

```

*****
5793 Mon Aug 29 10:16:14 2016
new/usr/src/man/man1/psecflags.1
sync further changes from uts/aslr
*****
1 \" te
2.\" This file and its contents are supplied under the terms of the
3.\" Common Development and Distribution License (\"CDDL\"), version 1.0.
4.\" You may only use this file in accordance with the terms of version
5.\" 1.0 of the CDDL.
6.\"
7.\" A full copy of the text of the CDDL should have accompanied this
8.\" source. A copy of the CDDL is also available via the Internet at
9.\" http://www.illumos.org/license/CDDL.
10.\"
11.\" Copyright 2015, Richard Lowe.
12.\"
13.TH \"PSECFLAGS\" \"1\" \"June 6, 2016\"
14.SH \"NAME\"
15 \fBpsecflags\fR - inspect or modify process security flags
16.SH \"SYNOPSIS\"
17.LP
18.nf
19 \fB/usr/bin/psecflags\fR \fI-s\fR \fI-spec\fR \fI-e\fR \fI-command\fR \
20 [\fIarg\fR]...
21.fi
22.LP
23.nf
24 \fB/usr/bin/psecflags\fR \fI-s\fR \fI-spec\fR [\fI-i\fR \fI-idtype\fR] \
25 \fI-id\fR ...
26.fi
27.LP
28.nf
29 \fB/usr/bin/psecflags\fR [\fI-F\fR] { \fIpid\fR | \fIcore\fR }
30.fi
31.LP
32.nf
33 \fB/usr/bin/psecflags\fR \fI-l\fR
34.fi

36.SH \"DESCRIPTION\"
37 The first invocation of the \fBpsecflags\fR command runs the specified
38 \fI-command\fR with the security-flags modified as described by the \fI-s\fR
39 argument.
40 .P
41 The second invocation modifies the security-flags of the processes described
42 by \fI-idtype\fR and \fI-id\fR according as described by the \fI-s\fR argument.
43 .P
44 The third invocation describes the security-flags of the specified processes
45 or core files. The effective set is signified by '\fBE\fR', the inheritable
46 set by '\fBI\fR', the lower set by '\fBL\fR', and the upper set by '\fBU\fR'.
47 .P
48 The fourth invocation lists the supported process security-flags, documented
49 in \fBsecurity-flags\fR(5).

51.SH \"OPTIONS\"
52 The following options are supported:
53 .sp
54 .ne 2
55 .na
56 \fB-e\fR
57 .ad
58 .RS 11n
59 Interpret the remaining arguments as a command line and run the command with
60 the security-flags specified with the \fI-s\fR flag.
61 .RE

```

```

63 .sp
64 .ne 2
65 .na
66 \fB-F\fR
67 .ad
68 .RS 11n
69 Force. Grab the target process even if another process has control.
70 .RE

72 .sp
73 .ne 2
74 .na
75 \fB-i\fR \fI-idtype\fR
76 .ad
77 .RS 11n
78 This option, together with the \fI-id\fR arguments specify one or more
79 processes whose security-flags will be modified. The interpretation of the
80 \fI-id\fR arguments is based on \fI-idtype\fR. If \fI-idtype\fR is omitted the
81 default is \fBpid\fR.

83 Valid \fI-idtype\fR options are:
84 .sp
85 .ne 2
86 .na
87 \fBball\fR
88 .ad
89 .RS 11n
90 The \fBpsecflags\fR command applies to all processes
91 .RE

93 .sp
94 .ne 2
95 .na
96 \fBcontract\fR, \fBctid\fR
97 .ad
98 .RS 11n
99 The security-flags of any process with a contract ID matching the \fI-id\fR
100 arguments are modified.
101 .RE

103 .sp
104 .ne 2
105 .na
106 \fBgroup\fR, \fBgid\fR
107 .ad
108 .RS 11n
109 The security-flags of any process with a group ID matching the \fI-id\fR
110 arguments are modified.
111 .RE

113 .sp
114 .ne 2
115 .na
116 \fBpid\fR
117 .ad
118 .RS 11n
119 The security-flags of any process with a process ID matching the \fI-id\fR
120 arguments are modified. This is the default.
121 .RE

123 .sp
124 .ne 2
125 .na
126 \fBppid\fR
127 .ad

