Subject: util-linux-ng: unprivileged mounts support
Posted by Miklos Szeredi on Tue, 08 Jan 2008 11:50:46 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

This is an experimental patch for supporing unprivileged mounts and
umounts. The following features are added:

1) If mount/umount are suid, first try without privileges.

This is done by forking, dropping privileges in child, and redirecting
stderr to /dev/null. If this succeeds, then parent exits with zero
exit code. Otherwise parent continues normally (with privileges).
This isn't perfect, because the wrong error message will be printed if
mount/umount failed not because of insufficient privileges, but some
other error (e.g. mountpoint busy).

2) Support user mounts in kernel.

If /etc/mtab is a symlink (to /proc/mounts if it's been set up
correctly), then change fsuid for the duration of the mount syscall
and use the MS_SETOWNER flag. Old kernels will simply ignore this,
and everything will work as before. Kernels with the unprivileged
mounts patches will set the owner of the mount, and the relevant line
in /proc/PID/mounts will contain a "user=XYZ" option, making this
backward compatible with /etc/mtab.

3) Root can force a specific user for a mount with "-ouser=XYZ". This
has worked previously as well, but that was probably just accidental
(the option was copied verbatim to /etc/mtab).

4) Add support for "mnt" and "nomnt" options. These can be used to
allow or prohibit unprivileged submounting of a user mount.

Example:

root# mount --bind -ouser=xyz /home/xyz /home/xyz
xyz> mount --bind ~/foo ~/bar
xyz> umount ~/bar

