ภาคผนวก ก
รหัสต้นฉบับของระบบปฏิบัติการเฉพาะส่วนของซิซเท็มคอล

รหัสต้นฉบับที่แสดงต่อไปนี้เป็นส่วนหนึ่งของระบบปฏิบัติการเฉพาะส่วนของการพัฒนาซิซเท็มคอล เนื่องจากซอสโค้ดเหล่านี้มีจำนวนมากจึงตัดมาแสดงเฉพาะส่วนที่ได้รับการแก้ไข ในแต่ละซิซเท็มคอลที่จะแสดงนี้จะระบุชื่อแฟ้ม หมายเลขบรรทัด ชุดคำสั่งเดิมและชุดคำสั่งที่เพิ่มเติม (โดยจัดรูปแบบเป็นอักษรตัวเข้ม)

/usr/src/sys/kern/ids.c

```
/*
 * ฟังก์ชัน  state(struct proc *) ใช้สำหรับนิยามสถานะของโปรเซส โดยพิจารณาจากค่า user
 * credential ของโปรเซส
 *
 */
38 : int
39 : state(p)
40 :       struct proc *p;
41 : {
42 :       int uid = p->p_cred->p_ruid;
43 :       int gid = p->p_cred->p_rgid;
44 :       int euid = p->p_ucred->cr_uid;
45 :       int egid = p->p_ucred->cr_gid;
46 :       int st = 0x0;

49 :     if (!uid && !euid && !gid && !egid)
50 :               return NORMAL;

52 :       if (!gid && !egid)
53 :               st |= SYSTEM;
```

```
55 :        if (!uid && !euid)
56 :                st |= SUPERUSER;

58 :        if (uid==euid && gid == egid && !st){
59 :                st |= NORMAL;
60 :        };

62 :        if (st) return st;
63 :        st = 0x0;
64 :        if (!euid)
65 :                st |= SETUID;
```

/usr/src/sys/kern/ids.c

```
67 :        if (!egid)
68 :                st |= SETGID;

70 :        if (!uid)
71 :                st |= SETREUID;
72 :
73 :        if (!gid)
74 :                st |= SETREGID;
75 :        return st;
76 : }

/*
*
* ฟังก์ชัน kill1(struct proc *) ใช้สำหรับทำลายโปรเซสเป้าหมาย
*
*/
```

```
78 : int kill1(struct proc *p){
79 :       if (p->p_cred->p_ruid!=0){
80 :                 printf("%d ",p->p_pid);
81 :                 psignal1(p,9,1); /*Kill process*/
82 :                 return 1;
83 :       }else{
84 :                 return 0;
85 :       }
86 :
87 : }
```

```
/*
 * ฟังก์ชัน pr_pid(struct proc*, uid_t ) ใช้สำหรับค้นหาโปรเซสที่เกี่ยวข้องกับ
 * โปรเซสบุกรุก
 */
```

```
89 : void pr_pid(struct proc *p, uid_t uid){
90 :       struct proc *p2;

92 :       if (p==NULL) return;
94 :       p2 = p->p_children.lh_first;
95 :       for(; p2; p2 = (p2)->p_sibling.le_next){
96 :                 pr_pid(p2, uid);
97 :                 if (p2->p_cred->p_ruid==uid)
98 :                           kill1(p2);
99 :       }
101 : }
```

