

Tru64 UNIX

Debugging with Audit

June 2000

This Best Practice describes how to use the audit subsystem as a tool to trace system calls for debugging purposes.

Contents

Debugging with Audit

Is This Best Practice Right for You?	1
Before You Begin	2
Familiarize Yourself with the Audit Subsystem	2
Build the Audit Subsystem into the Kernel	2
Applying the Best Practice	4
Tracing a Process	4
Reading Trace Data	6
Modifying the Kernel to Get More Data	8
System Calls That Do Not Always Get Audited	8
Verifying Success	9
Troubleshooting	9
Alternative Practices	9
Comments and Questions	10
Legal Notice	10

Debugging with Audit

This Best Practice describes how to use the audit subsystem as a powerful tool to trace system calls for debugging purposes. While you do not need to run the enhanced security option, you must build the audit subsystem into your kernel and configure the audit subsystem on your Tru64™ UNIX system. You do not have to load the C2 subsets.

See the Tru64 UNIX Best Practices Web page for more information about Best Practice documentation.

Is This Best Practice Right for You?

Not all Best Practices apply to all configurations, so you must be sure that it is appropriate for your system and circumstances. To use this Best Practice, you must meet the requirements described in the following table.

Requirement	Description
Operating System	Tru64 UNIX V5.0 or higher
System Configuration	The audit subsystem must be built into the running kernel.
Impact on Availability	If you must add the audit subsystem to your running kernel, you must reboot your system. Using audit to trace system calls does not significantly affect performance.
Audit Configuration	The audit subsystem must be configured on your system. Familiarity with audit and security principles makes the configuration easier. See the Tru64 UNIX <i>Security</i> guide and the audit reference pages for complete information about the audit subsystem.

If you do not meet the previous requirements, see *Alternative Practices* for information.

Before You Begin

Before you apply the Best Practice for Debugging with Audit, you must understand some background information and perform some preliminary tasks. You should learn about the audit subsystem by reading the audit subsystem chapter in the *Security* guide and reviewing the audit reference pages.

You need to determine the status of the audit subsystem on your HOST. To determine if audit is in the running kernel, enter the following:

```
# /usr/sbin/audgen test
```

The message “audgen: Function not implemented” indicates that the audit subsystem is not built in to the running kernel.

To determine if audit is currently running on the system, enter the following:

```
# ps x | grep audit
```

To check the configuration file to see if the audit option is selected for the kernel build, enter the following:

```
# grep DEC_AUDIT /sys/conf/HOSTNAME
options DEC_AUDIT
```

Where *HOSTNAME* is the name of the machine you are on in capital letters. The “DEC_AUDIT” string indicates that a kernel build will have the audit subsystem built in.

Familiarize Yourself with the Audit Subsystem

If you are not a regular user of the audit subsystem, you should read the following documents:

- The audit chapter of the *Security* guide
- The `auditconfig(8)`, `auditmask(8)`, `auditd(8)`, and `audit_tool(8)` reference pages

Build the Audit Subsystem into the Kernel

If a new kernel with the audit subsystem must be created, audit is enabled when the new kernel is booted. If you want to keep your existing kernel in place and run audit with a different kernel, do the following:

1. Create a kernel configuration file with the audit option:

```
# cp /sys/conf/HOSTNAME /sys/conf/AUDIT
```

2. Add the string "options DEC_AUDIT" to the configuration file.

```
# vi /sys/conf/AUDIT
```

3. Create a new kernel with audit while leaving the original kernel and configuration file in place:

```
# doconfig -c AUDIT
```

4. Answer "no" when asked if you want to edit the configuration file. The new kernel is located at `/sys/AUDIT/vmunix`.

5. Put the new kernel in place and boot the new kernel:

```
# mv /sys/AUDIT/vmunix /vmunix.audit
# shutdown -h now
>>> boot device -file /vmunix.audit
```

Where *device* is the disk with the root file system.

6. Configure and start audit as shown in the following procedures.

If the kernel already includes the audit subsystem, the `sysman auditconfig` utility is used to configure and start audit.

