTRATIVE function are given below.

2.2 About Getting HELP

To receive online HELP regarding any menu or form-field, simply type a question mark **?**. To receive HELP regarding a menu selection, type the selection value first, then type a question mark.

For further information about each SECURITY function you may enter its number, followed by a question mark **?**, from the SPI main menu.

Example To learn more about the Quick System Profile, type **1?** from the SPI main menu.

For further information about each ADMINISTRATIVE function, enter its letter, followed by **?**.

Example To learn more about Managing Runtime Parameters, type **D?**.

In general, **?** will provide online HELP at any point in the SPI application.

3

SECURITY Functions

3.1 The Quick System Profile (QSP)

The *Quick System Profile (QSP)* script will conduct the security checks outlined below. QSP also may be run on the command line.

SEE ALSO the accompanying *UNIX man pages*. The online location of these pages is shown in *Appendix C—Installed Directory Structure.*

The subordinate functions are:

`check_access`

`Check_access` checks the permissions of all files and directories listed in the file `access.list`, and prints warning messages. It uses the `is_writeable` program.

`check_cron`

`Check_cron` checks pathnames and files inside the `cron` files for writability.

`check_devices`

`Check_devices` checks the permissions of all devices listed in the file `/etc/fstab`, and flags them if they are world readable or writable. It also checks for unrestricted NFS mountings.

`check_ftp`

If you have not selected `ANONYMOUS_FTP`, this script only ensures that your `ftpusers` file contains no root accounts. (ftps which allow `chmod` must NOT have home directory owned by ftp.) With `ANONYMOUS_FTP` selected, this ensures: User ftp exists in the password file, root, or all root equivalents, is/are in the `ftpusers` file, home directory for ftp exists, and is not '/', the `~ftp/etc/{passwd|group}` files are not the same as the real ones, and that various critical files/directories exist with correct permissions and owners.

check_group

Check_group checks the file /etc/group for incorrect number of fields, duplicate groups, non-alphanumeric group names, and non-numeric group IDs.

check_misc

Check_misc checks to see if tftp and rexd are enabled, if the uudecode alias is in the mail alias file and not commented out, and if uudecode can create a SUID file.

check_rc_files

Check_rc_files checks pathnames and files inside the shell script files /etc/rc* for writability.

check_root

Check_root checks pathnames inside root startup files (/.login, /.cshrc, /.profile) for writability, improper umask settings (world writable), non-root entries in /.rhosts, and to ensure that root is in /etc/ftpusers. Also check for a single "+" in /etc/hosts.equiv (world is trusted), and that /bin, /etc and certain key files are root owned.

check_suid

Check_suid reports any world-writable SUID binaries, and ALL SUID shell scripts.

check_users

Check_users checks all home directories listed in /etc/passwd to ensure that the directories exist, are in fact directories, and are not world-writable. This also checks each user's configuration files for world-writability or world-execute permissions. The files checked are: rhosts, profile, login, logout, cshrc, bashrc, kshrc, tcshrc, netrc, forward, dbxinit, distfile, exrc, and emacsrc.

3.1.1 Options

See *Managing Runtime Parameters* for more details.

Check the ANONYMOUS ftp configuration.

Check for GROUP access to devices.

Scan the file system for SUID scripts and writable binaries.

⇨Note: `QSP` requires the binaries `is_able`, `is_writable`, and `check_users` be present in the installation directory.

3.1.2 Acknowledgment

The QSP suite of security checks were derived and adapted from COPS, a security package largely developed by Dan Farmer, formerly of Carnegie Mellon University.

SEE ALSO the supplied UNIX manual pages: *spi(8-spi)*. The 8 means that the text is in chapter 8. The online location of these pages is shown in *Appendix C—Installed Directory Structure*.

3.2 The Access Control Test (ACT)

The *Access Control Test (ACT)* inspector examines sequential dependencies in UNIX access control files. You may also run ACT on the command line.

ACT is a rule-based, goal-seeking system designed to assess sequential dependencies in computer access control mechanisms. Loosely based upon Bob Baldwin's "Kuang" tool, this utility applies an external rulebase particular to UNIX access controls. An example of a sequential dependency:

Example If "A" is vulnerable to attack by "B", and "B" is vulnerable to attack by "C", then "A" is vulnerable to attack by "C".

The behavior of ACT depends on three fundamental parameters:

- ❒ *TARGET* the target goal being attempted. This target goal is usually to gain root (superuser) access.
- ❒ *ASSUMED* the assumed initial condition.
- ❒ *RULEBASE* the list of rules defining goal transition.

ACT uses the transition rules specified in the supplied *RULEBASE* to discover a way to gain the *TARGET* goal. It begins with the *ASSUMED* initial condition. An example is given in section E.2, *Access Control Test Reports*.

You initialize ACT by giving it a *TARGET* goal (e.g., become root) and an *ASSUMED* initial state. These *TARGET* goals and *ASSUMED* initial states are described in *Access Control Test Runtime Parameters* of Chapter 4.

The initial state is typically U=WORLD, which assumes that you are an ordinary, unspecified user on the system being tested, and that you can modify only those files which are world-writable.

The basic goal types are:

- Write FILE — Gain the ability to modify a specified FILE.
- Become USER — Gain the ability to execute commands with the authority of a given USER.
- Member GROUP — Gain the ability to execute commands with the authority of a given GROUP.

Each goal has associated conditional subgoals to be added and similarly assessed.

An access-graph (directed, acyclic) is internally generated for the selected target goal. When you run ACT from the command line (not from the menu) you can invoke "verbose" mode. This generates a printout "representing" the directed graph. An example of this printout is shown in the next figure.

Figure 2-2 Map of system vulnerabilities generated by ACT

```
ACT:  Access Control Test Report                                  DATE=061892

Assumed Initial Condition: U=WORLD
Attempting Target Goal: UID=0 . . .

***  U=WORLD * (ASSUMED) FR=/user/bob/.login * trojan U=bob * grant UID=8765 *
member G=staff * member GID=10 * group_modify FR=/.login * trojan U=root * grant
UID=0 * TARGET

UID=0  OPEN
    U=root   OPEN
        FR=/etc/passwd   SAFE
            FW=/etc      SAFE
        FR=/usr/lib/aliases    SAFE
            FW=/usr/lib  SAFE
                UID=3    SAFE
                    U=bin      SAFE
                        FR=/etc/passwd     SAFE
                            FW=/etc  SAFE
                    FR=/etc/hosts.equiv    SAFE
                        FW=/etc      SAFE
        FW=/.rhosts      SAFE
        FR=/.login       OPEN
            GID=10       OPEN
                G=staff  OPEN
                    UID=2345   SAFE
                        U=fred       SAFE
                            FR=/etc/passwd       SAFE
                                FW=/etc    SAFE
                            FR=/usr/lib/aliases  SAFE
                                FW=/usr/lib      SAFE
                                    UID=3  SAFE
                                        U=bin    SAFE
                                            FR=/etc/passwd   SAFE
                                                FW=/etc      SAFE
                                        FR=/etc/hosts.equiv  SAFE
                                            FW=/etc    SAFE
                                 • • •
```

The list of vulnerabilities shown in Fig. 2.2 is just a sample and incomplete. The man page for ACT explains how to generate a list like this.

SEE ALSO the supplied UNIX manual pages: *act (8-act).* The online location of these pages is shown in *Appendix C—Installed Directory Structure*.

The database of rule prototypes is scanned for entries appropriate to the indicated goal. Each rule-prototype is expanded to a complete rule (or list of rules) based upon the particular rule-prototype and system-specific information. The completed rules contain conditions which must be met to finalize the decision to apply the given rule. Rules that pass are then used to generate subgoals for the goal in question. These subgoals are, in turn, recursively examined for further subgoal generation.

Now each of these new subgoals is examined, and the RULES database is searched for rule-prototypes matching the class of new subgoals to be treated.

Example

Suppose the target goal is to write (modify) a file `FFF` owned by root. Suppose this file belongs to group "operator". One of the rule-prototypes states "To modify a file, try to become a MEMBER of the group that it belongs to, if the file is WRITABLE by its group."

Now based upon the facts of the specific target goal, this rule-prototype becomes "To modify file `FFF`, try to become a MEMBER of the group 'operator', assuming that `FFF` is writable by the group 'operator'.

So we examine the file's permission bits (inode information) to see if it is indeed writable by its group. If it is, we must try to become a member of "operator", so we look in the group file to see who is already a member of group "operator". It turns out that "Bob, Carol, Ted, and Alice" are members of the group "operator". Thus the rule-prototype expands into the following FOUR rules:

1. To modify file `FFF`, try to become "bob".
2. To modify file `FFF`, try to become "carol".
3. To modify file `FFF`, try to become "ted".
4. To modify file `FFF`, try to become "alice".

For further information, see Section E.2, *Access Control Test Reports* and Appendix F, *ACT Rules*.

3.3 The Password Security Inspector (PSI)

The Password Security Inspector checks the password file for easily guessed passwords. (In UNIX, the password file can usually be found in `/etc/passwd`.) The password file contains information such as account names, passwords, project names, and a comment field (`GCOS` field) that usually contains information about the user. SPI checks passwords in the password file against various dictionaries and certain algorithmic permutations to find passwords that are easy-to-guess or find, using trial-and-error methods. The system manager can choose which dictionaries and permutations to use. The checks include the following:

- ❒ comparing 1-, 2-, or 3-letter combinations of alphabetic and numeric permutations against users' passwords to see if any match.
- ❒ comparing users' passwords against passwords derived from the user's account name or `GCOS` field of the password file.
- ❒ comparing users' passwords with entries in three separate dictionaries: common English words, local jargon, and trivial words (e.g. names of characters in books or names of cars).
- ❒ additionally, the Password Inspector checks for accounts without a password, or with a password that is too old. Old passwords could indicate an inactive account. Passwords should be changed at regular intervals to prevent a compromised password from giving unlimited access. SPI has a mechanism for finding old passwords involving preparing a log file and subsequent checks using this log file. For more information on setting up and using a password log file refer to the section *Managing Runtime Parameters*.
- ❒ The Password Inspector will automatically detect the existence of a shadow password file, and will default to testing those accounts which are common to both the shadow file and the standard password file, unless you choose to override this (see *Managing Runtime Parameters*).