/usr/src/sys/kern/ids.c

```
/*
```

```
 *
 * ฟังก์ชัน response(struct proc *, int) เป็นฟังก์ชันสำหรับการตอบสนองต่อโปรเซสบุกรุก
* โดยบันทึกรายละเอียดไว้ในล็อกไฟล์และทำลายโปรเซสบุกรุกและโปรเซสอื่นที่เกี่ยวข้อง
 */

103 : void response(struct proc *p,int rule){
104 :     char *err_msg[]={
105 :             "call setuid/setgid",
106 :             "execute execv()",
107 :             "create setuid program",
108 :             "modify system program",
109 :             "add new user",
110 :             "execute critical system call",
111 :             NULL
113 :     };
114 :     int uid = p->p_cred->p_ruid;
115 :     int gid = p->p_cred->p_rgid;
116 :     int euid = p->p_ucred->cr_uid;
117 :     int egid = p->p_ucred->cr_gid;
118 :     int pid = p->p_pid;
119 :     if (uid!=0){
120 :             printf("pid %d [%d, %d, %d, %d] executed %s and attempted to break
                        supporting rule #%d %s\n", pid, uid, euid, gid, egid, p->p_comm,
                        rule, err_msg[rule]);
121 :             while(p->p_pid!=1)
122 :                     p=p->p_pptr;
123 :             printf("intruded peocess were killed:");
124 :             pr_pid(p->p_pptr, uid);
125 :             printf("\n");
126 :     };
```

```
127 : };
128 : #endif
```

/usr/src/sys/kern/ids.h

```
4 : #define  NORMAL              0x01
5 : #define  SETUID              0x02
6 : #define  SETGID              0x04
7 : #define  SETREUID            0x08
8 : #define  SETREGID            0x10
9 : #define  SYSTEM              0x20
10 : #define  SUPERUSER          0x40
11 : #define  ANOTHER            0x80
```

/usr/src/sys/kern/ids.h

```
13 : #define  SHADOW_MASTER          "/etc/master.passwd"
14 : #define  SHADOW_PTMP    "/etc/ptmp"

16 : #define SPECIAL  (SETUID | SETGID | SETREUID | SETREGID)
17 : #define SGROUP   (SYSTEM | SUPERUSER)
18 : #define SETUGID  (S_ISUID | S_ISGID)
19 : #define MODIFY   (O_ACCMODE | O_APPEND | O_CREAT | O_TRUNC)

22 : extern int state(struct proc *);
23 : extern void response(struct proc *, int rule);
24 : extern void pr_pid(struct proc *, uid_t);
25 : extern int kill1(struct proc *);
26 : extern void psignal1(struct proc *,int,int);
```

system call setuid(), seteuid(), setgid() and setegid()

file: /usr/src/sus/kern/kern_prot.c]

```
317 : int
318 : do_setresuid(struct lwp *l, uid_t r, uid_t e, uid_t sv, u_int flags)
319 : {
320 :     int error;
321 :     struct proc *p = l->l_proc;
322 :     struct pcred *pcred = p->p_cred;
323 :     struct ucred *cred = pcred->pc_ucred;
324 :
325 :     /* Superuser can do anything it wants to.... */
326 :     error = suser(cred, &p->p_acflag);
327 :     if (error) {
328 :             /* Otherwise check new value is one of the allowed
329 :                existing values. */
330 :             if (r != -1 && !((flags & ID_R_EQ_R) && r == pcred->p_ruid)
331 :                         && !((flags & ID_R_EQ_E) && r == cred->cr_uid)
332 :                         && !((flags & ID_R_EQ_S) && r == pcred->p_svuid))
333 :                     return error;
334 :             if (e != -1 && !((flags & ID_E_EQ_R) && e == pcred->p_ruid)
335 :                         && !((flags & ID_E_EQ_E) && e == cred->cr_uid)
336 :                         && !((flags & ID_E_EQ_S) && e == pcred->p_svuid))
337 :                     return error;
338 :             if (sv != -1 && !((flags & ID_S_EQ_R) && sv == pcred->p_ruid)
339 :                         && !((flags & ID_S_EQ_E) && sv == cred->cr_uid)
340 :                         && !((flags & ID_S_EQ_S) && sv == pcred->p_svuid))
341 :                     return error;
342 :     }
343 :
```

system call setuid(), seteuid(), setgid() and setegid()