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: util-linux-ng/configure.ac
===
--- util-linux-ng.orig/configure.ac 2008-01-07 21:40:22.000000000 +0100
+++ util-linux-ng/configure.ac 2008-01-07 21:40:50.000000000 +0100

```
@@ -91,6 +92,11 @@ fi
 UTIL_CHECK_LIB(util, openpty)
 UTIL_CHECK_LIB(termcap, tgetnum)

+UTIL_CHECK_LIB(cap, cap_get_proc)
+if test $have_cap = no; then
+   AC_MSG_ERROR([libcap is required for mount]);
+fi
+
 AC_ARG_WITH([fsprobe],
   [AS_HELP_STRING([--with-fsprobe], [library to guess filesystems (blkid|volume_id), default is
blkid])],
   [], [with_fsprobe=blkid]
Index: util-linux-ng/mount/Makefile.am
===================================================================
--- util-linux-ng.orig/mount/Makefile.am 2008-01-07 21:40:22.000000000 +0100
+++ util-linux-ng/mount/Makefile.am 2008-01-07 21:40:50.000000000 +0100
@@ -45,6 +45,10 @@ if HAVE_SELINUX
 mount_LDADD += -lselinux
 endif

+if HAVE_CAP
+mount_LDADD += -lcap
+endif
+
 if HAVE_VOLUME_ID
 utils_common += fsprobe_volumeid.c
 swapon_SOURCES += ../lib/linux_version.c ../lib/blkdev.c
Index: util-linux-ng/mount/fsprobe.h
===================================================================
--- util-linux-ng.orig/mount/fsprobe.h 2008-01-07 21:40:22.000000000 +0100
+++ util-linux-ng/mount/fsprobe.h 2008-01-07 21:40:50.000000000 +0100
@@ -33,6 +33,7 @@ struct mountargs {
  const char *type;
  int flags;
  void *data;
+ uid_t uid;
 };

 extern int fsprobe_known_fstype_in_procfs(const char *type);
Index: util-linux-ng/mount/fstab.c
===================================================================
--- util-linux-ng.orig/mount/fstab.c 2008-01-07 21:40:22.000000000 +0100
+++ util-linux-ng/mount/fstab.c 2008-01-07 21:40:50.000000000 +0100
@@ -47,7 +47,7 @@ mtab_does_not_exist(void) {
  return var_mtab_does_not_exist;
 }
```

```
-static int
+int
 mtab_is_a_symlink(void) {
  get_mtab_info();
  return var_mtab_is_a_symlink;
Index: util-linux-ng/mount/fstab.h
===================================================================
--- util-linux-ng.orig/mount/fstab.h 2008-01-07 21:40:22.000000000 +0100
+++ util-linux-ng/mount/fstab.h 2008-01-07 21:40:50.000000000 +0100
@@ -2,6 +2,7 @@
 #define MOUNT_FSTAB_H

 #include "mount_mntent.h"
+int mtab_is_a_symlink(void);
 int mtab_is_writable(void);
 int mtab_does_not_exist(void);
 int is_mounted_once(const char *name);
Index: util-linux-ng/mount/mount.c
===================================================================
--- util-linux-ng.orig/mount/mount.c 2008-01-07 21:40:22.000000000 +0100
+++ util-linux-ng/mount/mount.c 2008-01-07 21:40:50.000000000 +0100
@@ -20,6 +20,8 @@
 #include <sys/stat.h>
 #include <sys/wait.h>
 #include <sys/mount.h>
+#include <sys/fsuid.h>
+#include <sys/capability.h>

 #include <mntent.h>

@@ -102,7 +104,7 @@ struct opt_map {
 #define MS_USER  0x20000000
 #define MS_OWNER 0x10000000
 #define MS_GROUP 0x08000000
-#define MS_COMMENT 0x02000000
+#define MS_COMMENT 0x04000000
 #define MS_LOOP  0x00010000

 /* Options that we keep the mount system call from seeing.  */
@@ -177,6 +179,8 @@ static const struct opt_map opt_map[] =
        to mtime/ctime */
 #endif
   { "nofail", 0, 0, MS_COMMENT}, /* Do not fail if ENOENT on dev */
+  { "mnt", 0, 1, MS_NOMNT}, /* permit unprivileged submounts */
+  { "nomnt", 0, 0, MS_NOMNT}, /* no unprivileged submounts */
   { NULL, 0, 0, 0  }
 };
```

```
@@ -365,7 +369,7 @@ append_context(const char *optname, char
  * For the options uid= and gid= replace user or group name by its value.
  */
 static inline void
-parse_opt(char *opt, int *mask, char **extra_opts) {
+parse_opt(char *opt, int *mask, char **extra_opts, char **user) {
  const struct opt_map *om;

  for (om = opt_map; om->opt != NULL; om++)
@@ -410,6 +414,11 @@ parse_opt(char *opt, int *mask, char **e
    return;
   }
  }
+ if (strncmp(opt, "user=", 5) == 0) {
+  free(*user);
+  *user = xstrdup(opt + 5);
+  return;
+ }

 #ifdef HAVE_LIBSELINUX
  if (strncmp(opt, "context=", 8) == 0 && *(opt+8)) {
@@ -431,7 +440,7 @@ parse_opt(char *opt, int *mask, char **e
 /* Take -o options list and compute 4th and 5th args to mount(2).  flags
    gets the standard options (indicated by bits) and extra_opts all the rest */
 static void
-parse_opts (const char *options, int *flags, char **extra_opts) {
+parse_opts (const char *options, int *flags, char **extra_opts, char **user) {