While configuring the Password Inspector function, you will be able to choose which type of tests to perform. We recommend that you use the dictionaries provided by SPI. We also recommend that you customize the dictionary called `local.dict` to reflect local jargon used at your site since passwords consisting of local jargon are extremely vulnerable to compromise. The

PSI dictionaries are located in the `database/psi` subdirectory.

The SPI 2.1 Password Inspector uses a fast encryption routine to check passwords. Still, a typical UNIX machine can try anywhere from 60 to 300 attempts per second. This means that trying 25 password entries using approximately 30,000 words (dictionary plus trivial passwords) will, however, take several hours of CPU time.

Additionally, checking all permutations of 3 characters will almost double the run time. We recommend that you schedule a full check (using large dictionaries) when you first install SPI. Select the “password log file” option in the “PSI Parameters” form, so that a log file will be generated and maintained. Then, future runs of the Password Inspector will only check the passwords which have changed since the previous check. After that, run the Password Inspector at least once a week to check any passwords changed during the week.

3.4 The Binary Inspector Tool (BIT)

The *Binary Inspector Tool (BIT)* determines the version level of important system binaries to report those that have not had the most recent security patches applied. BIT also may be run on the command line.

Figure 3-1 Binary Inspector parameters

```
         B I N A R Y   I N S P E C T O R   P A R A M E T E R S

                        Parameter Filename:  default

   Table to use for Bit        : bit.tbl
   Use all table entries (Y/N) : N
   Ignore file modes, except SETUID, SETGID (Y/N) :  N
   Ignore all file modes (Y/N) : N



   Write New Parameter File? (Y/N):  N
            New Parameter Filename:

Previous Field <ctrl b>          Help <?>           Return to Menu <esc>
```

BIT examines compatible host file systems to determine known vulnerabilities of system binaries and suggest the best patch or replacement binary to correct the vulnerability. Later patches (changes to the executable binary image, or the application source code), are offered to correct the vulnerability. Vendors often prematurely ship operating systems that are later found to have security holes. This method works alright if busy system managers have the time, which they often don't, to keep up on security patches and vulnerabilities.

BIT makes the system manager's life easier by helping to close these security holes in Unix operating systems. We have gathered information on vulnerabilities that the average system manager would have trouble obtaining. BIT functions by computing the RSA MD5 Encryption Signature for certain of your system's critical binaries, and then compares these to a supplied table containing the signatures for binary versions known to be either up-to-date or in need of replacement for security reasons. BIT packages that information in an easy-to-use program that warns the system manager and makes suggestions on how to protect the operating system.

3.5 The File Inode Change Detector (FCD)

FCD reports any significant changes found in a file's inode attributes. The purpose is both to help detect intrusions and to warn you when the attributes of system-critical files may have been inadvertently and deleteriously modified during routine administrative actions.

This program uses a previously written database file containing a list of critical system files and their inode attributes (permissions, number of links, size, checksum, modification time, etc.) as input. To generate this database file, select the "snapshot" option in the "Parameter Management" menu. You may also create it by running the `istat` program from the UNIX command line.

⇨Note: The initial snapshot is taken during SPI installation.

SEE ALSO the supplied UNIX manual pages: *fcd(8-fcd), istat(8-istat), spi(8-spi)*. The online location of these pages is shown in *Appendix C—Installed Directory Structure*.

3.6 The File Data Change Detector (DCD)

The *File Data Change Detector (DCD)* employs the RSA Data Security, Inc. MD5 Message-Digest Algorithm to generate a unique cryptographic signature value for each critical system file. DCD provides an additional measure of intrusion detection by reporting any change found in a file's MD5signature value.[2]

This program uses a previously written database file containing a listing of critical system files and their content signatures as input. To generate this database file, select the "snapshot" option in the "Parameter Management" menu. You can also create it by using the `sstat` program from the UNIX command line.

⇨Note: The initial snapshot is taken during SPI installation.

SEE ALSO the supplied UNIX manual pages: *dcd(8-dcd), sstat(8-sstat), spi(8-spi)*. The online location of these pages is shown in *Appendix C—Installed Directory Structure*.

---

[2] The MD5 Message-Digest Algorithm is licensed by RSA Data Security, Inc.

4

ADMINISTRATIVE Functions

4.1 Active Process Status

If there are any SPI SECURITY functions currently running as background processes, each will be listed along with its process ID. If there are more than one, you will need to type Return .

If the Password Security Inspector was running in the background, the following window would appear on the display:

Figure 4-1 Process Status message showing background processes

```
############################# PROCESS STATUS ############################
#                                                                        #
#           Function PSI (process id = nnnnn) is still running           #
#                                                                        #
#           [ Press Return to Continue . . . ]                           #
#                                                                        #
##########################################################################
```

A similar listing appears if you try to run a SPI SECURITY function while that same function is still running.

If there are no background tasks running, the following window will appear:

Figure 4-2 Process Status message showing no background processes

```
############################# PROCESS STATUS ############################
#                                                                        #
#           There are No active processes                                #
#                                                                        #
#           [ Press Return to Continue . . . ]                           #
#                                                                        #
##########################################################################
```

4.2 Viewing Output Reports

Each security function generates an output report detailing the findings of the given inspection. These reports are saved in a subdirectory corresponding to the chosen security function. The menu for viewing these reports is shown below.

Figure 4-3 SPI Output Reports Menu

```
        S P I   O U T P U T   R E P O R T S   M E N U
-----------------------------------------------------------

      VIEW OUTPUT REPORTS FOR WHICH INSPECTOR?

        1.   Quick System Profile Reports
        2.   Access Control Test Reports
        3.   Full Password Profile Reports
        4.   Binary Inspector Tool Reports
        5.   File Inode Change Detector Reports
        6.   File Data Change Detector Reports
        7.   Archive Reports
        8.   Trash

      Enter Selection Number:

            Return to Menu <esc>   Help <?>
```

The reports are listed top-down, beginning with the most recent.

You have the option of both viewing the contents of reports, and of marking reports for transfer to either the `ARCHIVE` or the `TRASH` subdirectories. (When viewing the `TRASH`, you also have the option of deleting files altogether.)

For more information, seek online HELP in the SPI view utility.

4.3 Scheduling Automatic Runtimes

You can schedule SPI/UNIX functions to run automatically. A sample schedule is shown below.

Figure 4-4 Scheduling runtimes for SPI

```
            SCHEDULE OF RUNTIMES FOR THE (CHOSEN) INSPECTOR

Select Weekly or Daily Scheduling Mode (W/D) : W

Weekly Schedule Parameter File: default
Day(s) of the Week      Mon   Tue   Wed   Thur  Fri   Sat   Sun
Run Times (hh:mm)        :     :     :     :     :     :     :

Daily Schedule  Parameter File: default
(select hour(s) by entering minutes offset (mm=00...59) for that hour)
      Early Morning Hours   00:    01:    02:    03:    04:    05
       Late Morning Hours   06:    07:    08:    09:    10:    11:
          Afternoon Hours   12:    13:    14:    15:    16:    17:
            Evening Hours   18:    19:    20:    21:    22:    23:

Save All Choices You Have Made to This Form? (Y/N):

Previous Field <cntl b>     Next Field <return>
Return to Menu <esc>        Help <?>
```

Scheduling Mode

You may schedule the runtimes for each SPI security function in both a WEEKLY (W) and a DAILY (D) mode. The schedules for each mode must be viewed and edited separately. Select either (W) or (D), and the cursor will move to the appropriate entry fields.

In WEEKLY mode, you may select one or more days for a run. Only one runtime may be scheduled for each selected day.

In DAILY mode, you may schedule one or more hours for each run.

The rationale for this bi-modal scheduling concerns both the conservation of system resources and the mission-oriented approach to security management taken with SPI 2.1. For example, to ensure that your users are selecting strong passwords, you would want the Password Inspector to apply a large dictionary of testwords. This may consume several hours of computer time. Therefore, you might schedule this to run WEEKLY, say at 9:45 PM on Sunday evenings. In addition, you may want to test for NULL and "Joe" passwords, and for the appearance of new user accounts twice every day, since this will take only a few minutes of CPU time. You would then select DAILY mode, enter the name of an appropriate parameter file, and then enter the desired runtimes.

Naturally, you will want to specify different parameter files for the DAILY and WEEKLY schedules. See the section *Managing Runtime Parameters* for more information.

Day/Time Selection for Weekly Schedule

To select one or more days for a WEEKLY run, move to the desired day(s) and enter the desired run time, in "hh:mm" format.

Example

To set a run time for 3:20 AM every Monday, move to the fields labeled "Mon" and enter "03:20" into the two fields provided.

To clear an entry, enter blank spaces in the appropriate field.

Hour(s) Selection for Daily Schedule

To select one or more hours for a DAILY run, move to the desired hour(s) and enter a value for the minutes.

Example

To set a runtime for 3:20 AM, move to the field labeled "03:" and enter "20" in the field.

To clear an entry, enter blank spaces in the appropriate field.

Commit Changes to `cron`?

Select **`Y`** to have all the changes you have made (both WEEKLY and DAILY) written to the `cron` system file.

Select **`N`** if you wish to continue editing, or if you wish to exit without saving the changes you have made.

4.4 Managing Runtime Parameters

Each SECURITY function runs according to its ACTIVE parameter file. Default parameter files have been supplied for each function. You may generate alternate parameter files for each function.

Figure 4-5 SPI Parameters Management Menu

```
S P I   P A R A M E T E R S   M A N A G E M E N T   M E N U

           MODIFY PARAMETERS FOR WHICH INSPECTOR?

                 1.   Quick System Profile
                 2.   Access Control Test
                 3.   Full Password Profile
                 4.   Binary Inspector Tool
                 5.   File Inode Change Detector
                 6.   File Data Change Detector

              Enter Selection Number:


              Return to Menu <esc>   Help <?>
```

⇨Note: The "SCHEDULE RUNTIMES" feature maintains two additional and independent ACTIVE parameter files for each SECURITY function. See *Scheduling Automatic Runtimes*.

When you select a SECURITY function in the Parameter Management Menu, you will be presented with a form containing fields for those parameters which you may modify. You may also type **?** in any field to receive additional help about that field.

You will also be given the option of saving the changes you have made in a new parameter file. If you choose to save the changes, you will be prompted to supply a name for the new file.