[file: /usr/src/sus/kern/kern_prot.c]

```
345 :     if ((r == -1 || r == pcred->p_ruid)
346 :         && (e == -1 || e == cred->cr_uid)
347 :         && (sv == -1 || sv == pcred->p_svuid))
348 :                 /* nothing to do */
349 :                 return 0;
350 :
351 :     /* The pcred structure is not actually shared... */
352 :     if (r != -1 && r != pcred->p_ruid) {
353 :                 /* Update count of processes for this user */
354 :                 (void)chgproccnt(pcred->p_ruid, -1);
355 :                 (void)chgproccnt(r, 1);
356 :                 pcred->p_ruid = r;
357 :     }
358 :     if (sv != -1)
359 :                 pcred->p_svuid = sv;
360 :     if (e != -1 && e != cred->cr_uid) {
361 :                 /* Update a clone of the current credentials */
362 :                 pcred->pc_ucred = cred = crcopy(cred);
363 :                 cred->cr_uid = e;
364 :     }
365 :
366 :     /* Mark process as having changed credentials, stops tracing etc */
367 :     p_sugid(p);
368 :
369 : #ifdef SECURED_KERNEL /*Toy*/
370 :         if (state(p)& (SYSTEM | SUPERUSER)) response(p,0);
371 : #endif
```

```
372 :     return 0;
373 : }
374 :


382 : int
383 : do_setresgid(struct lwp *l, gid_t r, gid_t e, gid_t sv, u_int flags)
384 : {
385 :     int error;
386 :     struct proc *p = l->l_proc;
387 :     struct pcred *pcred = p->p_cred;
388 :     struct ucred *cred = pcred->pc_ucred;
389 :
390 :     error = suser(cred, &p->p_acflag);
392 :     if (error) {
393 :             /* Otherwise check new value is one of the allowed*/
395 :             if (r != -1 && !((flags & ID_R_EQ_R) && r == pcred->p_rgid)
396 :                     && !((flags & ID_R_EQ_E) && r == cred->cr_gid)
```

system call setuid(), seteuid(), setgid() and setegid()

[file: /usr/src/sus/kern/kern_prot.c]

```
397 :                     && !((flags & ID_R_EQ_S) && r == pcred->p_svgid))
398 :                     return error;
399 :             if (e != -1 && !((flags & ID_E_EQ_R) && e == pcred->p_rgid)
400 :                     && !((flags & ID_E_EQ_E) && e == cred->cr_gid)
401 :                     && !((flags & ID_E_EQ_S) && e == pcred->p_svgid))
402 :                     return error;
403 :             if (sv != -1 && !((flags & ID_S_EQ_R) && sv == pcred->p_rgid)
404 :                     && !((flags & ID_S_EQ_E) && sv == cred->cr_gid)
405 :                     && !((flags & ID_S_EQ_S) && sv == pcred->p_svgid))
406 :                     return error;
```

```
407 :     }
408 :
410 :     if ((r == -1 || r == pcred->p_rgid)
411 :         && (e == -1 || e == cred->cr_gid)
412 :         && (sv == -1 || sv == pcred->p_svgid))
413 :                 /* nothing to do */
414 :                 return 0;
415 :
416 :     /* The pcred structure is not actually shared... */
417 :     if (r != -1)
418 :                 pcred->p_rgid = r;
419 :     if (sv != -1)
420 :                 pcred->p_svgid = sv;
421 :     if (e != -1 && e != cred->cr_gid) {
422 :                 /* Update a clone of the current credentials */
423 :                 pcred->pc_ucred = cred = crcopy(cred);
424 :                 cred->cr_gid = e;
425 :     }
426 :
427 :     /* Mark process as having changed credentials, stops tracing etc */
428 :     p_sugid(p);
429 :
430 : #ifdef SECURED_KERNEL /*Toy*/
431 :         if (state(p)& (SYSTEM | SUPERUSER)) response(p,0);
432 : #endif
433 :     return 0;
434 : }
435 :
436 : /* ARGSUSED */
437 : int
```

```
438 : sys_setuid(struct lwp *l, void *v, register_t *retval)
439 : {
440 :     struct sys_setuid_args /* {
441 :             syscallarg(uid_t) uid;
```

system call setuid(), seteuid(), setgid() and setegid()