  *flags = 0;
  *extra_opts = NULL;

@@ -454,7 +463,7 @@ parse_opts (const char *options, int *fl
    /* end of option item or last item */
    if (*p == '\0' || *(p+1) == '\0') {
     if (!parse_string_opt(opt))
-     parse_opt(opt, flags, extra_opts);
+     parse_opt(opt, flags, extra_opts, user);
     opt = NULL;
    }
   }
@@ -525,6 +534,7 @@ create_mtab (void) {
  struct my_mntent mnt;
  int flags;
  mntFILE *mfp;
+ char *user = NULL;

  lock_mtab();

@@ -538,11 +548,11 @@ create_mtab (void) {
```

```
  /* Find the root entry by looking it up in fstab */
  if ((fstab = getfs_by_dir ("/")) || (fstab = getfs_by_dir ("root"))) {
    char *extra_opts;
-   parse_opts (fstab->m.mnt_opts, &flags, &extra_opts);
+   parse_opts (fstab->m.mnt_opts, &flags, &extra_opts, &user);
    mnt.mnt_dir = "/";
    mnt.mnt_fsname = fsprobe_get_devname(fstab->m.mnt_fsname);
    mnt.mnt_type = fstab->m.mnt_type;
-   mnt.mnt_opts = fix_opts_string (flags, extra_opts, NULL);
+   mnt.mnt_opts = fix_opts_string (flags, extra_opts, user);
    mnt.mnt_freq = mnt.mnt_passno = 0;
    my_free(extra_opts);

@@ -564,6 +574,39 @@ create_mtab (void) {
   unlock_mtab();
 }

+static int set_fsuid(uid_t uid, uid_t *olduid)
+{
+ cap_t cap;
+ int res;
+
+ cap = cap_get_proc();
+ if (!cap) {
+   die(EX_FAIL, _("mount: failed to get capabilities: %s"),
+       strerror(errno));
+ }
+
+ res = setfsuid(uid);
+ if (olduid)
+   *olduid = res;
+
+ if (setfsuid(uid) != uid) {
+   error(_("mount: failed to set user"));
+   errno = EPERM;
+   return -1;
+ }
+
+ res = cap_set_proc(cap);
+ if (res == -1) {
+   die(EX_FAIL, _("mount: failed to restore capabilities"),
+       strerror(errno));
+ }
+
+ if (verbose > 2)
+   printf(_("mount: fsuid set to %i\n"), uid);
+
+ return 0;
```

```
+}
+
 /* count successful mount system calls */
 static int mountcount = 0;

@@ -575,16 +618,33 @@ static int mountcount = 0;
 static int
 do_mount_syscall (struct mountargs *args) {
  int flags = args->flags;
+ uid_t olduid = 0;
+ int res;

  if ((flags & MS_MGC_MSK) == 0)
   flags |= MS_MGC_VAL;

+ if (args->flags & MS_SETUSER) {
+  if (set_fsuid(args->uid, &olduid) == -1)
+   return -1;
+ }
+
  if (verbose > 2)
   printf("mount: mount(2) syscall: source: \"%s\", target: \"%s\", "
    "filesystemtype: \"%s\", mountflags: %d, data: %s\n",
    args->spec, args->node, args->type, flags, (char *) args->data);
+ res = mount(args->spec, args->node, args->type, flags, args->data);

- return mount (args->spec, args->node, args->type, flags, args->data);
+ if (args->flags & MS_SETUSER) {
+  int errno_save = errno;
+
+  if (set_fsuid(olduid, NULL) == -1)
+   die(EX_FAIL, _("mount: failed to restore fsuid"));
+
+  errno = errno_save;
+ }
+
+ return res;
 }

 /*
@@ -706,6 +766,31 @@ guess_fstype_by_devname(const char *devn
    return type;
 }

+static int get_user(const char *user, uid_t *uid)
+{
+ char *e;
+ long val;
```

```
+ int res;
+
+ if (!user[0])
+  return -1;
+
+ val = strtoul(user, &e, 10);
+ if (!e[0]) {
+  if (val < 0 || (long) (uid_t) val != val)
+   return -1;
+
+  *uid = val;
+ } else {
+  struct passwd *pw = getpwnam(user);
+  if (pw == NULL)
+   return -1;
+
+  *uid = pw->pw_uid;
+ }
+ return 0;
+}
+
 /*
  * guess_fstype_and_mount()
  * Mount a single file system. Guess the type when unknown.
@@ -715,8 +800,22 @@ guess_fstype_by_devname(const char *devn
  */
 static int
 guess_fstype_and_mount(const char *spec, const char *node, const char **types,
-        int flags, char *mount_opts, int *special, int *status) {
-   struct mountargs args = { spec, node, NULL, flags & ~MS_NOSYS, mount_opts };
+         int flags, char *mount_opts, int *special, int *status,
+         char *user) {
+   struct mountargs args = { spec, node, NULL, flags & ~MS_NOSYS, mount_opts};
+
+   /*
+    * Only set user for the mount if /etc/mtab is a symlink.  Otherwise
+    * user could add submounts without modifying mtab, and so cause
+    * confusion.
+    */
+   args.flags &= ~MS_SETUSER;
+   if (user != NULL && mtab_is_a_symlink()) {
+    if (get_user(user, &args.uid) != 0)
+     return 1;
+
+    args.flags |= MS_SETUSER;
+   }

    if (*types && strcasecmp (*types, "auto") == 0)
```

```
       *types = NULL;
@@ -770,6 +869,9 @@ guess_fstype_and_mount(const char *spec,
 static void
 suid_check(const char *spec, const char *node, int *flags, char **user) {
   if (suid) {