⇨Note: The most recently saved parameter file becomes the ACTIVE parameter file for the selected SECURITY function. Future runs of the chosen function made from within the SPI user-interface package will execute according to the new parameter settings. If you modify the parameters for a SECURITY function, but do not save these changes, the chosen function will still apply the new parameters as long as you do not exit from the SPI package. Once you have exited and then re-entered SPI, the last saved parameter files again become effective.

Quick System Profile Runtime Parameters

The following paragraphs describe how to use the QSP runtime parameters.

Figure 4-6 Quick System Profile Parameters

```
       Q U I C K   S Y S T E M   P R O F I L E   P A R A M E T E R S

                        Parameter Filename:

          Check for GROUP Access to Devices? (Y/N):  N
          Check ANONYMOUS FTP Configuration? (Y/N):  Y

Scanning the entire file system for SetUID files will lengthen the run
time significantly.

          Check for Problem SetUID Files? (Y/N):     N

          Read/Write Access File:    access.list


   Write New Parameter File? (Y/N): N
            New Parameter Filename:

Previous Field <cntl b>     Previous Page <cntl p>     Next Page <cntl n>
Return to Menu <esc>        Help <?>
```

Group Access to Devices

Select **Y** if you want QSP to warn you of GROUP read/write access to the devices listed in the file `/etc/fstab`.

Ordinarily, QSP will only warn of WORLD read/write access to these devices.

ANONYMOUS FTP

Select **Y** if your system supports ANONYMOUS FTP. This causes QSP to warn you when:

- ❒ your password file has no ftp user account.
- ❒ root users are not in the `ftpusers` file.
- ❒ ftp has no home directory, or is the root directory.
- ❒ the files `ftp/etc/passwd` and `ftp/etc/group` are identical to `/etc/passwd` or `/etc/group`.
- ❒ certain ftp files do not exist, or have incorrect permissions or ownership.

Select **N** if your system does not support Anonymous FTP. Then QSP will only warn you if your `ftpusers` file contains root accounts.

Set_UID Files

Selecting **Y** forces QSP to scan the file system to report World-Writable `SetUID` binaries, and all `SetUID` shell scripts.

➩Note: Scanning the entire file system for `SetUID` files lengthens the run time significantly.

Saving New Parameter Files

Select **Y** to save the parameter changes you have made in a new parameter file. If you choose to save the changes, you will be prompted to supply a name for the new file.

➩Note: The most recently saved parameter file becomes the ACTIVE parameter file for the selected SECURITY function. Future runs of the QSP function made from within the SPI user-interface package will execute according to the new parameter settings. If you modify the parameters for a SECURITY function, but do not save these changes, the QSP function will still apply the new parameters as long as you do not exit from the SPI package. Once you have exited and then re-entered SPI, the last saved parameter files again become effective.

Access Control Test Runtime Parameters

The following paragraphs describe how to use the ACT runtime parameters.

Figure 4-7 Access Control Test parameters

```
                  A C C E S S   C O N T R O L   T E S T

                        Parameter Filename:  default

          Target Goal    : U=root
          Initial State : U=WORLD

          Rule Base     :  act.rules
          System File   :  /etc/passwd
          Group File    :  /etc/group

          Maximum Processing Depth (0..99)  :  50
          Exit on First Success Path (Y/N)  :  N

   Write New Parameter File? (Y/N): N
            New Parameter Filename:

Previous Field <ctrl b>         Help <?>          Return to Menu <esc>
```

TARGET Goal

The *TARGET* goal must have one of the following forms:

***TARGET* Goal**	**Example**
UID=<number>	"UID=0"
U=<user>	"U=root"
GID=<number>	"GID=10"
G=<group>	"G=staff"
FW=<file>	"FW=/etc/passwd"

The default is "UID=0".

ASSUMED Initial State

The *ASSUMED* initial condition state must have one of the following forms:

ASSUMED Initial State	**Example**
UID=<number>	"UID=0"
U=<user>	"U=root"
U=WORLD	U=WORLD
GID=<number>	"GID=10"
G=<group>	"G=staff"
FW=<file>	"FW=/etc/passwd"

The default is "U=WORLD".

RULEBASE

This file ("`act.rules`" in the `database/act` directory) contains the transition rules that ACT uses to generate intermediate subgoals in its quest for the target goal. (Details in Appendix F, *ACT Rules*.)

System File

You may replace this file (default: `/etc/passwd`) with any file having the same format as the `/etc/passwd` file. Indicate the full path.

Group File

You may replace this file (default: `/etc/group`) with any file having the same format as the `/etc/group` file. Indicate the full path.

Processing Depth

This value limits the degree to which ACT will attempt to generate access dependency chains before giving up. An access dependency chain is a set of steps an intruder could use to gain unauthorized access, especially where each step must be taken in turn. A value of 30 is reasonable on small systems. Larger values may be appropriate in systems supporting over 100 user accounts.

Values between 0 and 99 are allowable.

First Success Exit

If **`Y`** (Yes) is selected, ACT reports the first vulnerable path (access dependency chain) discovered, and then exits immediately. Select **`N`** (No) if you want ACT to report ALL vulnerable paths.

The default value is **`N`**.

Saving New Parameter Files

Select **Y** to save the parameter changes you have made in a new parameter file. If you choose to save the changes, you will be prompted to supply a name for the new file.

⇨Note: The most recently saved parameter file becomes the ACTIVE parameter file for the selected SECURITY function. Future runs of the ACT function made from within the SPI user-interface package will execute according to the new parameter settings. If you modify the parameters, but do not save these changes, ACT will still apply the new parameters as long as you do not exit from SPI. Once you have exited and then re-entered SPI, the last saved parameter files again become effective.

Password Security Inspector Runtime Parameters

The following paragraphs describe how to use the PSI runtime parameters.

Figure 4-8 Password Inspector Parameters

```
        P A S S W O R D   I N S P E C T O R   P A R A M E T E R S

               Parameter Filename: default

  System Password Filename:     /etc/passwd
  Test Shadow Passwords? (Y/N):  Y
  Dictionary Level (1,2,3):     1
  Print Found Passwords? (Y/N): N
  Use Reverse Words? (Y/N) :    N
  Use Uppercase Words? (Y/N):   N
  Use/Update Password Log File? (Y/N): Y
       Previous Password Log Filename: passwd.log
        Enforce Password Aging? (Y/N): Y
                Maximum Age (###)days: 180

Write New Parameter File? (Y/N):    N
         New Parameter Filename:

Previous Field <cntl b>     Previous Page <cntl p>     Next Page <cntl n>
Return to Menu <esc>        Help <?>
```

Password Files

By default, the system password file `/etc/passwd` is the file examined by the Password Security Inspector. To apply password inspection to another file of the same format, you must enter its full pathname.

If your system has shadow passwords (`/etc/security/passwd.adjunct`) and you select **Y** for "Test Shadow Passwords" (next field) then only those user accounts common to both files will be tested.

Test Shadow Passwords

If your system uses the C2 shadow password file scheme, you may select **Y** to have these passwords examined. PSI then merges data from the selected standard password file (previous field) with the file "`/etc/security/passwd.adjunct`".

Note that only those user accounts common to both the standard and shadow password files will be examined when **Y** is selected.

If you select **N**, then the file indicated as the standard password file (previous field) must have the conventional UNIX password file format.

Password Dictionaries

The Password Inspector is prepared to use three password dictionaries. These are `local.dict`, `trivial.dict`, and `full.dict`. (These are located in the `database/psi` subdirectory.) `Local.dict` is supplied empty, and is intended for your use in applying terms specific to your site. `Trivial.dict` contains many words not usually found in dictionaries, such as male and female names, cars, cartoon characters, stars from stage and screen, etc. `Full.dict` is a large dictionary of the English language.

Select **1** to apply only `local.dict.`
Select **2** to add `trivial.dict`
Select **3** to add `full.dict` and `trivial.dict`

Printing Passwords

Selecting **Y** prints the bad passwords in the output report.

Selecting **N** prints the warning `<MATCHED>` in the output report whenever a bad password is discovered.

Reversing Testwords

Selecting **Y** causes the Password Inspector to also test for the reverse of each dictionary word. For example, "secret" becomes "terces".

⇨Note: This doubles the total execution time.

Selecting **N** results in only the account names and `GCOS` field words being reversed.

Uppercasing Testwords

Selecting **Y** causes the Password Inspector to also test for the uppercase of each dictionary word. For example, “secret” becomes “SECRET”.

⇨Note: This doubles the total execution time.

Selecting **N** results in only the account names and `GCOS` field words being uppercased.

Using Password Log Files

The Password Inspector can be used either with or without a log file. Two reasons to use a log file are:

- ❐ you can avoid retesting passwords which have not changed since the last log file update.
- ❐ passwords can be “dated” to support a password-aging policy.

To specify a password log file, select **Y** in the parameter form. You will be prompted for a file name. Selecting a previously unused file name will force the creation of a new (i.e. empty) log file. This causes all accounts to be retested (appropriate when new password test dictionaries are to be applied.)

To see a listing of previous password log files, invoke HELP when prompted for the filename.

⇨Note: If you choose not to use a log file, then all accounts will be retested, and no log file will be generated.

Password Aging

Select **Y** to apply password aging. You will then be prompted to enter a password age limit (in days.) When password aging is in effect, all passwords older than the specified age limit will be reported. Dating of passwords begins with the initial use of the given log file.
You must use a password log file to apply password aging.

Saving New Parameter Files

Select **Y** to save the parameter changes you have made in a new parameter file. If you choose to save the changes, you will be prompted to supply a name for the new file.

➩Note: The most recently saved parameter file becomes the ACTIVE parameter file. Future runs of the Password Inspector made from within the SPI user interface package will execute according to the new parameter settings. If you modify the parameters, but do not save these changes, the Password Inspector will still apply the new parameters as long as you do not exit from the SPI package. Once you have exited and then re-entered SPI, the last saved parameter files again become effective.

Binary Inspector Runtime Parameters

The following paragraphs describe how to use the BIT runtime parameters.

Figure 4-9 Binary Inspector Tool Runtime Parameters

```
          B I N A R Y   I N S P E C T O R   P A R A M E T E R S

                      Parameter Filename:  default

   Table to use for Bit        : bit.tbl
   Use all table entries (Y/N) : N
   Ignore file modes, except SETUID, SETGID (Y/N) :  N
   Ignore all file modes (Y/N) : N




   Write New Parameter File? (Y/N):  N
            New Parameter Filename:

 Previous Field <ctrl b>          Help <?>          Return to Menu <esc>
```

BIT Database Table

By default, the file `bit.tbl` is the file used by BIT. To have BIT use another file of the same format, but of a different name you must enter its full pathname.