The `auditconfig` utility does the following:

- Establishes startup flags for the audit daemon
- Establishes startup flags for the audit mask
- Creates `/dev/audit` (if needed)
- Configures a new kernel (if needed)

Startup flags are stored in the `/etc/rc.config.common` file. You should select all the defaults, with the following exceptions:

- When asked to create `/var/audit`, choose "y".
- When asked to choose the action to take for an overflow, choose to overwrite the current audit log.
- When prompted for an event list, press the Enter key to have no events audited.

The audit subsystem is activated in two steps:

1. Start the audit daemon (`auditd`) using one of the following methods:
 - Use the `auditconfig` utility to enable audit if the kernel includes the audit subsystem.

- Use the `init` scripts to enable audit according to the flags in the `rc.config.common` file previously set by `auditconfig`:

```
# /sbin/init.d/audit start
```

- Manually start the audit daemon:

```
# auditd -qf FILE -o o
```

Where the options perform the following functions:

- l Collect data in `./FILE.nnn`.
- q Return the location of the log file.
- o o Overwrite log on overflow condition.

2. Specify what events are to be audited (using `auditmask`). When tracing system calls, no events are audited by default.

To stop the collection of audit data, either clear the audit mask for the system or any process whose audit mask you set using the `auditmask -n` command, or turn off the audit daemon using the `auditd -k` command. See the `auditd(8)` reference page for further information.

Applying the Best Practice

Before you apply this Best Practice, be sure to follow the recommendations in *Before You Begin*.

Tracing a Process

The `auditmask` utility can be used to adjust the audit characteristics of the system, of a single process, or all processes associated with a user's audit ID (AUID). The generation of an audit record is a function of the system audit mask, the process audit mask, and the process `audcntl` flag. Each audit mask is a bitmap for all the events.

The process `audcntl` flag specifies:

or Audit if the event is in either the system or the process audit mask. The default `audcntl` flag setting is `or`.

and Audit if the event is in both the system and the process audit mask.

`off` No auditing done for this process.

`usr` Audit if the event is in the process audit mask.

Specifying which events are audited can be done by naming a set of system calls and/or alias names. Aliases are defined in (and may be added to) the `/etc/sec/event_aliases` file. You can edit this file to add or delete system calls. An optional extension can be used to distinguish between successful occurrences and/or failed occurrences of any event. For example:

`open:1:0` Specifies successful occurrences of the `open()` call.

`open:0:1` Specifies failed occurrences of the `open()` call.

The following examples demonstrate the use of the `auditmask` utility for various purposes. These examples modify the process audit mask. Unless specified, the process `audcntl` flag remains at its default setting of `or`.

- To execute command arguments and audit everything for the newly created process:

```
# auditmask -E command args
```
- To execute command arguments, audit failed `open()` system calls, and successful `ipc` events (defined in `/etc/sec/event_aliases`) for the newly created process:

```
# auditmask open:0:1 ipc:1:0 -e command args
```
- For process ID (PID) 999, audit all (`-f`) events except `gettimeofday()`:

```
# auditmask -p 999 -f gettimeofday:0:0
```
- For PID 999, audit no (`-n`) events:

```
# auditmask -p 999 -n
```
- For PID 999, audit `open()` and `exec()` events:

```
# auditmask -p 999 open exec
```
- For PID 999, add `exit()` to the set of events being audited:

```
# auditmask -p 999 exit
```
- For PID 999, get the set of events being audited:

```
# auditmask -p 999
```
- For PID 999, set the `audcntl` flag to `usr`:

```
# auditmask -p 999 -c usr
```

When tracing a running process, the `auditmask -c usr` command traces all options for the system calls.

- For all processes owned by the user with AUID 1123, audit all `ipc` events (the AUID is the same as the user's initial RUID):

```
# auditmask -a 1123 ipc
```

- To access the `auditmask` help option:

```
# auditmask -h
```

See the `auditmask(8)` reference page for further information.

Reading Trace Data

To read the trace data, use the following procedure:

1. Use the `auditd -d` command to flush any buffered audit data:

```
# auditd -dq  
alpha1.005
```

Where the additional `-q` option displays the name of the current log file.