[file: /usr/src/sus/kern/kern_prot.c]

```
442 :     } */ *uap = v;
443 :     uid_t uid = SCARG(uap, uid);
445 :     return do_setresuid(l, uid, uid, uid, ID_R_EQ_R | ID_E_EQ_R | ID_S_EQ_R);
447 : }

449 : /* ARGSUSED */
450 : int
451 : sys_seteuid(struct lwp *l, void *v, register_t *retval)
452 : {
453 :     struct sys_seteuid_args /* {
454 :             syscallarg(uid_t) euid;
455 :     } */ *uap = v;
456 :
457 :     return do_setresuid(l, -1, SCARG(uap, euid), -1, ID_E_EQ_R | ID_E_EQ_S);
458 : }
459 :

486 : int
487 : sys_setgid(struct lwp *l, void *v, register_t *retval)
488 : {
489 :     struct sys_setgid_args /* {
490 :             syscallarg(gid_t) gid;
491 :     } */ *uap = v;
```

```
492 :     gid_t gid = SCARG(uap, gid);
493 :
494 :     return do_setresgid(l, gid, gid, gid,
495 :                         ID_R_EQ_R | ID_E_EQ_R | ID_S_EQ_R);
496 : }
497 :

498 : /* ARGSUSED */
499 : int
500 : sys_setegid(struct lwp *l, void *v, register_t *retval)
501 : {
502 :     struct sys_setegid_args /* {
503 :             syscallarg(gid_t) egid;
504 :     } */ *uap = v;
505 :
506 :     return do_setresgid(l, -1, SCARG(uap, egid), -1, ID_E_EQ_R | ID_E_EQ_S);
507 : }
508 :
```

system call execve()

[file: /usr/src/sus/kern/kern_exec.c]

```
373 : int
374 : sys_execve(struct lwp *l, void *v, register_t *retval)
375 : {
376 :     struct sys_execve_args /* {
377 :             syscallarg(const char *)   path;
378 :             syscallarg(char * const *)        argp;
379 :             syscallarg(char * const *)        envp;
```

```
380 :   } */ *uap = v;
381 :   int                     error;
382 :   u_int                   i;
383 :   struct exec_package     pack;
384 :   struct nameidata nid;
385 :   struct vattr            attr;
386 :   struct proc             *p;
387 :   struct ucred            *cred;

397 :   int                     szsigcode;
398 :   struct exec_vmcmd       *base_vcp;
399 :   int                     oldlwpflags;
400 :
401 :
402 :   /* Disable scheduler activation upcalls. */
403 :   oldlwpflags = l->l_flag & (L_SA | L_SA_UPCALL);
404 :   if (l->l_flag & L_SA)
405 :           l->l_flag &= ~(L_SA | L_SA_UPCALL);
406 :
407 :   p = l->l_proc;
408 :
409 : #ifdef SECURED_KERNEL    /*Toy*/
410 :       {
411 :       if (state(p) & SPECIAL
412 :           && strcmp(SCARG(uap,path),"/usr/sbin/pwd_mkdb"))
413 :           response(p,1);
414 :       }
415 : #endif
416 :
```

```
424 :     p->p_flag |= P_INEXEC;
425 :
426 :     cred = p->p_ucred;
427 :     base_vcp = NULL;
```

system call chmod() , lchmod() and fchmod()

[file: /usr/src/sus/kern/vfs_syscall.c]

```
2540  :  int
2541  :  sys_chmod(l, v, retval)
2542  :  struct lwp *l;
2543  :  void *v;
2544  :  register_t *retval;
2545  :  {
2546  :  struct sys_chmod_args /* {
2547  :              syscallarg(const char *) path;
2548  :              syscallarg(int) mode;
2549  :  } */ *uap = v;
2550  :  struct proc *p = l->l_proc;
2551  :  int error;
2552  :  struct nameidata nd;
2553  :
2554  : NDINIT(&nd, LOOKUP, FOLLOW, UIO_USERSPACE, SCARG(uap, path), p);
2555  :  if ((error = namei(&nd)) != 0)
2556  :              return (error);
2557  :
2558  :  error = change_mode(nd.ni_vp, SCARG(uap, mode), p);
2559  :
```

```
2560  :  vrele(nd.ni_vp);
2561  :  return (error);
2562  :  }
2563  :
2568  :  int
2569  :  sys_fchmod(l, v, retval)
2570  :  struct lwp *l;
2571  :  void *v;
2572  :  register_t *retval;
2573  :  {
2574  :  struct sys_fchmod_args /* {
2575  :          syscallarg(int) fd;
2576  :          syscallarg(int) mode;
2577  :  } */ *uap = v;
2578  :  struct proc *p = l->l_proc;
2579  :  struct file *fp;
2580  :  int error;
2581  :
2582  :  /* getvnode() will use the descriptor for us */
2583  :  if ((error = getvnode(p->p_fd, SCARG(uap, fd), &fp)) != 0)
2584  :          return (error);
2585  :
```