Use All Table Entries

Selecting **Y** causes BIT to examine all BIT database table entries.

Selecting **N** causes BIT to use only those database table entries matching the operating system type and operating system level. This is the default.

Ignore File Modes, except SETUID, and SETGID

Selecting **Y** causes BIT to ignore file mode and permission differences when comparing binaries, except when `SETUID` or `SETGID` files are being compared.

Selecting **N** causes BIT to check file mode and permission differences when comparing binaries, including `SETUID` and `SETGID`permissions. This is the default.

Ignore All File Modes

Selecting **Y** causes BIT to ignore all file mode and permission differences when comparing binaries, including `SETUID` and `SETGID` permissions.

Selecting **N** causes BIT to check file mode and permission diffences when comparing binaries. Depending on the previous flag's value `SETUID` and `SETGID` permissions may or may not be checked. The default value for this flag is **N**.

Saving New Parameter Files

Select **Y** to save the parameter changes you have made in a new parameter file. If you choose to save the changes, you will be prompted to supply a name for the new file.

⇨Note: The most recently saved parameter file becomes the ACTIVE parameter file for the selected SECURITY function. Future runs of the BIT function made from within the SPI user-interface package will execute according to the new parameter settings. If you modify the parameters, but do not save these changes, the BIT function will still apply the new parameters as long as you do not exit from the SPI package. Once you have exited and then re-entered SPI, the last saved parameter files again become effective.

File Inode Change Detector Runtime Parameters

The following paragraphs describe how to use the FCD runtime parameters.

Figure 4-10 File Inode Change Detector

```
      F I L E   I N O D E   C H A N G E   D ET E C T O R


     1.  Change Configuration Parameters
     2.  Edit Inode Database File
     3.  Run Snapshot

         Selection:

Parameter File:
-------------------------------------------------------------------
OPTIONS
  Database File:
     Report All Files Tested [Y/N]:
     Report All Conflicts [Y/N]:
     Report Complete CRCs [Y/N]:
-------------------------------------------------------------------
Save This Configuration [Y/N]:
        New Parameter File Name:

Previous Field <cntl b> Previous Page <cntl p> NextPage <cntl n>
Return to Menu <esc>    Help <?>
```

Menu Selection

Select **1** to examine or modify the runtime parameters.

Select **2** to edit the list of files selected for inode status verification.

Select **3** to "snapshot" the system. This rewrites the indicated database file which is a baseline of inode data from which future deviations will be reported.

Report All Files Tested

Selecting **Y** causes all examined files to be included in the output report. Files will be listed even when no inode changes have been detected.

Selecting **N** causes only those files in which inode changes have been detected, to be reported. This is the default setting.

Report All Conflicts

Selecting **Y** causes a file to be reported even when the only inode change indicated is that the file has been read.

Selecting **N** causes a file to be reported only when the more substantive kinds of inode changes are detected (permissions, links, ownerships, etc.) This is the default setting.

Report CRC Values

Selecting **Y** causes each file's CRC value to be included in the output report.

Selecting **N** causes suppression of file CRC values in the output report. This is the default setting.

Saving New Parameter Files

Select **Y** to save the parameter changes you have made in a new parameter file. If you choose to save the changes, you will be prompted to supply a name for the new file.

⇨Note: The most recently saved parameter file becomes the ACTIVE parameter file for the selected SECURITY function. Future runs of the FCD function made from within the SPI user-interface package will execute according to the new parameter settings. If you modify the parameters, but do not save these changes, the FCD function will still apply the new parameters as long as you do not exit from the SPI package. Once you have exited and then re-entered SPI, the last saved parameter files again become effective.

File Data Change Detector Runtime Parameters

The following paragraphs describe how to use the DCD runtime parameters.

Figure 4-11 File Data Change Detector

```
      F I L E   D A T A   C H A N G E   D E T E C T O R


     1.  Change Configuration Parameters
     2.  Edit Checksum Database File
     3.  Run Snapshot

         Selection:

Parameter File:
---------------------------------------------------------------
OPTIONS
  Database File:
     Report All Files Tested [Y/N]:
---------------------------------------------------------------
     Save This Configuration [Y/N]:
     New Parameter File Name:

Previous Field <cntl b> Previous Page <cntl p> NextPage <cntl n>
Return to Menu <esc>    Help <?>
```

Menu Selection

Select **1** to examine or modify the runtime parameters.

Select **2** to edit the list of files selected for checksum verification.

Select **3** to "snapshot" the system. This rewrites the indicated database file, a baseline of file checksums from which future deviations will be reported.

Report All Files Tested

Selecting **Y** causes all examined files to be included in the output report. Files are listed even when no checksum changes have been detected.

Selecting **N** causes only those files in which checksum changes have been detected, to be reported. This is the default setting.

Saving New Parameter Files

Select **Y** to save the parameter changes you have made in a new parameter file. If you choose to save the changes, you will be prompted to supply a name for the new file.

⇨Note: The most recently saved parameter file becomes the ACTIVE parameter file. Future runs of DCD made from within the SPI user-interface package will execute according to the new parameter settings. If you modify the parameters, but do not save these changes, DCD will still apply the new parameters as long as you do not exit from the SPI package. Once you have exited and then re-entered SPI, the last saved parameter files again become effective.

A

Installation Guide

A.1 Purpose

This appendix describes how to install SPI/UNIX.

For best results, build and run SPI as root. At the very least, build SPI under the same user ID as it will be run.

Once the SPI distribution is in the desired source directory, the structure of files and subdirectories should match the listing given in *Appendix B—Distribution Directory Structure*. When you are ready to begin the installation, type **`Build`**. You will be asked a series of questions concerning the product configuration.

⇨Note: You will have the opportunity to select a different directory in which to place the SPI executables, databases, and parameter files.

If for some reason **`Build`** fails on your system, try **`Build -v`**. This causes the `build` procedure to report its progress in much greater detail, and gives you the opportunity to interactively resolve or override the default configuration values. See the file `A_README` included with the distribution for more information on configuration problems.

Example

Below is a sample installation session. User input is indicated in boldface type. The symbol <cr> indicates a carriage return. Default values are shown in [brackets].

Figure A-1 Sample installation session

```
host.llnl.gov# Build

            Security Profile Inspector

      . . . Configuration in Progress . . .

          ( This may take a few minutes )

The SPI executables must be placed in a fixed location,
called the installation directory.

Select installation directory [/usr/lib/spi]  <cr>

The SPI database files must be placed in a fixed
location, called the database directory.

Select database directory [/usr/lib/spi]  <cr>

Which user id should be assigned to the executables?
[root]  <cr>

Which group id should be assigned to the executables?
[wheel]  <cr>


End of configuration questions.


Doing variable substitutions on .SH files...

            User Preference Configuration

       ( setting internals, please wait  . . . )


What is the default editor on your system?
[/usr/ucb/vi]  <cr>

Mail address to send mail to? [root]  <cr>

End of user preference questions.
```

Figure A-1 Sample installation session (continued)

```
      Compiling and installing SPI executables . . .

This may take 5 to 15 minutes, depending upon the speed
of your machine.  All actions will be logged in a file
called "make.log" for you.

Do you wish to compile and install SPI now? [y]  <cr>

     . . . Compiling SPI binaries . . .

     . . . Installation Complete.

Warning:  Running this final step will remove all of
the SPI source code.  If the source code is removed,
and you wish to change anything you will have to re-
install SPI from the installation tape.

In order to completely secure the SPI program, I can
remove all of the source code and installation scripts.
After SPI has been installed, the source should be
removed from the system to protect it from theft
or tampering.

Do you wish to remove SPI source now? [n]  <cr>
Ok, leaving source alone ...

host.llnl.gov#
```

This completes the installation procedure. If no errors were encountered, your chosen installation directory should contain files and subdirectories structured according to *Appendix C—Installed Directory Structure*.

B

Distribution Directory Structure

Prior to installation, your SPI distribution should contain the files and directories shown in the following figure.

Figure B-1 Pre-installation SPI files and directories

```
   SPI
---------
    |
    | (executables)
    | [ Build ]       ( controlling installation script )
    |                 ( calls Configure and Reconfig )
    | [ Configure ]   ( installation portability configuration script )
    |                 ( creates "config.sh" )
    | [ Reconfig.SH ] ( Reconfig is the user preferences configuration script )
    |                 ( It is called after Configure in Build, and may be called )
    |                 ( independently thereafter )
    | [ config.h.SH ] ( config.h with substitutable variables )
    |                 ( creates "config.h" )
    | [ Makefile.SH ] ( Makefile with substitutable variables )
    |                 ( creates "Makefile" )
    |
    |------     ui
    |       ----------
    |            |
    |            |------ lib    [ user interface library routines ]
    |            |------ ptables[ parameter parsing tables ]
    |            |------ strat  [ developer-defined forms processing routines ]
    |            |------ screen [ main.menu ]
    |                           [ report.menu param.menu sched.menu ]
    |                           [ qsp.form act.form psi.form bit.form ]
    |                           [ fcd.form dcd.form cron.form ]
    |
    |------   mexec   [ source for mx, view, flist ]
    |
    |------   quisp   [ source for qsp ]
    |
    |------   acid    [ source for act ]
    |
    |------   pass    [ source for psi ]
    |
    |------   pwd     [ components of password dictionaries ]
    |
    |------   bitsrc  [ components of bit ]
    |
    |------   sum     [ source for dcd, sstat, sum ]
    |
    |------   inode   [ source for fcd, istat, iprt ]
    |
    |------   cron    [ Makefile for cron database ]
    |
    |------   doc     [ spi.man mx.man qsp.man act.man psi.man fcd.man dcd.man ]
                      [ istat.man  sstat.man ]
```

C

Installed Directory Structure

After you have installed SPI, the installation directory should contain the following files and subdirectories:

Figure C-1 Post-installation SPI files and subdirectories

```
   SPI
---------
    |
    | (executables)
    | [ spi mx view qsp act psi bit fcd dcd istat sstat iprt ]
    |
    |------ parameters
    |       ----------
    |             |
    |             |------ qsp  [ .config   default daily weekly ... ]
    |             |            [ .config.w .config.d .W .D ]
    |             |------ act  [ .config   default daily weekly ... ]
    |             |            [ .config.w .config.d .W .D ]
    |             |------ psi  [ .config   default daily weekly ... ]
    |             |            [ .config.w .config.d .W .D ]
    |             |------ bit  [ .config   default daily weekly ... ]
    |             |            [ .config.w .config.d .W .D ]
    |             |------ fcd  [ .config   default daily weekly ... ]
    |             |            [ .config.w .config.d .W .D ]
    |             |------ dcd  [ .config   default daily weekly ... ]
    |                          [ .config.w .config.d .W .D ]
    |
    |
    |------ databases
    |       ----------
    |             |
    |             |------ qsp  [ access.list ]
    |             |------ act  [ act.rules ]
    |             |------ psi  [ passwd.log dictionaries ... ]
    |             |------ bit  [ bit.tbl ... ]
    |             |------ fcd  [ inode.db  suid.db ]
    |             |------ dcd  [ chksum.db ... ]
    |
    |------ results
    |       ----------
    |             |
    |             |------ qsp  [ qsp_pid ... ]
    |             |------ act  [ act_pid ... ]
    |             |------ psi  [ psi_pid ... ]
    |             |------ bit  [ bit_pid ... ]
    |             |------ fcd  [ fcd_pid ... ]
    |             |------ dcd  [ dcd_pid ... ]
    |
                          • • •
```

Figure C-1 Post-installation SPI files and subdirectories (continued)

```
|------ screen
     |
     |              [ main.menu (runqsp runact runpsi runbit runfcd rundcd) ]
     |              [            (active reports schedule params EXIT) ]
     |
     |              [ param.menu ]
     |                        \
     |                     [ qsp.form act.form psi.form bit.form fcd.form ]
     |                     [ dcd.form ]
     |
     |              [ report.menu (qsp act psi bit fcd dcd) ]
     |
     |              [ sched.menu  (qsp act psi bit fcd dcd) [ cron.form ] ]
     |
     |------ man  [ mx.man qsp.man act.man psi.man bit.man fcd.man dcd.man ]
```

D

Sample Parameter Files

D.1 Parameter Files for the Quick System Profile

The following is a list of sample parameter files for the QSP function.

Figure D-1 Sample QSP parameter file

```
#
#  default parameter file for the Quick System Profile
#
#     check device group permissions also?      [-g]
group=Y
#     check anonymous FTP?          [-a]
aftp=Y
#     scan for all SetUID files?    [-s]
suid=N
#     read/write access files list  [-f]
file=access.list
#


#
#  "daily" parameter file for the Quick System Profile
#
#     check device group permissions also?      [-g]
group=Y
#     check anonymous FTP?          [-a]
aftp=Y
#     scan for all SetUID files?    [-s]
suid=N
#     read/write access files list  [-f]
file=access.list
#


#
#  "weekly" parameter file for the Quick System Profile
#
#     check device group permissions also?      [-g]
group=Y
#     check anonymous FTP?          [-a]
aftp=Y
#     scan for all SetUID files?    [-s]
suid=Y
#     read/write access files list  [-f]
file=access.list
#
```

D.2 Parameter Files for the Password Security Inspector

The following is a list of sample parameter files for the Password Security Inspector function.

Figure D-2 Sample PSI parameter file

```
#
#  default parameter file for the Password Security Inspector
#
#     password file to inspect      [-p]
pwname=/etc/passwd
#     dictionary level
dlevel=1
#     print found passwords?        [-f]
fprint=N
#     reverse dictionary words?     [-r]
reverse=N
#     uppercase dictionary words?   [-u]
upper=N
#     use and update a logfile?
updlog=Y
#     specified logfile       [-l logfile]
logname=passwd.log
#     enforce password aging?
pold1=Y
#     password age limit            [-o ndays]
pold2=180
#

#  daily parameter file for the Password Security Inspector
#
#     password file to inspect
pwname=/etc/passwd
#     dictionary level
dlevel=1
#     print found passwords?
fprint=N
#     reverse dictionary words?
reverse=N
#     uppercase dictionary words?
upper=N
#     use and update a logfile?
updlog=N
#

#  weekly parameter file for the Password Security Inspector
#
#     password file to inspect
pwname=/etc/passwd
#     dictionary level
dlevel=2
#     print found passwords?
fprint=N
#     reverse dictionary words?
reverse=Y

#     uppercase dictionary words?
upper=Y
#     use and update a logfile?
updlog=Y
#     specified logfile
logname=weekly.log
#     check password aging?
pold1=N
```

D.3 Parameter Files for the Binary Inspector Tool

The following is a list of sample parameter files for the Binary Inspector Tool function.

Figure D-3 Sample BIT parameter file

```
#
#  default parameter file for the Binary Inspector Tool
#
#
#  Table for bit to use
#
table=bit.tbl
#
#  Force the use of all table entries
#
force=N
#
#  Ignore file modes, except SETUID, SETGID
#
mode=N
#
#  Ignore all file modes
#
allmode=N
```

D.4 Parameter Files for the File Inode Change Detector

The following is a list of sample parameter files for the FCD function.

Figure D-4 Sample FCD parameter file

```
#
#  default parameter file for the File Inode Change Detector
#
#     Original critical file list.
m_file=inode.log
#     Report all files tested?
r_all=N
#     Report checksum differences?
r_crc=N
#     Report any file access?
r_conf=Y
#

#
#  daily parameter file for the File Inode Change Detector
#
#     Original critical file list.
m_file=inode.log
#     Report all files tested?
r_all=N
#     Report checksum differences?
r_crc=N
#     Report any file access?
r_conf=Y
#

#
#  weekly parameter file for the File Inode Change Detector
#
#     Original critical file list.
m_file=inode.log
#     Report all files tested?
r_all=Y
#     Report checksum differences?
r_crc=Y
#     Report any file access?
r_conf=Y
#
```

D.5 Parameter Files for the File Data Change Detector

The following is a list of sample parameter files for the DCD function.

Figure D-5 Sample DCD parameter file

```
#
#  default parameter file for the File Data Change Detector
#
#     Master List of Critical Files
m_file=chksum.log
#     Report All Files Tested?
r_all=N
#


#
#  daily parameter file for the File Data Change Detector
#
#     Master List of Critical Files
m_file=chksum.log
#     Report All Files Tested?
r_all=N
#


#
#  weekly parameter file for the File Data Change Detector
#
#     Master List of Critical Files
m_file=chksum.log
#     Report All Files Tested?
r_all=Y
#
```

E

Sample Output Reports

E.1 System Profile Reports

The System Profiler consists of 12 major subfunctions, each involving a particular area of system security. Lines beginning with the word `Warning!` are considered possible sources of system vulnerability, and require further investigation to determine whether the warning is appropriate for your system configuration and situation.

Figure E-1 Sample Profile report

```
QSP Security Report:  Wed Nov 27 14:19:06 PST 1991
from host zaphod

QSP: Root Environment . . .
Warning!  ".” is in roots path!
QSP: Device Permissions . . .
QSP: Critical File Read/Write . . .
Warning!  /etc/motd is _World_ writable!
Warning!  /usr/spool/mail is _World_ writable!
QSP: rc (boot) File Dependencies . . .
Warning!  File /etc/motd (in /etc/rc*) is _World_ writable!
QSP: Cron File Dependencies . . .
QSP: Group File Syntax . . .
Warning!  Group file, line 904, does not have 4 fields: mikez:*:1263
QSP: Home Directories . . .
QSP: User Configuration Files . . .
Warning!  User smith: /home/smith/.netrc is readable; mode 0644!
QSP: Miscellaneous Details . . .
Warning -- rexecd is enabled in /etc/inetd.conf!
Warning! /usr/bin/uudecode creates setuid files!
QSP: FTP Configuration . . .
Warning! Need user /etc/ftpusers for anonymous ftp to work!
Warning! Home directory for ftp doesn't exist!
ftp-Warning! /etc/passwd and /home2/azb/spiA/p.4196 are the same!
ftp-Warning! /etc/group and /home2/azb/spiA/g.4196 are the same!
ftp-Warning! should be owned by root or root!
ftp-Warning! Incorrect permissions on "ls” in /bin!
ftp-Warning! Incorrect permissions on "passwd” in /etc!
ftp-Warning! Incorrect permissions on "group” in /etc!

END QSP Security Report:  Wed Nov 27 14:22:47 PST 1991
```

Below is a more detailed explanation of the various warning messages that may appear in a System Profile output report.

check_root

The first things QSP checks are the following files and directories:

```
/bin
/etc
/.login
/.cshrc
/.profile
/.rhosts
```

and if enabled (currently disabled on QSP)

```
/bin/*
/etc/*
```

The next check is to scan important root files (`/.login /.profile /.cshrc`) for special strings (strings that look like filenames or directory search paths) and check the permissions of those filenames under the assumption that if someone were to somehow subvert the files used by the root control files, you could affect root’s environment. No distinction is made between files that the control file writes versus reads. Some errors may not be a problem.