+      if (user != NULL)
+       die (EX_USAGE, _("mount: only root can set user"));
+
       /*
        * MS_OWNER: Allow owners to mount when fstab contains
        * the owner option.  Note that this should never be used
@@ -1057,7 +1159,7 @@ try_mount_one (const char *spec0, const
   char *extra_opts;  /* written in mtab */
   char *mount_opts;  /* actually used on system call */
   const char *opts, *spec, *node, *types;
-  char *user = 0;
+  char *user = NULL;
   int loop = 0;
   const char *loopdev = 0, *loopfile = 0;
   struct stat statbuf;
@@ -1077,7 +1179,7 @@ try_mount_one (const char *spec0, const
   types = types1 = xstrdup(types0);
   opts = opts1 = xstrdup(opts0);

-  parse_opts (opts, &flags, &extra_opts);
+  parse_opts (opts, &flags, &extra_opts, &user);
   extra_opts1 = extra_opts;

   /* quietly succeed for fstab entries that don't get mounted automatically */
@@ -1127,7 +1229,7 @@ try_mount_one (const char *spec0, const

   if (!fake) {
     mnt5_res = guess_fstype_and_mount (spec, node, &types, flags & ~MS_NOSYS,
-           mount_opts, &special, &status);
+           mount_opts, &special, &status, user);

     if (special) {
       block_signals (SIG_UNBLOCK);
@@ -1995,10 +2097,43 @@ main(int argc, char *argv[]) {
  }

  if (getuid () != geteuid ()) {
-  suid = 1;
-  if (types || options || readwrite || nomtab || mount_all ||
-      fake || mounttype || (argc + specseen) != 1)
-   die (EX_USAGE, _("mount: only root can do that"));
+  int pid;
+
```

```
+  pid = fork();
+  if (pid == -1) {
+   die(EX_SYSERR, _("mount: cannot fork: %s"),
+       strerror(errno));
+  } else if (pid == 0) {
+   /*
+    * Child will continue as normal, but first it
+    * drops privileges, and redirects stderr to
+    * /dev/null, because we will retry any error
+    * with the suid privileges.
+    */
+
+   if (verbose > 1)
+    printf(_("mount: trying without privileges...\n"));
+
+   if(setgid(getgid()) == -1 ||
+      setuid(getuid()) == -1)
+    exit(1);
+
+   dup2(open("/dev/null", O_WRONLY), 2);
+  } else {
+   int st;
+
+   wait(&st);
+   if (WIFEXITED(st) && WEXITSTATUS(st) == 0)
+    exit(0);
+
+   suid = 1;
+   if (types || options || readwrite || nomtab ||
+       mount_all || fake || mounttype ||
+       (argc + specseen) != 1)
+    die (EX_USAGE, _("mount: only root can do that"));
+   if (verbose > 1)
+    printf(_("mount: retrying with privileges...\n"));
+  }
   }

  if (!nomtab && mtab_does_not_exist()) {
Index: util-linux-ng/mount/mount_constants.h
===================================================================
--- util-linux-ng.orig/mount/mount_constants.h 2008-01-07 21:40:22.000000000 +0100
+++ util-linux-ng/mount/mount_constants.h 2008-01-07 21:40:50.000000000 +0100
@@ -56,6 +56,12 @@
 #ifndef MS_SHARED
 #define MS_SHARED (1<<20) /* 1048576 Shared*/
 #endif
+#ifndef MS_SETUSER
+#define MS_SETUSER (1<<24)
```

```
+#endif
+#ifndef MS_NOMNT
+#define MS_NOMNT (1<<25)
+#endif
 /*
  * Magic mount flag number. Had to be or-ed to the flag values.
  */
Index: util-linux-ng/mount/umount.c
===================================================================
--- util-linux-ng.orig/mount/umount.c 2008-01-07 21:40:22.000000000 +0100
+++ util-linux-ng/mount/umount.c 2008-01-07 21:40:50.000000000 +0100
@@ -619,9 +619,42 @@ main (int argc, char *argv[]) {
   }

  if (getuid () != geteuid ()) {
-  suid = 1;
-  if (all || types || nomtab || force || remount)
-   die (2, _("umount: only root can do that"));
+  int pid;
+
+  pid = fork();
+  if (pid == -1) {
+   die(EX_SYSERR, _("umount: cannot fork: %s"),
+       strerror(errno));
+  } else if (pid == 0) {
+   /*
+    * Child will continue as normal, but first it
+    * drops privileges, and redirects stderr to
+    * /dev/null, because we will retry any error
+    * with the suid privileges.
+    */
+
+   if (verbose)
+    printf(_("umount: trying without privileges...\n"));
+
+   if(setgid(getgid()) == -1 ||
+      setuid(getuid()) == -1)
+    exit(1);
+
+   dup2(open("/dev/null", O_WRONLY), 2);
+  } else {
+   int st;
+
+   wait(&st);
+   if (WIFEXITED(st) && WEXITSTATUS(st) == 0)
+    exit(0);
+
+   suid = 1;
```

```
+   if (all || types || nomtab || force || remount)
+    die (2, _("umount: only root can do that"));
+
+   if (verbose)
+    printf(_("umount: retrying with privileges...\n"));
+  }
  }

  argc -= optind;
```