```

```

128 .RS 11n
129 The security-flags of any processes whose parent process ID matches the
130 \fIid\fR arguments are modified.
131 .RE

133 .sp
134 .ne 2
135 .na
136 \fBproject\fR, \fBprojid\fR
137 .ad
138 .RS 11n
139 The security-flags of any process whose project ID matches the \fIid\fR
140 arguments are modified.
141 .RE

143 .sp
144 .ne 2
145 .na
146 \fBsession\fR, \fBsid\fR
147 .ad
148 .RS 11n
149 The security-flags of any process whose session ID matches the \fIid\fR
150 arguments are modified.
151 .RE

153 .sp
154 .ne 2
155 .na
156 \fBtaskid\fR
157 .ad
158 .RS 11n
159 The security-flags of any process whose task ID matches the \fIid\fR arguments
160 are modified.
161 .RE

163 .sp
164 .ne 2
165 .na
166 \fBuser\fR, \fBuid\fR
167 .ad
168 .RS 11n
169 The security-flags of any process belonging to the users matching the \fIid\fR
170 arguments are modified.
171 .RE

173 .sp
174 .ne 2
175 .na
176 \fBzone\fR, \fBzoneid\fR
177 .ad
178 .RS 11n
179 The security-flags of any process running in the zones matching the given
180 \fIid\fR arguments are modified.
181 .RE
182 .RE

184 .sp
185 .ne 2
186 .na
187 \fB-l\fR
188 .ad
189 .RS 11n
190 List all supported process security-flags, described in
191 \fBsecurity-flags\fR(5).
192 .RE

```

```

194 .sp
195 .ne 2
196 .na
197 \fB-s\fR \fIspecification\fR
198 .ad
199 .RS 11n
200 Modify the process security-flags according to
201 \fIspecification\fR. Specifications take the form of a comma-separated list of
202 flags, optionally preceded by a '-' or '!'. Where '-' and '!' indicate that the
203 given flag should be removed from the specification. The pseudo-flags "all",
204 "none" and "current" are supported, to indicate that all flags, no flags, or
205 the current set of flags (respectively) are to be included.
206 .P
207 By default, the inheritable flags are changed. You may optionally specify the
208 set to change using their single-letter identifiers and an equals sign.
209 .P
210 For a list of valid security-flags, see \fBpsecflags -l\fR.
211 .RE

213 .SH "EXAMPLES"
214 .LP
215 \fBExample 1\fR Display the security-flags of the current shell.
216 .sp
217 .in +2
218 .nf
219 example$ \fBpsecflags $$\fR
220 100718: -sh
221      E:      aslr
222      I:      aslr
223      L:      none
224      U:      aslr,forbidnullmap,noexecstack
225 .fi
226 .in -2
227 .sp

229 .LP
230 \fBExample 2\fR Run a user command with ASLR enabled in addition to any
231 inherited security flags.
232 .sp
233 .in +2
234 .nf
235 example$ \fBpsecflags -s current,aslr -e /bin/sh\fR
236 $ psecflags $$
237 100724: -sh
238      E:      none
239      I:      aslr
240      L:      none
241      U:      aslr,forbidnullmap,noexecstack
242 .fi
243 .in -2
244 .sp

246 .LP
247 \fBExample 3\fR Remove aslr from the inheritable flags of all Bob's processes.
248 .sp
249 .in +2
250 .nf
251 example# \fBpsecflags -s current,-aslr -i uid bob\fR
252 .fi
253 .in -2

255 .LP
256 \fBExample 4\fR Add the aslr flag to the lower set, so that all future
257 child processes must have this flag set.
258 .sp
259 .in +2