2. Examine the log file with the `audit_tool` utility:

```
# auditmask -E date  
Fri Nov 26 19:17:52 EST 1999
```

```
# audit_tool 'auditd -dq' -B
```

AUID:RUID:EUID	PID	RES/(ERR)	EVENT
1123:0:0	6691	0x14	audcntl (0x7 0x0 0x14 0x0)
1123:0:0	6691	0x0	execve (/sbin/date date)
1123:0:0	6691	0x2000	getpagesize ()
1123:0:0	6691	0x2000	getpagesize ()
1123:0:0	6691	0x0	getrlimit ()
1123:0:0	6691	0x3ffc0004000	mmap (-1 0x7 0x3ffc0004000 0x12 0x2000)
1123:0:0	6691	0x0	getrlimit ()
1123:0:0	6691	0x3ffc0006000	mmap (-1 0x7 0x3ffc0006000 0x12 0x4000)
1123:0:0	6691	0x0	getuid ()
1123:0:0	6691	0x1	getgid ()
1123:0:0	6691	0xa68	read (3)
1123:0:0	6691	0x120000000	mmap (3 0x5 0x120000000 0x102 0x4000)
1123:0:0	6691	0x140000000	mmap (3 0x7 0x140000000 0x2 0x2000)
1123:0:0	6691	0x4	open (/shlib/libc.so 0x0)
1123:0:0	6691	0xa68	read (/shlib/libc.so)
1123:0:0	6691	0x3ff80080000	mmap (/shlib/libc.so 0x5 0x3ff80080000 0x2 0x10e000)
1123:0:0	6691	0x3ffc0080000	mmap (/shlib/libc.so 0x7 0x3ffc0080000 0x2 0x10000)
1123:0:0	6691	0x3ffc0090000	mmap (-1 0x7 0x3ffc0090000 0x12 0x9bf0)
1123:0:0	6691	0x0	close (4)
1123:0:0	6691	0x0	stat (/shlib/libc.so)
1123:0:0	6691	0x0	set_program_attributes ()
1123:0:0	6691	0x0	close (3)
1123:0:0	6691	0x2000	getpagesize ()

```

1123:0:0      6691 0x0      obreak ( 0x14000ea10 )
1123:0:0      6691 0x0      gettimeofday ( )
1123:0:0      6691 0x3      open ( /etc/zoneinfo/localtime 0x0 )
1123:0:0      6691 0x32e    read ( /etc/zoneinfo/localtime )
1123:0:0      6691 0x0      close ( 3 )
1123:0:0      6691 0x1d    write ( 1 )
1123:0:0      6691 0x0      close ( 0 )
1123:0:0      6691 0x0      close ( 1 )
1123:0:0      6691 0x0      close ( 2 )
1123:0:0      6691 0x0      exit ( )

```

3. Examine the exec of the date command:

```

# audit_tool 'auditd -dq' -e exec

audit_id: 1123      ruid/euid: 0/0
pid: 6691      ppid: 6688      ttydev: (6,7)
event: execve
char param: /sbin/date
char param: date
inode id: 7390      inode dev: (8,16384) [regular file]
object mode: 0755
result: 0
ip address: 16.140.128.241 (alpha1.zzk.abc.com)
timestamp: Fri Nov 26 19:17:52.77 1999 EST

```

The following additional `audit_tool` options are available to help select data of interest.

- Selection options:

```

-a audit_id      -e event[.subevent][:succeed:fail]
-E error# or error_string  -h hostname or ip address
-p [-]pid      -P ppid
-r real_uid      -s string_parameter
-t start_time    -T end_time      format: yymmdd[hh[mm[ss]]]
-u uid          -U username
-v inode_id      -V inode's device-major#,minor#
-x device-major#,minor#  -y procname
-/ search-string

```

- Control options:

```

-b      Output in binary format
-B      Output in abbreviated format
-d file  Use specified deselection rules file (-D to print ruleset)
-f      Keep reading auditlog (like tail -f)
-F      Fast mode; no state data maintained
-i      Interactive selection mode
-o      Override switching log file due to change_auditlog records
-O format  Output in specified format
        {cpu,usec,time,username,userid,ppid,res,event}
-Q      Suppress progress messages
-R [name]  Generate reports by audit_id
-S      Sort audit records by time (for SMP only)
-w      Map ruid, group #'s to names using passwd, group tables
-Z      Display statistics for selected events

```

See the `audit_tool(8)` reference page for further information.

