LXC modification

Introduction

Lxc-execute creates a container with the identifier NAME and execute COMMAND in this container. Lxc-execute command will run the specified command into the container via an intermediate process, **lxc-init**. Lxc-init after launching the specified command, will wait until command ends and all other child processes. (to support daemons in the container). In other words, in the container, **lxc-init** has the pid 1 and the first process of the application has the pid 2. Lxc.init can be start with given UID and GID:

- Sets the UID/GID to use for the init system, and subsequent command, executed by lxc-execute. **These options are only used when lxc-execute is started in a private user namespace.**
- `Defaults to: UID(0), GID(0)`
- **`lxc.init_uid`** `-UID to use within a private user namesapce for init.`
- **`lxc.init_gid`** `-GID to use within a private user namesapce for init.`

The lxc-execute can be run in two modes:

- Privileged container (default behavior) -The privileged container start init.lxc process inside the container as root then spawn to a given command still with root privileges (UID/GID = 0).
- Unprivileged container (with private user namespace)
  - To run unprivileged container we can use lxc.id_map functionality, which maps container UID/GID to host UID/GID
  - lxc.id_map takes 4 params *<u/g> <UID_on_container> <UID_on_host> <range>*
  - The lxc-execute will be run as root but everything inside it (init.lxc) will be mapped to proper user id on host (in this case uid = 200). Unfortunately this also means that the following common operations aren't allowed:
    - mounting some of filesystems/dirs (*/proc, /sysfs, /dev/shm*)
    - creating device nodes
    - any operation against a UID/GID outside of the mapped set
    - accessing socket (this could be resolved by capabilities)

Modification

To fulfill security requirements the lxc.init was modify to change effective UID/GID (drop root in fact) after mounting file system for container, then it spawn the command/process with given UID/GID (also all groups, which user belongs to are red and set). The UID and GID parameters are passed to lxc-execute and from there pass to lxc.init. This allows to have unprivileged containers without separate namespaces and allow us to mount */proc* file system, and using file capabilities access sockets.

The possible configuration:

id_map (enable private user namespace)	**init_gid/init_uid (spawn process with given uid/gid)**	**Behavior**
ENABLE	ENABLE	Unprivileged container (with privet user namespace). The process inside container is not root and is mapped to to proper non-root UID/GID on host. No possibility to mount /proc /syfs or access sockets
ENABLE	DISABLE	Unprivileged container (with privet user namespace). The process is root in container but on host is mapped to proper non-root UID/GID on host. No possibility to mount /proc /syfs or access sockets.
DISABLE	DISABLE	The container is Privileged. The process run inside container has root privileges in container as well in a host.
DISABLE	ENABLE	This behavior was modify. Without modification the process will be privileged in container as well on a host. Enabling init_uid/init_gid does not make a difference cause the privet namespace is not enabled. After modification the lxc.init is run as root to mount /proc and devices. Then it's drop privileges and spawn to given gid/uid.

Dropping root privileges in LXC

This modification was done in 2 separate patches, one of patch drop root privileges for lxc-execute and second for lxc-attach. Both patches use the same function which drop root privileges. This requires that user and group entries for particular user are available in rootfs container in /etc/passwd and /etc/group files.