```

```
260 .nf
261 example# \fBpsecflags -s L=current,aslr $$\fR
262 .fi
263 .in -2
```

```
265 .SH "EXIT STATUS"
266 The following exit values are returned:
```

```
268 .TP
269 \fB0\fR
270 .IP
271 Success.
```

```
273 .TP
274 \fBnon-zero\fR
275 .IP
276 An error has occurred.
276 An error has occurred.
```

```
278 .SH "ATTRIBUTES"
279 .LP
280 See \fBattributes\fR(5) for descriptions of the following attributes:
281 .sp
```

```
283 .sp
284 .TS
285 box:
286 c | c
287 l | l .
288 ATTRIBUTE TYPE ATTRIBUTE VALUE
289 _
290 Interface Stability Volatile
291 .TE
```

```
293 .SH "SEE ALSO"
294 .BR exec (2),
295 .BR attributes (5),
296 .BR contract (4),
297 .BR security-flags (5),
298 .BR zones (5)
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_core.sh
```

1

```
*****
```

```
1301 Mon Aug 29 10:16:16 2016
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_core.sh
```

```
sync further changes from uts/aslr
```

```
*****
```

```
_____unchanged_portion_omitted_____
```

```
31 trap cleanup EXIT
```

```
33 ## gcore-produced core
```

```
34 gcore $pid >/dev/null
```

```
36 cat > gcore-expected.$$ <<EOF
```

```
37 core 'core.$pid' of $pid:      sleep 100000
```

```
38     E:      aslr
```

```
39     I:      aslr
```

```
40 EOF
```

```
42 /usr/bin/psecflags core.${pid} | grep -v '[LU]:' > gcore-output.$$
```

```
44 if ! diff -u gcore-expected.$$ gcore-output.$$; then
```

```
45     exit 1;
```

```
46 fi
```

```
48 ## kernel-produced core
```

```
49 kill -SEGV $pid
```

```
50 wait $pid >/dev/null 2>&1
```

```
51 #endif /* !codereview */
```

```
53 cat > core-expected.$$ <<EOF
```

```
54 core 'core' of $pid:      sleep 100000
```

```
55     E:      aslr
```

```
56     I:      aslr
```

```
57 EOF
```

```
59 /usr/bin/psecflags core | grep -v '[LU]:' > core-output.$$
```

```
61 if ! diff -u core-expected.$$ core-output.$$; then
```

```
62     exit 1;
```

```
63 fi
```

```
65 exit 0
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_dts.sh 1
*****
1728 Mon Aug 29 10:16:17 2016
new/usr/src/test/os-tests/tests/secflags/secflags_dts.sh
sync further changes from uts/aslr
*****
_unchanged_portion_omitted_
37 EOF

39 gcc -o tester-aslr tester.c -Wl,-z,aslr=enabled
40 gcc -o tester-noaslr tester.c -Wl,-z,aslr=disabled

42 # This is the easiest way I've found to get many many DTs, but it's gross
43 gcc -o many-dts-aslr tester.c -Wl,-z,aslr=enabled $(for elt in /usr/lib/lib*.so;
44 gcc -o many-dts-noaslr tester.c -Wl,-z,aslr=disabled $(for elt in /usr/lib/lib*.

46 check() {
47     bin=$1
48     state=$2
49     set=$3
50 #endif /* ! codereview */
51     ret=0

53     $bin &
54     pid=$!
55     psecflags $pid | grep -q "${set}::*aslr"
49     psecflags $pid | grep -q 'E::*aslr'
56     (( $? != $state )) && ret=1
57     kill -9 $pid
58     return $ret
59 }

_unchanged_portion_omitted_

66 psecflags -s none $$
67 check ./tester-aslr 0 E || fail "DT_SUNW_ASLR 1 failed"
68 check ./many-dts-aslr 0 E || fail "DT_SUNW_ASLR 1 with many DTs failed"
69 check ./tester-aslr 1 I || fail "DT_SUNW_ASLR 1 incorrectly set the inheritable"
61 check ./tester-aslr 0 || fail "DT_SUNW_ASLR 1 failed"
62 check ./many-dts-aslr 0 || fail "DT_SUNW_ASLR 1 with many DTs failed"

71 psecflags -s aslr $$
72 check ./tester-noaslr 1 E || fail "DT_SUNW_ASLR 0 failed"
73 check ./many-dts-noaslr 1 E || fail "DT_SUNW_ASLR 0 with many DTs failed"
65 check ./tester-noaslr 1 || fail "DT_SUNW_ASLR 0 failed"
66 check ./many-dts-noaslr 1 || fail "DT_SUNW_ASLR 0 with many DTs failed"
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_elfdump.sh
```