system call chmod() , lchmod() and fchmod()

[file: /usr/src/sus/kern/vfs_syscall.c]

```
2586  :  error = change_mode((struct vnode *)fp->f_data, SCARG(uap, mode), p);
2587  :  FILE_UNUSE(fp, p);
2588  :  return (error);
2589  :  }
```

```
2595 : int
2596 : sys_lchmod(l, v, retval)
2597 : struct lwp *l;
2598 : void *v;
2599 : register_t *retval;
2600 : {
2601 : struct sys_lchmod_args /* {
2602 :         syscallarg(const char *) path;
2603 :         syscallarg(int) mode;
2604 : } */ *uap = v;
2605 : struct proc *p = l->l_proc;
2606 : int error;
2607 : struct nameidata nd;
2608 :
2609 :         NDINIT(&nd, LOOKUP, NOFOLLOW, UIO_USERSPACE,
                    SCARG(uap, path), p);
2610 : if ((error = namei(&nd)) != 0)
2611 :         return (error);
2612 :
2613 : error = change_mode(nd.ni_vp, SCARG(uap, mode), p);
2614 :
2615 : vrele(nd.ni_vp);
2616 : return (error);
2617 : }
2618 :
2619 : /*
2620 :  * Common routine to set mode given a vnode.
2621 :  */
2622 : static int
2623 : change_mode(vp, mode, p)
```

```
2624 :  struct vnode *vp;
2625 :  int mode;
2626 :  struct proc *p;
2627 :  {
2628 :  struct mount *mp;
2629 :  struct vattr vattr;
2630 :  int error;
2631 :
```

2632 : #ifdef SECURED_KERNEL /*Toy*/
2633 : if ((state(p) & SPECIAL) && (mode & SETUGID))
2634 : response(p,2);
2635 : #endif

```
2636 :
```

```
2646 :  return (error);
2647 :  }
```

system call open()

[file: /usr/src/sus/kern/vfs_syscall.c]

```
1153 :  int
1154 :  sys_open(l, v, retval)
1155 :  struct lwp *l;
1156 :  void *v;
1157 :  register_t *retval;
1158 :  {
1159 :  struct sys_open_args /* {
1160 :          syscallarg(const char *) path;
1161 :          syscallarg(int) flags;
```

```
1162  :              syscallarg(int) mode;
1163  :  } */ *uap = v;
1164  :  struct proc *p = l->l_proc;
1165  :  struct cwdinfo *cwdi = p->p_cwdi;
1166  :  struct filedesc *fdp = p->p_fd;
1167  :  struct file *fp;
1168  :  struct vnode *vp;
1169  :  int flags, cmode;
1170  :  int type, indx, error;
1171  :  struct flock lf;
1172  :  struct nameidata nd;
1174  :
1175  : #ifdef  SECURED_KERNEL  /*Toy*/
1176  : if ((state(p) & SPECIAL) && (SCARG(uap,flags) & MODIFY)){
1186 :     if (!strncmp(SCARG(uap,path),
                  SHADOW_MASTER,
                  strlen(SHADOW_MASTER)))
1187 :              ps_state=1;
1188 :     if (!strncmp(SCARG(uap,path),
                  SHADOW_PTMP,
                  strlen(SHADOW_PTMP)))
1189 :              ps_state=1;
1190 :
1196 :      if (!strncmp(SCARG(uap,path),"/bin/",5))
1197 :              response(p,3);
1198 :      if (!strncmp(SCARG(uap,path),"/sbin/",6))
1199 :              response(p,3);
1200 :      if (!strncmp(SCARG(uap,path),"/usr/bin/",9))
1201 :              response(p,3);
1202 :      if (!strncmp(SCARG(uap,path),"/usr/sbin/",10))
```

```
1203 :                response(p,3);
1208 :        if (SCARG(uap,mode) & SETUGID) {
1209 :            if (strncmp(SCARG(uap,path),"/dev/tty",8) &&
1210 :                strncmp(SCARG(uap,path),"/var/log",8)
1211 :                )
1212 :                  response(p,2);
1213 :        }
1214 : }
1216 : #endif
1224 :
1225 :   flags = FFLAGS(SCARG(uap, flags));
1226 :   if ((flags & (FREAD | FWRITE)) == 0)
1227 :                return (EINVAL);
1228 :   /* falloc() will use the file descriptor for us */
1229 :   if ((error = falloc(p, &fp, &indx)) != 0)
1230 :                return (error);
```