MODULE OUTPUT

```
QSP: Root Environment . . .

Warning!  Root does not own the following
file(s): <file list>
```

The warning message shown above warns users if important root files are not owned by root. If this is the case, then the file’s owners can change the contents of these files possibly tricking ‘root’ into using the bogus contents.

```
Warning!  File $i (inside root file $stf)
is _World_ writable!
Warning!  File /etc/motd (inside root
file /etc/rc.local) is _World_ writable!
```

The warning message shown above indicates a file which is referenced inside a root control file that is writable by others. If this file is used as input to the control file processing, a non-root user could possibly subvert the processing of the control file. The control file must be carefully checked to see if the offending file could possibly influence the behavior of the control file. It is better to err on the side of caution here.

```
Warning!  Root's umask set to $umsk in $i
Warning!  Root's umask set to 002 in
/.profile
```

The warning message shown above indicates that a umask setting in a root control file is insufficient to protect a file created by the root user. Upon creation, a file gets its default permissions from the umask setting. If the umask is set incorrectly, root may create a sensitive file which is world readable or world writable.

```
Warning!  $ftp exists and root is not in it
Warning!  /etc/ftpusers exists and root is
not in it
```

If ftp is enabled on a system, and there is a file `/etc/ftpusers`, this file, shown above, contains a list of the users *NOT* allowed to ftp. If root is in this file, then root will not be allowed to ftp.

```
Warning!  A "+" entry in $equ!
Warning!  A "+" entry in /etc/hosts.equiv!
```

The `/etc/hosts.equiv` file, shown above, holds the names of the trusted hosts. Any user may log in as himself from any trusted hosts without supplying a password. A "+" says to trust *any* host. This basically allows any machine which can communicate with your host to log in as any valid user (except root) without needing a password.

```
Warning!  "." is in roots path!
```

The "." in a root path, shown above, is dangerous because a user can place a Trojan horse program in any directory accessible to that user. If root accidentally were to be in that directory and run that program, it could cause a break in security. Putting "." last in the path minimizes, but does not completely eliminate, the chances of this happening.

```
Warning!  Directory $i is _World_ writable
and in roots path!
Warning!  Directory /usr/lib is _World_
writable and in roots path!
```

Similar to having "." in the root path. Having a world-writable directory in root's path, as shown above, is also a big security risk. All directories in root's search path should not be world writable.

```
Warning!  Non root entry in $file! $user
Warning!  Non root entry in /.rhosts! joe
```

If there is a non-root entry in root's `.rhosts` file (`/.rhosts`), as shown above, then the indicated user on the specified machine may log in to your machine as root without supplying the root password. If the other machine is compromised then so is root on your machine.

check_devices

`Check_devices` first checks the `/etc/exports` file and informs you if any file systems are exported with no restrictions. Then it gets a list of all local and NFS mounted file systems. It then checks the devices and directories associated with each mount for correct permissions. Directories and devices should never be world writable. Local devices should never be world readable (to prevent snooping of data using the raw device).

MODULE OUTPUT

```
QSP: Device Permissions . . .
Warning!  NFS file system %s exported with
no restrictions.
```

The warning shown above is given if a file system is exported with no restrictions. The default is to allow all hosts to mount the file system and read and write to it according to standard UNIX permissions. This could allow others to mount and read file systems you may wish to keep private.

```
Warning!  %s is group readable!
Warning!  %s is _World_ readable!
```

The warning shown above indicates that a raw device file is readable, allowing world (or a specific group) to read all data on the device, bypassing normal file system security.

```
Warning!  %s is group writable!
Warning!  %s is _World_ writable!
```

The warning shown above indicates that a raw device file is writable, allowing world (or a specific group) to change or destroy data on the device, bypassing normal file system security.

check_access

The check_access module takes a list of critical files (in access.list) and checks for proper permissions. A list of files typically used and the reasons for checking are given below:

MODULE OUTPUT

```
QSP: Critical File Read/Write . . .

Config file access.list doesn't exist!
```

The error message shown above indicates that the file containing the list of files to be checked is missing.

```
Warning!  %s is group readable!
Warning!  %s is _World_ readable!
Warning!  %s is group writable!
Warning!  %s is _World_ writable!
Warning!  %s is SUID!
Warning!  %s is SGID!
```

The warnings shown above indicate that the test specified in the access.list file failed. Consult the list below to see why the warning is a security problem.

Critical files:

```
/             /usr/lib
/etc          /usr/spool
/usr          /usr/spool/mail
/bin          /usr/spool/news
/usr/bin      /usr/local
/usr/etc      /usr/local/bin
/usr/adm      /usr/local/lib
```

Each of the directories shown above contains either root control files or binaries that are used by root or precious system data files. If they are world writable, anyone may replace a file in these directories with one of their own. Data can also be destroyed.

```
/usr/spool/uucp
/usr/spool/at
```

The directories shown above contain sensitive control files for critical system function. By allowing write access to anyone, one can subvert the control files by replacing them with different ones.

```
/usr/users
/Mail
```

Some directories, such as the ones listed above, should be protected to protect the personal privacy of valid users.

```
/etc/security
```

The directory shown above should not be readable or writable because some UNIX systems store security control data there.

```
/.*
(/.login
/.profile
/.cshrc
/.rhosts)
```

All "." files in /, shown above, should not be writable by others because they are typically root's control files.

```
/etc/*          /usr/local/lib/*
/bin/*          /usr/local/bin/*
/usr/bin/*      /usr/etc/yp*
/usr/etc/*      /usr/lib/aliases
/usr/adm/*      /usr/lib/sendmail
/usr/lib/*
```

The files shown above are system control files and system binaries. They should not be world writable as root executes these binaries and uses these control files.

```
/usr/lib/crontab
/usr/spool/uucp/L.sys
/.netrc
/usr/adm/sulog
/etc/btmp         (HP-UX)
/etc/securetty         (HP-UX)
```

The files shown above should not be writable or readable as they contain system configuration information, sensitive security log information, and/or clear-text passwords.

```
/dev/kmem
/dev/mem
/dev/drum       (SunOS)
```

The files shown above should not be writable or readable by others since they allow access to core memory. A sufficiently intelligent program could get sensitive information from in-core data structures.

```
/dev/nit        (SunOS)
```

The file shown above must not be writable or readable by others as it allows eavesdropping on the Ethernet.

check_rc_files

This module checks the permission on the `/etc/rc*` files, which are the startup files for UNIX (should do `/etc/init*` files for sysV). Since they are run as root in single user mode, they must not be writable or someone could cause root to execute bogus code. Additionally any files that are referenced from these files need to also be checked.

MODULE OUTPUT

```
QSP: rc (boot) File Dependencies . . .

Warning!  %s is group writable!
Warning!  %s is _World_ writable!
```

All control files shown above are world writable. This allows someone to change files used by root at start up time.

```
Warning!  File $i (in /etc/rc*) is _World_
writable!
```

Files referenced from single user control files, as shown above, are world writable. This allows someone to change files used by root at start up time.

check_cron

This module checks the cron system for security problems. Since cron runs as root, should a user get write access to the cron control files, he can trick the system into running a program for them as root.

Additionally any programs referenced from the cron files must be check to ensure they can't be overwritten.

MODULE OUTPUT

```
QSP: Cron File Dependencies . . .

Warning!  %s is group writable!
Warning!  %s is _World_ writable!
```

Cron control files, as shown above, are world writable. This allows someone to schedule a program to be run by root.

```
Warning!  $i (in $cron_kid) is World
writable!
```

The message shown above indicates that programs executed from a cron control file are world writable. This allows someone to overwrite the program file and have cron execute a different program.

check_group

This module checks the group file for correct syntax. Entries which have incorrect syntax or missing elements may be a security problem. Warning lines which contain "`YGroup`" indicate that the group yp map (Network Information System map) has the indicated problem, not the local `/etc/group` file.

MODULE OUTPUT

```
QSP: Group File Syntax . . .

Warning!  Group file, line %d, is blank
Warning!  YGroup file, line %d, is blank
```

The above message refers to invalid syntax—no blank lines or comments are allowed in a group file.

```
Warning!  Group file, line %d, does not
have 4 fields: %s
Warning!  YGroup file, line %d, does not
have 4 fields: %s
```

The above message refers to invalid syntax—some systems may ignore the entry. If the entry is ignored, there will be no indication of what group name belongs to a group ID.

```
Warning!  Group file, line %d,
nonalphanumeric user id: %s
Warning!  YGroup file, line %d,
nonalphanumeric user id: %s
```

The message above refers to invalid syntax—the group name must be [A-Za-z0-9]. Punctuation is discouraged, especially +-@ which indicate yp map entries.

```
Warning!  Group file, line %d, group has
password: %s
Warning!  YGroup file, line %d, group has
password: %s
```

The message above refers to the group entry having a password. This is a poor idea, because group passwords are known by all members needing group access, there is no accountability as to who is using the group account. It is better to have no passwords (i.e. *) and add users to the group as members. They can use the `newgrp` command to get group privileges.

```
Warning!  Group file, line %d, group has
invalid field for C2: %s
```

The message above refers to invalid syntax. Sun C2 syntax looks like '`#$name`'.

```
Warning!  Group file, line %d, nonnumeric
group id: %s
Warning!  YGroup file, line %d, nonnumeric
group id: %s
```

The message above refers to invalid syntax. A group ID must be numeric and between 0-65535 inclusive.

```
Warning!  Duplicate Group(s) found in
$etc_group:
Warning!  Duplicate Group(s) found in
yellow pages group:
```

The message above refers to invalid syntax. Two or more group entries have the same group name. They must be unique.

```
Warning!  Group "$1" has duplicate user(s):
Warning!  YGroup "$1" has duplicate user(s):
```

The message above refers to invalid syntax. The user has been listed more than once in the `members` field of the group entry.

`check_users`

Look in each user's home directory and make a few simple checks.

MODULE OUTPUT

```
QSP: Home Directories . . .

Warning! User %s's home directory %s is not
a directory! (mode 0%o)
```

The message above points out that a user doesn't have a home directory.

```
Warning! User %s's home directory %s is
mode 0%3.3o!
```

The message above indicates that a user's home directory is writable to others. This is not desirable because others can write and modify his files.

```
QSP: User Configuration Files . . .

Warning!  User %s: %s is mode 0%3.3o!
Warning!  User %s: %s is readable; mode
0%3.3o!
```

The messages shown above are given when a control file in the user's home directory is writable by others and also if important files contain clear-text passwords (`.netrc`). The following files are checked:

```
.rhosts    .kshrc
.profile   .tcshrc
.login     .netrc
.logout    .forward
.cshrc     .dbxinit
.bashrc    .distfile
.exrc      .emacsrc
```

`check_misc` Check miscellaneous details.

MODULE OUTPUT

```
QSP: Miscellaneous Details . . .

Warning!  %s is group writable!
Warning!  %s is _World_ writable!
```

The message above refers to `/etc/inetd.conf` which contains services that get executed when a port is opened over the network. If any of the programs referenced in `inetd.conf` are world writable, then a user can overwrite the program with his own program and run it by connecting to the `inetd` port.

```
Warning -- $rexd is enabled in $inetd!
```

The message above refers to `rexd` being enabled in `inetd.conf`. There are serious security problems with `rexd` and it should be disabled.

```
Warning -- $uu is enabled in $aliases!
```

The message above refers to `uudecode` being enabled in the `/etc/alisaes`. This can be a security problem depending on the version of `uudecode` on your system.

```
Warning!  $UUDECODE is SUID!
Warning!  $UUDECODE creates setuid files!
```

The message above refers to `uudecode` being `setuid` or creating `setuid` files. A properly devised mail message can cause `uudecode` to create `setuid` files. This is fatal if mail can be tricked into creating a `setuid` root file.

```
Warning! tftp is enabled on $HOSTNAME!
```

The message above refers to `tftp` being enabled and not very secure. Do not enable it unless you have diskless users who must boot. Some versions of `tftp` will give away your password file.

check_ftp

This module checks to see if `ftp` is enabled and sees if there are any problems with the setup.