1

```
*****  
1780 Mon Aug 29 10:16:18 2016  
new/usr/src/test/os-tests/tests/secflags/secflags_elfdump.sh  
sync further changes from uts/aslr  
*****  
_____unchanged_portion_omitted_____
```

```
31 trap cleanup EXIT  
  
33 ## gcore-produced core  
34 gcore $pid >/dev/null  
  
36 cat > gcore-expected.$$ <<EOF  
37 namesz: 0x5  
38 descsz: 0x28  
39 type: [ NT_SECFLAGS ]  
40 name:  
41 CORE\0  
42 desc: (prsecflags_t)  
43 pr_version: 1  
44 pr_effective: [ ASLR ]  
45 pr_inherit: [ ASLR ]  
46 pr_lower: 0  
47 pr_upper: [ ASLR FORBIDNULLMAP NOEXECSTACK ]  
48 EOF  
  
50 /usr/bin/elfdump -n core.${pid} | grep -B5 -A5 prsecflags_t > gcore-output.$$  
  
52 if ! diff -u gcore-expected.$$ gcore-output.$$; then  
53 exit 1;  
54 fi  
  
56 ## kernel-produced core  
57 kill -SEGV $pid  
58 wait $pid >/dev/null 2>&1  
59 #endif /* ! codereview */  
  
61 cat > core-expected.$$ <<EOF  
62 namesz: 0x5  
63 descsz: 0x28  
64 type: [ NT_SECFLAGS ]  
65 name:  
66 CORE\0  
67 desc: (prsecflags_t)  
68 pr_version: 1  
69 pr_effective: [ ASLR ]  
70 pr_inherit: [ ASLR ]  
71 pr_lower: 0  
72 pr_upper: [ ASLR FORBIDNULLMAP NOEXECSTACK ]  
73 EOF  
  
75 /usr/bin/elfdump -n core | grep -B5 -A5 prsecflags_t > core-output.$$  
  
77 if ! diff -u core-expected.$$ core-output.$$; then  
78 exit 1;  
79 fi  
  
81 exit 0
```

59159 Mon Aug 29 10:16:19 2016

new/usr/src/uts/common/exec/elf/elf.c

sync further changes from uts/aslr

unchanged portion omitted

```
168 static int
169 handle_secflag_dt(proc_t *p, uint_t dt, uint_t val)
170 {
171     uint_t flag;
172
173     switch (dt) {
174     case DT_SUNW_ASLR:
175         flag = PROC_SEC_ASLR;
176         break;
177     default:
178         return (EINVAL);
179     }
180
181     if (val == 0) {
182         if (secflag_isset(p->p_secflags.psf_lower, flag))
183             return (EPERM);
184         if ((secpolicy_psecflags(CRED(), p, p) != 0) &&
185             secflag_isset(p->p_secflags.psf_inherit, flag))
186             return (EPERM);
187
188         secflag_clear(&p->p_secflags.psf_inherit, flag);
188         secflag_clear(&p->p_secflags.psf_effective, flag);
189     } else {
190         if (!secflag_isset(p->p_secflags.psf_upper, flag))
191             return (EPERM);
192
193         if ((secpolicy_psecflags(CRED(), p, p) != 0) &&
194             !secflag_isset(p->p_secflags.psf_inherit, flag))
195             return (EPERM);
196
197         secflag_set(&p->p_secflags.psf_inherit, flag);
197         secflag_set(&p->p_secflags.psf_effective, flag);
198     }
199
200     return (0);
201 }
```