The function, which drop root privileges, obtain the passwd entry for user with given UID. Based on information obtain from passwd it gates list of group to which user is assigned. Then the all additional groups are set, the setresgid and setresguid function set the proper user and group.

```
/*
* drop_root_privileges(int new_gid, int new_uid)
*     new_gid - new group id for process
*     new_uid - new user id for process
*
* Changing user or group of a process shall only be
* done by a call to setresuid() and setresgid() and
* shall always be confirmed by a call to getresuid()
* and getresgid(). A safe behavior in case of failure
* must be implemented.
*/
static void drop_root_privileges(int new_gid, int new_uid)
{
    uid_t real, eff, saved;
    uid_t uid = (uid_t)new_uid;
    gid_t gid = (gid_t)new_gid;
    gid_t *groups, *effgroups;
    int ngroups=20, neffgroups=20, nmatchedgroups;
    int i,j;
    struct passwd *pw;

    groups = malloc(ngroups * sizeof (gid_t));
    effgroups = malloc(neffgroups * sizeof (gid_t));

    pw = getpwuid(uid);
    if (pw == NULL)
    {
        SYSERROR("NO PASSWD ENTRY AVAILABLE for %d",uid);
        exit(EXIT_FAILURE);
    }

    if (getgrouplist(pw->pw_name, pw->pw_gid, groups, &ngroups) == -1)
    {
        SYSERROR("getgrouplist() returned -1; ngroups = %d\n", ngroups);
        exit(EXIT_FAILURE);
    }

    if (setgroups(ngroups, groups) != 0)
    {
        SYSERROR("Failed setting all user groups");
        exit(EXIT_FAILURE);
    }
```

```
    if (setresgid(gid, gid, gid) != 0)
    {
        SYSERROR("Failed changing GID to %d",gid);
        exit(EXIT_FAILURE);
    }

    if (setresuid(uid, uid, uid) != 0)
    {
        SYSERROR("Failed changing UID to %d",uid);
        exit(EXIT_FAILURE);
    }

    if (getresuid(&real, &eff, &saved) != 0)
    {
        SYSERROR("Failed reading UID");
        exit(EXIT_FAILURE);
    }

    if (real != uid || eff != uid || saved != uid)
    {
        SYSERROR("UID sanity check failed");
        exit(EXIT_FAILURE);
    }

    if (getresgid(&real, &eff, &saved) != 0)
    {
        SYSERROR("Failed reading GID");
        exit(EXIT_FAILURE);
    }

    if (real != gid || eff != gid || saved != gid)
    {
        SYSERROR("GID sanity check failed");
        exit(EXIT_FAILURE);
    }

    neffgroups = getgroups(neffgroups, effgroups);
    if (neffgroups == -1)
    {
        SYSERROR("getgroups() returned -1;\n");
        exit(EXIT_FAILURE);
    }
    else if (neffgroups != ngroups)
    {
        SYSERROR("User groups sanity check failed: mismatch in number of
groups. Expected %d but got %d groups.\n", ngroups, neffgroups);
        exit(EXIT_FAILURE);
    }

    nmatchedgroups = 0;
    for(i=0;i<neffgroups;i++)
    {
        for(j=0;j<ngroups;j++)
        {
            if (effgroups[i] == groups[j])
            {
                nmatchedgroups++;
```

```
                break;
            }
        }
    }
    if (neffgroups != nmatchedgroups)
    {
        SYSERROR("User groups sanity check failed: mismatch in groups. Only
%d groups matched of %d\n", nmatchedgroups, neffgroups);
        exit(EXIT_FAILURE);
    }

    free(groups);
    free(effgroups);

    NOTICE("Dropped root privileges. Now running as UID %u GID %u \n", uid,
gid);
}
```

Hardening of lxc-attach

Several parameters, their help and their implementation are not compiled in when in production or release mode build. Only in debug build they are available. The following options **are gone** in release/production builds:

```
  -e, --elevated-privileges=PRIVILEGES
                      Use elevated privileges instead of those of the
                      container. If you don't specify privileges to be
                      elevated as OR'd list: CAP, CGROUP and LSM (capabilities,
                      cgroup and restrictions, respectively) then all of them
                      will be elevated.
                      WARNING: This may leak privileges into the container.
                      Use with care.
  -s, --namespaces=FLAGS
                      Don't attach to all the namespaces of the container
                      but just to the following OR'd list of flags:
                      MOUNT, PID, UTSNAME, IPC, USER or NETWORK.
                      WARNING: Using -s implies -e with all privileges
                      elevated, it may therefore leak privileges into the
                      container. Use with care.
  -R, --remount-sys-proc
                      Remount /sys and /proc if not attaching to the
                      mount namespace when using -s in order to properly
                      reflect the correct namespace context. See the
                      lxc-attach(1) manual page for details.
      --clear-env     Clear all environment variables before attaching.
                      The attached shell/program will start with only
                      container=lxc set.
      --keep-env      Keep all current environment variables. This
                      is the current default behaviour, but is likely to
                      change in the future.
                      Log pty output to FILE
  -v, --set-var       Set an additional variable that is seen by the
                      attached program in the container. May be specified
                      multiple times.
      --keep-var      Keep an additional environment variable. Only
                      applicable if --clear-env is specified. May be used
                      multiple times.
```

Hardening of lxc-tools

When building in release or production mode, all lxc tools except lxc-attach, lxc-execute and lxc-stop are removed from the package/image. This means the following tools are only available in debug builds:

```
lxc-cgroup
lxc-checkpoint
lxc-console
lxc-create
lxc-device
lxc-freeze
lxc-ls
lxc-snapshot
lxc-unfreeze
lxc-usernsexec
lxc-autostart
lxc-checkconfig
lxc-config
lxc-copy
lxc-destroy
lxc-info
lxc-monitor
lxc-top
lxc-unshare
lxc-wait
lxc-user-nic
lxc-containers
lxc-net
```