MODULE OUTPUT

```
QSP: FTP Configuration . . .

Warning!  Need user $ftp for anonymous ftp
to work!
```

The message above refers to the flag that checks anonymous ftp being turned on, but no “ftp” account was found in the password file.

```
Warning!  $i should be in $ftpusers!
```

The message above refers to the need for all users with `uid 0` (root) to be in the `/etc/ftpusers` file, if it exists.

```
Warning!  Home directory for ftp doesn't
exist!
```

The message above refers to an "ftp" account in the password file, but the home directory specified doesn't exist. When someone logs in as "ftp", they will be placed in `/`. This is a security hole because it allows external users to read system files (like the `real /etc/passwd`).

```
Warning!  $ftproot ftp's home directory
should not be "/"!
```

The message above refers to ftp's home directory. This directory should be a captive directory where you can set up a sterile environment. If `/` is the home directory, it will allow external users to read system files like `/etc/passwd.`

```
ftp-Warning!  $ftppasswd and $passwd are
the same!
ftp-Warning!  $ftpgroup and $group are the
same!
```

The message above refers to the idea behind using a captive directory for ftp being so you can set up a sterile password and group file. If they are the same as the real files, this benefit flies right out the window.

```
ftp-Warning!  File $i is missing!
ftp-Warning!  $i should be owned by
$primary or $secondary!
```

The message shown above refers to the interaction between files and directories, and anonymous ftp. Several files and directories must be present and owned by root for anonymous ftp to work correctly. These files are:

```
~ftp/etc
~ftp/bin
~ftp/etc/passwd
~ftp/etc/group
~ftp/bin/ls
```

```
ftp-Warning!  $ftproot should be owned by
$primary or $secondary!
```

The message shown above refers to `~ftp`. `~ftp` should be owned by root or ftp, if not, an ftp user could overwrite system files in the ftp directory.`~ftp`.

```
ftp-Warning!  $ftproot should be mode 555!
```

The message shown above again refers to `~ftp`. If `~ftp` is owned by ftp, it must be mode 555.

```
ftp-Warning!  $ftprhosts should be empty!
ftp-Warning!  $ftprhosts should be owned
by $primary or $secondary!
```

The message shown above refers to the `~ftp/.rhosts` file. If present, `~ftp/.rhosts` should be empty and owned by root. If not, someone can ftp in as a user in the `~ftp/.`rhosts file.

```
ftp-Warning!  Incorrect permissions on
"ls" in $ftpbin!
```

The message shown above refers to the `~ftp/bin/ls` file. `~ftp/bin/ls` should be r--r--r--. Users should not be able to write to it.

```
ftp-Warning!  Incorrect permissions on
"passwd" in $ftpetc!
ftp-Warning!  Incorrect permissions on
"group" in $ftpetc!
```

The message above refers to `~ftp/etc/passwd` and `~ftp/etc/group`. They should not be writable for the same reasons `/etc/passwd` and `/etc/group` should not be writable. If they were writable (by anyone with an account on the system), then that person could replace any or all of it with new information. They could give themselves a root account, or even make themselves the only person on the system with an account at all. The `/etc/passwd` file is the heart of UNIX access control.

```
Warning!  Anon-ftp directory $i is World
Writable!
```

The message above refers to `~ftp` being world writable. If `~ftp` is world writable, any user can remove and recreate all the ftp subdirectories.

`check_suid`

This module starts at / and does a find looking for `suid` files. It checks various attributes about the file and reports it. Note: it you have a lot of NFS mounts, you may be spending the day searching for `suid` files. This should be pruned out.

MODULE OUTPUT

```
QSP: SetUID Files . . .

Warning!  %s is group writable!
Warning!  %s is _World_ writable!
```

The message above refers to an `suid` file being world writable. If an `suid` file is world writable, then anyone can modify the file to allow any program to be run as the `suid` owner. If the `suid` owner is root, the entire file system may be compromised.

```
Warning!  ROOT owned SUID file $i is type:
$type!
Warning!  User: $owner SUID file $i is
type: $type!
```

The message above refers to any `suid` file other than an executable being a probable security risk. Certainly `suid` scripts (other than `perl`) are a security hole and should be disabled. There are standard techniques of subverting `setuid` shell scripts.

E.2 Access Control Test Reports

This section contains some sample ACT test reports.

Positive Findings

In the example shown in Fig. E-2, ACT has found a way to compromise a root account (UID=0) from the initial state U=WORLD. Each element listed represents a subgoal achieved along the way to the target goal.

Figure E-2 Sample ACT report: POSITIVE

```
ACT:  Access Control Test Report                              DATE=061892

Initial Condition:  U=WORLD
Attempting Target Goal: UID=0 . . .

***  U=WORLD * (ASSUMED) FR=/user/bob/.login * trojan U=bob * grant UID=8765 *
member G=staff * member GID=10 * group_modify FR=/.login * trojan U=root * grant
UID=0 * TARGET
```

First Vulnerability Exploited:

It appears that user "bob" left his `.login` file world-writable. Hence anyone with an account on the system (U=WORLD) can "Trojan" this file to become "bob", and therefore gain UID=8765. Because "bob" is a member of group "staff", anyone can gain membership to G=staff, and so gain GID=10.

Second Vulnerability Exploited:

It seems that "root" has a `.login` file which is group-owned by "staff" and was left writable by its group. Anyone in this group can Trojan root's `.login` file to gain the root account, and UID=0.

Conclusion: Anyone with an account on this system can become root.

Negative Findings

In the following figure, the message "ALL CLEAR" indicates that the target goal could not be compromised from the given initial condition, based upon the rules listed in the file `act.rules` and limited by the given processing depth. See "Managing Runtime Parameters: Access Control Test" for more information.

Figure E-3 Sample ACT report: NEGATIVE

```
ACT:  Access Control Test Report                              DATE=061892

Initial Condition:  U=WORLD
Attempting Target Goal: UID=0 . . .

ALL CLEAR:  No vulnerable paths found to depth 14.
```

E.3 Password Inspector Reports

If a password log file is specified, the output report will contain only those accounts which have been retested. These are accounts with passwords that have changed from the value held in the log file, are older than the password age limit (if specified), or are new accounts.

Figure E-4 Sample Password Inspector report

```
====================================================================
==  SPI Password Inspector                      DATE: 091691:  ====
====================================================================
==  Password File To Inspect: /etc/passwd
==  Password Log File Use  d: /usr/lib/spi/database/psi/passwd.log
==  Password Dictionary Used: /usr/lib/spi/database/psi/local.dict
====================================================================
Acct-Name:     Too Old:        New-Acct:  Matched-Password
--------------------------------------------------------------------
root
--------------------------------------------------------------------
nobody
--------------------------------------------------------------------
daemon
--------------------------------------------------------------------
sys
--------------------------------------------------------------------
bin
--------------------------------------------------------------------
uucp
--------------------------------------------------------------------
sync                                      <NULL>
--------------------------------------------------------------------
alex                           YES        <MATCHED>
--------------------------------------------------------------------
brown
--------------------------------------------------------------------
longstar                                  <NULL>
====================================================================

Total Accounts=10    Total Matched=3     (Total Null=2)

Accounts with UID 0 (root privileges):

      root  brown

====================================================================
```

If you select to have the output report “print found passwords,” then the word `<MATCHED>` will be replaced with the actual plain text password found.

E.4 Binary Inspector Tool Reports

This section contains a sample BIT report.

Figure E-5 Sample Binary Inspector Tool report

```
--------------------- Binary Inspector Tool Report ----------------------

Target host       :  host.xxx.gov
OS Type           :  SunOS
OS Level          :  4.1.2
Inspection date   :  07/31/92

BIT table version      :  2.1
BIT table creation date:  07/01/92

/bin/csh:  program found - vulnerable
install patch 100083-01

/bin/rmail:   program found - vulnerable
install patch 100224-02

/bin/login:   program found - vulnerable
install patch 100201-01

/bin/mail:   program found - best available

/bin/crontab:   program found - best available

/usr/etc/rpc.lockd:   program found - best available

/usr/etc/mount:  program found - best available

/usr/etc/inetd:  program found - vulnerable
install patch 100178-01

/usr/ucb/w:  program found - checksum does not match BIT table entry
"force" option selected (possible OS type/version mismatch)

/usr/ucb/dbx:  program found - checksum does not match BIT table entry
Binary may be good, it should be investigated and verified to be good.

/usr/bin/sunview1/selection_svc:  program found - best available

/usr/ucb/mail:  program not found
"force" option selected (possible OS type/version mismatch)

/usr/ucb/finger:  program not found
Expected to find "/usr/ucb/finger" on this system

=============================================================================
The BIT database table reflects the least vulnerable publicly known
= versions of Unix executables for selected computer manufactures operating
= systems as of the date of the table.  Failure of this program to identify
= a program as vulnerable, shouldn't imply that its not vulnerable.
= Conversely,identifying a program as best, doesn't mean that it isn't
= vulnerable, only the best we know of as of the date of the table. If your
= vendor cannot help you, contact CIAC for further information. CIAC can be
= contacted by e-mail at ciac@cheetah.llnl.gov.  Fax messages can be sent
= to: (510) 423-0913.  CIAC's phone number is (510) 422-8193.
============================================================================
```

Below is a more detailed explanation of the various warning messages that may appear in a BIT output report.

BIT reads an entry from the BIT database table and then compares this entry with a file on the system being checked. If the binary checksum matches, BIT will produce output as follows:

```
/bin/login:   program found - vulnerable
install patch 100201-01
```

```
/bin/mail:   program found - best available
```

The first line of the output shows that the system being checked has a binary that BIT has identified as being vulnerable. BIT has given a patch number of the fix available.

The output remaining indicates that the binary found has matched a BIT table entry of a "known good" binary and there is no need for replacement.