unchanged portion omitted

```

*****
26807 Mon Aug 29 10:16:20 2016
new/usr/src/uts/common/os/grow.c
sync further changes from uts/aslr
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /* Copyright 2013 OmniTI Computer Consulting, Inc. All rights reserved. */

24 /*
25  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
26  * Use is subject to license terms.
27  */

29 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
30 /*      All Rights Reserved      */

32 #include <sys/types.h>
33 #include <sys/inttypes.h>
34 #include <sys/param.h>
35 #include <sys/sysmacros.h>
36 #include <sys/system.h>
37 #include <sys/signal.h>
38 #include <sys/user.h>
39 #include <sys/errno.h>
40 #include <sys/var.h>
41 #include <sys/proc.h>
42 #include <sys/tuneable.h>
43 #include <sys/debug.h>
44 #include <sys/cmn_err.h>
45 #include <sys/cred.h>
46 #include <sys/vnode.h>
47 #include <sys/vfs.h>
48 #include <sys/vm.h>
49 #include <sys/file.h>
50 #include <sys/mman.h>
51 #include <sys/vmparam.h>
52 #include <sys/fcntl.h>
53 #include <sys/lwpchan_impl.h>
54 #include <sys/nbmlck.h>

56 #include <vm/hat.h>
57 #include <vm/as.h>
58 #include <vm/seg.h>
59 #include <vm/seg_dev.h>
60 #include <vm/seg_vn.h>

```

```

62 int use_brk_lpg = 1;
63 int use_stk_lpg = 1;

65 /*
66  * If set, we will not randomize mappings where the 'addr' argument is
67  * non-NULL and not an alignment.
68  */
69 int aslr_respect_mmap_hint = 0;

71 #endif /* ! codereview */
72 static int brk_lpg(caddr_t nva);
73 static int grow_lpg(caddr_t sp);

75 intptr_t
76 brk(caddr_t nva)
77 {
78     int error;
79     proc_t *p = curproc;

81     /*
82      * Serialize brk operations on an address space.
83      * This also serves as the lock protecting p_brksize
84      * and p_brkpageszc.
85      */
86     as_rangelock(p->p_as);

88     /*
89     #endif /* ! codereview */
90     * As a special case to aid the implementation of sbrk(3C), if given a
91     * new brk of 0, return the current brk. We'll hide this in brk(3C).
92     */
93     if (nva == 0) {
94         as_rangeunlock(p->p_as);
95         if (nva == 0)
96             return ((intptr_t)(p->p_brkbase + p->p_brksize));
97     }
98 #endif /* ! codereview */

99     /*
100    * Serialize brk operations on an address space.
101    * This also serves as the lock protecting p_brksize
102    * and p_brkpageszc.
103    */
104    as_rangelock(p->p_as);
105    if (use_brk_lpg && (p->p_flag & SAUTOLPG) != 0) {
106        error = brk_lpg(nva);
107    } else {
108        error = brk_internal(nva, p->p_brkpageszc);
109    }
110    as_rangeunlock(p->p_as);
111    return ((error != 0 ? set_errno(error) : 0));
112 }
113 #endif /* ! codereview */

114 #define RANDOMIZABLE_MAPPING(addr, flags) (((flags & MAP_FIXED) == 0) && \
115     !(((flags & MAP_ALIGN) == 0) && (addr != 0) && aslr_respect_mmap_hint))

116 #endif /* ! codereview */
117 static int
118 smmap_common(caddr_t *addrp, size_t len,
119     int prot, int flags, struct file *fp, offset_t pos)
120 {
121     struct vnode *vp;
122     struct as *as = curproc->p_as;
123     uint_t uprot, maxprot, type;
124     int error;