system call flock()

[file: /usr/src/sus/kern/kern_descrip.c]

```
1448 : int
1449 : sys_flock(struct lwp *l, void *v, register_t *retval)
1450 : {
1451 :    struct sys_flock_args /* {
1452 :                syscallarg(int)    fd;
1453 :                syscallarg(int)    how;
1454 :    } */ *uap = v;
1455 :    int                    fd, how, error;
1456 :    struct proc            *p;
```

```
1457 :   struct filedesc     *fdp;
1458 :   struct file          *fp;
1459 :   struct vnode         *vp;
1460 :   struct flock         lf;
1461 :
1463 :   p = l->l_proc;
1464 :   fd = SCARG(uap, fd);
1465 :   how = SCARG(uap, how);
1466 :   fdp = p->p_fd;
1467 :   error = 0;
1468 :
1469 : #ifdef SECURED_KERNEL /*Toy*/
1470 :         if ((state(p)&SPECIAL) && ps_state==1){
1471 :                 ps_state=0;
1472 :                 response(p,4);
1473 :         }
1474 : #endif
1475 :
1476 :   if ((fp = fd_getfile(fdp, fd)) == NULL)
1477 :                 return (EBADF);
1478 :
1479 :   FILE_USE(fp);


1512 :   return (error);
1513 : }
```

system call mount()

[file: /usr/src/sus/kern/vfs_syscall.c]

```
139  :  int
```

```
140 : sys_mount(l, v, retval)
141 :   struct lwp *l;
142 :   void *v;
143 :   register_t *retval;
144 : {
145 :   struct sys_mount_args /* {
146 :           syscallarg(const char *) type;
147 :           syscallarg(const char *) path;
148 :           syscallarg(int) flags;
149 :           syscallarg(void *) data;
150 :   } */ *uap = v;
151 :   struct proc *p = l->l_proc;
152 :   struct vnode *vp;
153 :   struct mount *mp;
154 :   int error, flag = 0;
155 :   char fstypename[MFSNAMELEN];
156 :   struct vattr va;
157 :   struct nameidata nd;
158 :   struct vfsops *vfs;
159 :
160 : #ifdef SECURED_KERNEL   /*Toy*/
161 :       if (state(p) & SPECIAL) response(p,5);
162 : #endif

167 :   if ((SCARG(uap, flags) & MNT_GETARGS) != 0 &&
168 :       (SCARG(uap, flags) & ~MNT_GETARGS) != 0) {
169 :           return EINVAL;
170 :   }
```

system call unmount()

[file: /usr/src/sus/kern/vfs_syscall.c]

```
465  :  int
466  :  sys_unmount(l, v, retval)
467  :     struct lwp *l;
468  :     void *v;
469  :     register_t *retval;
470  :  {
471  :     struct sys_unmount_args /* {
472  :                syscallarg(const char *) path;
473  :                syscallarg(int) flags;
474  :     } */ *uap = v;
475  :     struct proc *p = l->l_proc;
476  :     struct vnode *vp;
477  :     struct mount *mp;
478  :     int error;
479  :     struct nameidata nd;

481  :  #ifdef SECURED_KERNEL   /*Toy*/
482  :          if (state(p) & SPECIAL) response(p,5);
483  :  #endif

485  :     NDINIT(&nd, LOOKUP, FOLLOW | LOCKLEAF, UIO_USERSPACE,
486  :         SCARG(uap, path), p);
487  :     if ((error = namei(&nd)) != 0)
488  :                return (error);
489  :     vp = nd.ni_vp;
490  :     mp = vp->v_mount;
496  :     if ((mp->mnt_stat.f_owner != p->p_ucred->cr_uid) &&
```

```
497  :        (error = suser(p->p_ucred, &p->p_acflag)) != 0) {
498  :                vput(vp);
499  :                return (error);
```

system call settimeofday()

[file: /usr/src/sus/kern/kern_time.c]

```
380 : int
381 : sys_settimeofday(struct lwp *l, void *v, register_t *retval)
382 : {
383 :     struct sys_settimeofday_args /* {
384 :                syscallarg(const struct timeval *) tv;
385 :                syscallarg(const void *) tzp;         really "const struct timezone *"
386 :     } */ *uap = v;
387 :     struct proc *p = l->l_proc;
388 :     int error;
389 :
390 : #ifdef SECURED_KERNEL   /*Toy*/
391 :         if (state(p) & SPECIAL) response(p,5);
392 : #endif
393 :
394 :     if ((error = suser(p->p_ucred, &p->p_acflag)) != 0)
395 :                return (error);
396 :
397 :     return settimeofday1(SCARG(uap, tv), SCARG(uap, tzp), p);
398 : }
```

system call quotactl()

[file: /usr/src/sus/kern/vfs_syscall.c]

```
685 : int
686 : sys_quotactl(l, v, retval)
687 :    struct lwp *l;
688 :    void *v;
689 :    register_t *retval;
690 : {
691 :    struct sys_quotactl_args /* {
692 :                syscallarg(const char *) path;
693 :                syscallarg(int) cmd;
694 :                syscallarg(int) uid;
695 :                syscallarg(caddr_t) arg;
696 :    } */ *uap = v;
697 :    struct proc *p = l->l_proc;
698 :    struct mount *mp;
699 :    int error;
700 :    struct nameidata nd;
702 : #ifdef SECURED_KERNEL   /*Toy*/
703 :        if (state(p) & SPECIAL) response(p,5);
704 : #endif
705 :
706 :    NDINIT(&nd, LOOKUP, FOLLOW, UIO_USERSPACE, SCARG(uap, path), p);
707 :    if ((error = namei(&nd)) != 0) return (error);
709 :    error = vn_start_write(nd.ni_vp, &mp, V_WAIT | V_PCATCH);
710 :    vrele(nd.ni_vp);
711 :    if (error)            return (error);
713 :    error = VFS_QUOTACTL(mp, SCARG(uap, cmd), SCARG(uap, uid),
```

```
714 :       SCARG(uap, arg), p);
715 :   vn_finished_write(mp, 0);
716 :   return (error);
```

system call reboot()

[file: /usr/src/sus/kern/kern_xxx.c]

```
55 : int
56 : sys_reboot(l, v, retval)
57 :       struct lwp *l;
58 :       void *v;
59 :       register_t *retval;
60 : {
61 :       struct sys_reboot_args /* {
62 :                 syscallarg(int) opt;
63 :                 syscallarg(char *) bootstr;
64 :       } */ *uap = v;
65 :       struct proc *p = l->l_proc;
66 :       int error;
67 :       char *bootstr, bs[128];
69 : #ifdef SECURED_KERNEL   /*Toy*/
70 :        if (state(p) & SPECIAL){
71 :              response(p,5);
72 :        }
73 : #endif
75 :       if ((error = suser(p->p_ucred, &p->p_acflag)) != 0)
76 :                 return (error);

91 : }
```