The next series of output (shown below) indicates that a program was found by BIT, but the checksum of the binary did not match the BIT table entry for the binary. There are several possibilities to explain output such as that shown. One possibility is that the binary installed on your system is newer than the binary checksum in the BIT table. Another reason perhaps, is that your binary has been replaced with another "known good" binary. This may occur in such cases where your system has had a third party software installed to provide password filtering. In any case, this binary should be investigated, and its origin discovered, in case it may have been Trojaned by an unauthorized user.

```
/usr/ucb/dbx:  program found - checksum does
not match BIT table entry
Binary may be good, it should be
investigated and verified to be good.
```

The following output indicates that the pathname of the binary was found on the system BIT was checking, but the checksum of that binary does not match the table entry. This is similar to the previous output, but may not be a problem because the "force" option was used, forcing BIT to check every table entry, even if the operating system version or even vendor is different. Rerun the BIT utility without the "force" flag and follow the explanation given for the prior message.

```
/usr/ucb/w:  program found - checksum does
not match BIT table entry
"force" option selected (possible OS
type/version mismatch)
```

The output below indicates that the binary being checked could not be found on the system being checked. The binary could have been removed. or its location changed. Further investigation is probably necessary.

```
/usr/ucb/finger:  program not found
Expected to find "/usr/ucb/finger" on this
system
```

E.5 File Inode Change Detector Report

By default, the FCD reports only files that have changed attributes (permission mode, ownership, size, modification time, number of links, etc) or that have either missing or corrupt log data. That is, files with unchanged attributes, or whose only change is the access time (files were only read from) are not usually listed. Some examples follow:

Figure E-6 Sample File Inode Change Detector report

```
/etc/motd:
      File Mode: 100666 100644
      Change Time: Sat Aug  3 16:09:26 1991   Tue Aug  6 12:08:54 1991
/etc/utmp:
      File Size:: 2323 2362
      Change Time: Tue Aug  6 10:22:00 1991   Tue Aug  6 14:32:18 1991

. . .

Total file count: 366
Total file input errors: 0
```

⇨Note: The contents of `/etc/utmp` change from moment to moment. Hence size changes are expected for this file. (One should be alarmed if `/etc/utmp` were unchanged!)

If you select to have the report list all files that have been accessed in any way, most files will be listed as follows:

Figure E-7 Sample File Inode Change Detector report showing all accessed files

```
/etc/group:
      Access Time: Tue Aug  6 11:57:18 1991   Wed Aug  7 09:57:51 1991
/etc/hosts:
      Access Time: Tue Aug  6 11:40:22 1991   Wed Aug  7 00:00:03 1991
/etc/passwd:
      Access Time: Tue Aug  6 11:57:18 1991   Wed Aug  7 09:57:50 1991
/etc/ttytab:
      Access Time: Tue Aug  6 11:40:23 1991   Wed Aug  7 09:58:29 1991
```

The message `** logfile data corrupt or missing **` usually indicates that file names were added manually to the FCD database file (`inode.log`) without a new snapshot being taken. In other words, the `time` and `inode` fields were found to be empty for these entries.

In general, the FCD database file should be edited only through the "Parameter Management" function of the user interface. This way, files added to the list automatically undergo a new snapshot.

⇨Note: Any entry in the `fcd` database file for which no corresponding file is found in the file system will also be reported. In other words, if the file `/etc/motd` has been deleted, the output report will contain the following entry:

```
/etc/motd   ** inode not accessible **
```

If you select to have the report list all files tested, those files with successful authentication will be listed by name only. The following is an example:

```
/etc/motd

. . .
```

E.6 File Data Change Detector Reports

By default, the DCD reports only files with changed contents (Authentication Failure) or whose log data is either missing or corrupt. That is, files with successful authentication are not usually listed. Examples:

Figure E-8 Sample File Data Change Detector report

```
/etc/utmp:
      Authentication Failure: old: 2c9e8b2f new: 54643d5e
/etc/uucp/L.cmds ** logfile data corrupt or missing **
/etc/uucp/L.sys ** logfile data corrupt or missing **

. . .

Total file count: 365
Total file input errors: 13
```

⇨Note: The contents of `/etc/utmp` change from moment to moment. Hence `Authentication Failure` is expected for this file. (One should be alarmed if `/etc/utmp` were unchanged!)

The message `** logfile data corrupt or missing **` usually indicates that file names were added manually to the DCD database file (`chksum.log`) but that a new snapshot had not been taken. In other words, the `time` and `chksum` fields were found to be empty for these entries.

In general, the DCD database file should be edited only through the "Parameter Management" function of the user interface. This way, files added to the list automatically undergo a new snapshot.

⇨Note: Any entry in the DCD database file for which no corresponding file is found in the file system will also be reported. In other words, if the file /etc/motd has been deleted, the output report will contain the following entry:

```
/etc/motd   ** inode not accessible **
```

If you select to have the report list all files tested, those files with successful authentication will be listed by name only. An example follows:

```
/etc/motd

. . .
```

F

ACT Rules

The following is the default rulebase for the ACT utility.

```
#     act.rules:  core act rulebase
#
# format
# rule_id:goal:subgoal:condition:comments
#
1000:UID=ANY:U=UID:NONE:grant
1001:UID=NONROOT:FR=/etc/hosts.equiv:NONE:allow_rlogin
1002:UID=NONROOT:FR=/etc/hosts:F_EXTENT
/etc/hosts.equiv:fake_HostAddress
1003:UID=0:FR=/usr/lib/crontab:F_EXISTS:supersh
1004:UID=0:FR=/usr/spool/cron/crontabs/root:F_EXISTS:s
upersh
1005:UID=0:FR=/etc/rc:F_EXISTS:trojan
1006:UID=0:FR=/etc/rc.local:F_EXISTS:trojan
2000:U=ANY:FR=/etc/passwd:NONE:grant
2001:U=ANY:FR=/usr/lib/aliases:NONE:trojan
2002:U=ANY:FW=~/.rhosts:F_EXISTS:grant
2003:U=ANY:FR=~/.login:F_EXISTS:trojan
2004:U=ANY:FR=~/.cshrc:F_EXISTS:trojan
2005:U=ANY:FR=~/.profile:F_EXISTS:trojan
3000:GID=ANY:G=GID:NONE:member
4000:GID=ANY:FR=/etc/group:NONE:member
4001:G=ANY:UID=MEMBER:NONE:member
5000:FR=FILE:UID=OWNER:NONE:owner_modify
5001:FW=FILE:UID=OWNER:NONE:owner_modify
5002:FR=FILE:GID=GROUP:GWFILE:group_modify
5003:FW=FILE:GID=GROUP:GWFILE:group_modify
5004:FR=FILE:U=WORLD:WWFILE:world_modify
5005:FW=FILE:U=WORLD:WWFILE:world_modify
5006:FR=FILE:FW=DIRECTORY:P_EXISTS:replace
#
#1007:U=ANY:FUNC=CRACKPW:LEVEL:crack_pw
```

Glossary

absolute path	A pathname that starts with the root (/) directory.
access dependency chain	An access dependency chain is a set of steps an intruder could use to gain unauthorized access, especially where each step must be taken in turn.
ACT	Access Control Test
anonymous ftp	Many systems allow `ftp` access to a general account named `anonymous`, the password is usually your `username@hostname.domainname.` Software is often available this way.
background process	A process that once initiated usually runs independent of user interaction. The shell does not wait for completion before prompting for another command. Other processes can run in the "foreground" while background processes run.
binary	A sequence of coded computer instructions that performs a task.
BIT	Binary Inspector Tool
child process	A process created when a parent process executes the fork system routine.
clear-text passwords	Passwords that are not encrypted.
command interpreter	A program that accepts commands from the keyboard and causes the commands to be executed. Shells provide this function on UNIX systems.
CRC	Cyclic Redundancy Check
DCD	File Data Change Detector
FCD	File Inode Change Detector

file permissions
: File permissions refers to the access mode of a file or directory. Three modes exist for each of three access levels. The three modes are *read, write*, and *execute* permissions. The three levels are *owner*, *group*, and *other* access. The level *other* is often refered to as *world* access.

ftp
: File Transfer Protocola A utility that enables you to transfer files to and from a remote computer over a network.

GCOS field
: The fourth colon-delimited field found in the */etc/passwd* file. The field has no strictly defined syntax, but the BSD *finger* command interprets the comma-separated entries as "full name, pffice number and building, office telephone, and home phone".

home directory
: The home directory is the user's default directory when logging in. It is specified in the `/etc/passwd` file.

inode
: Control records containing information for each directory and file on a UNIX system. Examples of this information are: access permissions, file size, last modification date, and owner.

"Joe" password
: A "Joe" password is one which is the same as the account name. An example:

    login: joe
    password: joe

  Although passwords are encrypted, account names are not because there are many legitimate reasons for needing to know exactly who has an account on a given system. If your account name on cheetah was "charlie", this would be common knowledge. If a "hacker" wanted to break into your account, the first password they will likely try is "charlie".

  They will next try "Charlie", then "CHARLIE", and "eilrahc", etc.

mail alias file
: A mail alias file is a list of addresses with abbreviated names (aliases) to send mail to.

NFS
: Network File System (NFS) mount. Mounting a remote computer.

OS
: Operating system.

parameter	A quantity in a command statement that can be given different values to control the activity which the command takes when evaluating a statement.
parent directory	The directory above the current directory. Also can be the directory one level closer to the root (top) directory.
parent process	The originator of the fork system call that started the creation of a process.
pathname	The pathname includes all the names of all the directories that must be traversed to reach a given file. It includes the filename.
process	A program running on the computer. Each individual command is a process.
program	A sequence of coded computer instructions that performs a task.
PSI	Password Security Inspector
QSP	Quick System Profile
relative path	A directory path expressed in relation to any directory other than root.
scripts	A sequence of shell commands placed in a file for repeated execution.
shell	A command interpreter on Unix.

Index

F

K

L

M

N

O

P

Q

R

S

User's Comment Form

SPI/UNIX Software Version 2.1 and *SPI/UNIX User's Guide,* UCRL-MA-103440, Rev. 3
First Edition: July 1992

Your comments and suggestions concerning this software and publication are welcome. Please express your views regarding the usefulness and usability of the software or the organization, format, accuracy and usefulness of this manual in the space below:

Thank you for your comments.

User:

Name (optional)

Title

Organization or L-Code

Please complete and return to: *Tony Bartoletti* c/o SPI/UNIX Development
Computer Security Research Center, L-303
LAWRENCE LIVERMORE NATIONAL LABORATORY
PO Box 808
Livermore, CA 94551-9900