```



```

622     int in_crit = 0;

624     if ((flags & ~(MAP_SHARED | MAP_PRIVATE | MAP_FIXED | _MAP_NEW |
625         _MAP_LOW32 | MAP_NORESERVE | MAP_ANON | MAP_ALIGN |
626         MAP_TEXT | MAP_INITDATA)) != 0) {
627         /* | MAP_RENAME */ /* not implemented, let user know */
628         return (EINVAL);
629     }

631     if ((flags & MAP_TEXT) && !(prot & PROT_EXEC)) {
632         return (EINVAL);
633     }

635     if ((flags & (MAP_TEXT | MAP_INITDATA)) == (MAP_TEXT | MAP_INITDATA)) {
636         return (EINVAL);
637     }

639     if ((flags & (MAP_FIXED | _MAP_RANDOMIZE)) ==
640         (MAP_FIXED | _MAP_RANDOMIZE)) {
641         return (EINVAL);
642     }

644     /*
645     * If it's not a fixed allocation and mmap ASLR is enabled, randomize
646     * it.
647     */
648     if (RANDOMIZABLE_MAPPING(*addrp, flags) &&
649         if ((flags & MAP_FIXED) == 0) &&
650         secflag_enabled(curproc, PROC_SEC_ASLR))
651         flags |= _MAP_RANDOMIZE;

652 #if defined(__sparc)
653     /*
654     * See if this is an "old mmap call". If so, remember this
655     * fact and convert the flags value given to mmap to indicate
656     * the specified address in the system call must be used.
657     * _MAP_NEW is turned set by all new uses of mmap.
658     */
659     if ((flags & _MAP_NEW) == 0)
660         flags |= MAP_FIXED;
661 #endif
662     flags &= ~_MAP_NEW;

664     type = flags & MAP_TYPE;
665     if (type != MAP_PRIVATE && type != MAP_SHARED)
666         return (EINVAL);

669     if (flags & MAP_ALIGN) {
670         if (flags & MAP_FIXED)
671             return (EINVAL);

673         /* alignment needs to be a power of 2 >= page size */
674         if (((uintptr_t)*addrp < PAGE_SIZE && (uintptr_t)*addrp != 0) ||
675             !ISP2((uintptr_t)*addrp))
676             return (EINVAL);
677     }
678     /*
679     * Check for bad lengths and file position.
680     * We let the VOP_MAP routine check for negative lengths
681     * since on some vnode types this might be appropriate.
682     */
683     if (len == 0 || (pos & (u_offset_t)PAGEOFFSET) != 0)
684         return (EINVAL);

686     maxprot = PROT_ALL; /* start out allowing all accesses */

```

```

687     uprot = prot | PROT_USER;

689     if (fp == NULL) {
690         ASSERT(flags & MAP_ANON);
691         /* discard lwpchan mappings, like munmap() */
692         if ((flags & MAP_FIXED) && curproc->p_lwp != NULL)
693             lwpchan_delete_mapping(curproc, *addrp, *addrp + len);
694         as_rangelock(as);
695         error = zmap(as, addrp, len, uprot, flags, pos);
696         as_rangeunlock(as);
697         /*
698         * Tell machine specific code that lwp has mapped shared memory
699         */
700         if (error == 0 && (flags & MAP_SHARED)) {
701             /* EMPTY */
702             LWP_MMODEL_SHARED_AS(*addrp, len);
703         }
704         return (error);
705     } else if ((flags & MAP_ANON) != 0)
706         return (EINVAL);

708     vp = fp->f_vnode;

710     /* Can't execute code from "noexec" mounted filesystem. */
711     if ((vp->v_vfsp->vfs_flag & VFS_NOEXEC) != 0)
712         maxprot &= ~PROT_EXEC;

714     /*
715     * These checks were added as part of large files.
716     *
717     * Return ENXIO if the initial position is negative; return EOVERFLOW
718     * if (offset + len) would overflow the maximum allowed offset for the
719     * type of file descriptor being used.
720     */
721     if (vp->v_type == VREG) {
722         if (pos < 0)
723             return (ENXIO);
724         if ((offset_t)len > (OFFSET_MAX(fp) - pos))
725             return (EOVERFLOW);
726     }

728     if (type == MAP_SHARED && (fp->f_flag & FWRITE) == 0) {
729         /* no write access allowed */
730         maxprot &= ~PROT_WRITE;
731     }

733     /*
734     * XXX - Do we also adjust maxprot based on protections
735     * of the vnode? E.g. if no execute permission is given
736     * on the vnode for the current user, maxprot probably
737     * should disallow PROT_EXEC also? This is different
738     * from the write access as this would be a per vnode
739     * test as opposed to a per fd test for writability.
740     */

742     /*
743     * Verify that the specified protections are not greater than
744     * the maximum allowable protections. Also test to make sure
745     * that the file descriptor does allow for read access since
746     * "write only" mappings are hard to do since normally we do
747     * the read from the file before the page can be written.
748     */
749     if (((maxprot & uprot) != uprot) || (fp->f_flag & FREAD) == 0)
750         return (EACCES);

752     /*

```

```

753     * If the user specified an address, do some simple checks here
754     */
755     if ((flags & MAP_FIXED) != 0) {
756         caddr_t userlimit;
757
758         /*
759          * Use the user address. First verify that
760          * the address to be used is page aligned.
761          * Then make some simple bounds checks.
762          */
763         if (((uintptr_t)*addrp & PAGEOFFSET) != 0)
764             return (EINVAL);
765
766         userlimit = flags & _MAP_LOW32 ?
767             (caddr_t)USERLIMIT32 : as->a_userlimit;
768         switch (valid_usr_range(*addrp, len, uprot, as, userlimit)) {
769             case RANGE_OKAY:
770                 break;
771             case RANGE_BADPROT:
772                 return (ENOTSUP);
773             case RANGE_BADADDR:
774             default:
775                 return (ENOMEM);
776         }
777     }
778
779     if ((prot & (PROT_READ | PROT_WRITE | PROT_EXEC)) &&
780         nbl_need_check(vp)) {
781         int svmand;
782         nbl_op_t nop;
783
784         nbl_start_crit(vp, RW_READER);
785         in_crit = 1;
786         error = nbl_svmand(vp, fp->f_cred, &svmand);
787         if (error != 0)
788             goto done;
789         if ((prot & PROT_WRITE) && (type == MAP_SHARED)) {
790             if (prot & (PROT_READ | PROT_EXEC)) {
791                 nop = NBL_READWRITE;
792             } else {
793                 nop = NBL_WRITE;
794             }
795         } else {
796             nop = NBL_READ;
797         }
798         if (nbl_conflict(vp, nop, 0, LONG_MAX, svmand, NULL)) {
799             error = EACCES;
800             goto done;
801         }
802     }
803
804     /* discard lwpchan mappings, like munmap() */
805     if ((flags & MAP_FIXED) && curproc->p_lcp != NULL)
806         lwpchan_delete_mapping(curproc, *addrp, *addrp + len);
807
808     /*
809      * Ok, now let the vnode map routine do its thing to set things up.
810      */
811     error = VOP_MAP(vp, pos, as,
812         addrp, len, uprot, maxprot, flags, fp->f_cred, NULL);
813
814     if (error == 0) {
815         /*
816          * Tell machine specific code that lwp has mapped shared memory
817          */
818         if (flags & MAP_SHARED) {

```

```

819         /* EMPTY */
820         LWP_MMODEL_SHARED_AS(*addrp, len);
821     }
822     if (vp->v_type == VREG &&
823         (flags & (MAP_TEXT | MAP_INITDATA)) != 0) {
824         /*
825          * Mark this as an executable vnode
826          */
827         mutex_enter(&vp->v_lock);
828         vp->v_flag |= VVMEXEC;
829         mutex_exit(&vp->v_lock);
830     }
831 }
832
833 done:
834     if (in_crit)
835         nbl_end_crit(vp);
836     return (error);
837 }

```

unchanged portion omitted