Modifying the Kernel to Get More Data

Along with the system call name, the result, error, time stamp, ID info, and various arguments passed to the system call are recorded. Only the arguments of interest from a security perspective are recorded. If additional arguments are required, you can use `dbx` to change which arguments get recorded for any system call. For example, `flock()` is system call 131 and takes as arguments a file descriptor and an option number. To audit these arguments, enter the following:

```
# dbx <object>
(dbx) a sysent[131].aud_param[0]='c'
99
(dbx) a sysent[131].aud_param[1]='a'
97
```

Where `c` indicates a file descriptor will be recorded and `a` indicates an integer argument will be recorded.

The set of encodings is described in the `<sys/audit.h>` file.

Encoding changes that might provide useful information to other software developers can be added to the `syscalls.master` file.

System Calls That Do Not Always Get Audited

Not every instance of each system call generates audit data. All instances of system calls not in the following table generate audit data. The conditions under which a particular system call does not generate an audit record are as follows:

System Call	No Audit Record If:
<code>close()</code>	EBADF failures
<code>dup()</code>	Failures from <code>getf()</code>
<code>execv()</code> , <code>execve()</code> , <code>exec_with_loader()</code>	Failures triggered by failed <code>namei()</code> lookups, failures to terminate threads, aborts from handler callouts
<code>fcntl()*</code>	Failures from <code>getf()</code>
<code>ioctl()*</code>	Failures from <code>getf()</code>
<code>msfs_syscall()</code>	Any system call failure
<code>prctlset()</code>	Failed input tests; calls other than those which modify another process.

System Call	No Audit Record If:
proplist_syscall()	Failures on copyin() of system call arguments
reboot()	Successful reboot()
security()	getluid option
swapctl()	Any call other than successful SC_ADD option
uadmin()	Any call other than a failed A_REBOOT or A_SHUTDOWN

* System calls that typically generate audit data only for security-relevant options. When executing processes from auditmask with the -cd e or -E option, however, all options generate audit data.

Verifying Success

After you apply the Best Practice for Debugging with Audit, success is verified by the usefulness of the trace data.

Troubleshooting

If you determine that the Best Practice was not successful, as described in *Verifying Success*, use the following table to identify and solve problems.

Problem	Possible Solutions
Audit is not running on the system.	The audit subsystem is not built in to the running kernel.
Audit is not running on the system.	The kernel with the audit subsystem is not running. Reboot the system.
Audit is running, but data is not being collected.	The audit mask is set improperly.

Alternative Practices

This Best Practice is an alternative to using the tracing features of dbx or other debuggers. Use this procedure if more detailed trace data is needed than the regular debuggers can provide.

See the programming documentation for traditional methods of tracing system calls.

Comments and Questions

We value your comments and questions on the information in this document. Please mail your comments to us at this address:

best_practices@zk3.dec.com

Legal Notice

COMPAQ and the Compaq logo are registered in the U.S. Patent and Trademark Office.

Confidential computer software. Valid license from Compaq required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendors standard commercial license.

Compaq shall not be liable for technical or editorial errors or omissions contained herein. The information in this publication is subject to change without notice and is provided "as is" without warranty of any kind. The entire risk arising out of the use of this information remains with recipient. In no event shall Compaq be liable for any direct, consequential, incidental, special, punitive, or other damages whatsoever (including without limitation, damages for loss of business profits, business interruption or loss of business information), even if Compaq has been advised of the possibility of such damages. The foregoing shall apply regardless of the negligence or other fault of either party and regardless of whether such liability sounds in contract, negligence, tort, or any other theory of legal liability, and notwithstanding any failure of essential purpose of any limited remedy.

The limited warranties for Compaq products are exclusively set forth in the documentation accompanying such products. Nothing herein should be construed as constituting a further or additional warranty.