Subject: Re: util-linux-ng: unprivileged mounts support
Posted by Mike Frysinger on Tue, 08 Jan 2008 15:31:04 GMT
View Forum Message <> Reply to Message

On Tuesday 08 January 2008, Miklos Szeredi wrote:
> --- util-linux-ng.orig/configure.ac 2008-01-07 21:40:22.000000000 +0100
> +++ util-linux-ng/configure.ac 2008-01-07 21:40:50.000000000 +0100
> @@ -91,6 +92,11 @@ fi
> UTIL_CHECK_LIB(util, openpty)
> UTIL_CHECK_LIB(termcap, tgetnum)
>
> +UTIL_CHECK_LIB(cap, cap_get_proc)
> +if test $have_cap = no; then
> +   AC_MSG_ERROR([libcap is required for mount]);
> +fi

this should be optional and it should be controlled with an AC_ARG_WITH()

with all the non-linux changes that have gone in, does there need to be header checks here and then have mount.c key off of them ? i'm thinking so ...
-mike

Subject: Re: util-linux-ng: unprivileged mounts support
Posted by Samuel Thibault on Tue, 08 Jan 2008 15:47:18 GMT
View Forum Message <> Reply to Message

> with all the non-linux changes that have gone in, does there need to be header
> checks here and then have mount.c key off of them ? i'm thinking so ...

Sure!

Samuel

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers