wiki.archlinux.org /title/Bubblewrap

Bubblewrap - ArchWiki

13-16 minutes

Bubblewrap is a lightweight sandbox application used by Flatpak and other container tools. It has a small installation footprint and minimal resource requirements. While the package is named bubblewrap, the actual command-line interface is bwrap(1). Notable features include support for cgroup/IPC/mount/network/PID/user/UTS namespaces and seccomp filtering. Note that bubblewrap drops all capabilities within a sandbox and that child tasks cannot gain greater privileges than its parent. Notable feature exclusions include the lack of explicit support for blacklisting/whitelisting file paths.

Warning: Bubblewrap is a tool which provides sandboxing technologies like namespaces and seccomp filter. It does not by default provide a full sandbox that isolates weakpoints like the X11 window system (see #Sandboxing X11). Running untrusted code is never safe, sandboxing cannot change this.

Installation

Install bubblewrap or bubblewrap-git[AUR].

Note:

- For information about user_namespaces(7) support in Arch Linux kernels see Security#Sandboxing applications.
- linux-hardened users may need to install bubblewrap-suid instead of the packages mentioned above. See FS#63316 for more information.

Configuration

Bubblewrap can be called directly from the command-line and/or within shell scripts as part of a complex wrapper. Unlike applications such as Firejail which automatically set `/var` and `/etc` to read-only within the sandbox, Bubblewrap makes no such operating assumptions. It is up to the user to determine which configuration options to pass in accordance to the application being sandboxed. Bubblewrap does not automatically create user namespaces when running with setuid privileges and can accommodate typical environment variables including `$HOME` and `$USER`.

It is highly recommended that you download strace to see what files the program you are trying to sandbox needs access to.

Configuration managers for bubblewrap

Instead of manually setting up the arguments a configuration manager can be used that configure bubblewrap automatically from a simpler configuration.

- **Bubblejail** — Bubblewrap-based sandbox with resource-based permission model (provides GUI to tweak permissions).

  https://github.com/igo95862/bubblejail || bubblejail[AUR]

Usage examples

Please see /Examples for examples on how bubblewrap can be used. Alternatively, there are various projects that demonstrate how bubblewrap can be used for common applications:

- bwscripts
- StandingPad's Bubblewrap scripts

No-op

A no-op bubblewrap invocation is as follows:

```
$ bwrap --dev-bind / / bash
```

This will spawn a Bash process which should behave exactly as outside a sandbox in most cases. If a sandboxed program misbehaves, you may want to start from the above no-op invocation, and work your way towards a more secure configuration step-by-step.

Note: This operation will modify all the owner and group to `nobody` if the owner or group is not the current one, which suggests running some program like `sudo` will not work properly.

Bash

Create a simple Bash sandbox:

- Determine available kernel namespaces

```
$ ls /proc/self/ns
```

```
cgroup  ipc  mnt  net  pid  user uts
```

Note: The presence of `user` indicates that the kernel has exposed support for user namespaces with `CONFIG_USER_NS=y`

- Bind as read-only the entire host `/` directory to `/` in the sandbox
- Create a new user namespace and set the user ID to `256` and the group ID to `512`

```
$ bwrap --ro-bind / / --unshare-user --uid 256 --gid 512 bash
```

```
bash-4.4$ id
uid=256 gid=512 groups=512,65534(nobody)
bash-4.4$ ls -l /usr/bin/bash
-rwxr-xr-x 1 nobody nobody 811752 2017-01-01 04:20 /usr/bin/bash
```

Desktop entries

Leverage Bubblewrap within desktop entries:

- Bind as read-write the entire host `/` directory to `/` in the sandbox
- Re-bind as read-only the `/var` and `/etc` directories in the sandbox
- Mount a new devtmpfs filesystem to `/dev` in the sandbox
- Create a tmpfs filesystem over the sandboxed `/run` directory
- Disable network access by creating new network namespace

```
[Desktop Entry]
Name=nano Editor
Exec=bwrap --bind / / --dev /dev --tmpfs /run --unshare-net  st -e nano -o . %f
Type=Application
MimeType=text/plain;
```

Note: `--dev /dev` is required to write to `/dev/pty`

- Example MuPDF desktop entry incorporating a `mupdf.sh` shell wrapper:

```
[Desktop Entry]
Name=MuPDF
Exec=mupdf.sh %f
Icon=application-pdf.svg
Type=Application
MimeType=application/pdf;application/x-pdf;
```

Note: Ensure that `mupdf.sh` is located within your executable PATH e.g. `PATH=$PATH:$HOME/bwrap`

Filesystem isolation

Warning: It is the bubblewrap user's responsibility to update the filesystem trees regularly.

To further hide the contents of the file system (such as those in `/var`, `/usr/bin` and `/usr/lib`) and to sandbox even the installation of software, pacman can be made to install Arch packages into isolated filesystem trees.

In order to use pacman for installing software into the filesystem trees, you will need to install fakeroot and fakechroot.

Suppose you want to install the `xterm` package with pacman into an isolated filesystem tree. You should prepare your tree like this:

```
$ MYPACKAGE=xterm
$ mkdir -p ~/sandboxes/${MYPACKAGE}/files/var/lib/pacman
$ mkdir -p ~/sandboxes/${MYPACKAGE}/files/etc
$ cp /etc/pacman.conf ~/sandboxes/${MYPACKAGE}/files/etc/pacman.conf
```

You may want to edit `~/sandboxes/${MYPACKAGE}/files/etc/pacman.conf` and adjust the pacman configuration used:

- Remove any undesired custom repositories and `IgnorePkg`, `IgnoreGroup`, `NoUpgrade` and `NoExtract` settings that are needed only for the host system.
- You may need to remove the `CheckSpace` option so pacman will not complain about errors finding the root filesystem for checking disk space.

Then install the `base` group along with the needed fakeroot into the isolated filesystem tree:

```
$ fakechroot fakeroot pacman -Syu \
    --root ~/sandboxes/${MYPACKAGE}/files \
    --dbpath ~/sandboxes/${MYPACKAGE}/files/var/lib/pacman \
    --config ~/sandboxes/${MYPACKAGE}/files/etc/pacman.conf \
    base fakeroot
```

Since you will be repeatedly calling bubblewrap with the same options, make an alias:

```
$ alias bw-install='bwrap                          \
     --bind ~/sandboxes/${MYPACKAGE}/files/ /    \
     --ro-bind /etc/resolv.conf /etc/resolv.conf \
```

```
    --tmpfs /tmp                                \
    --proc /proc                                \
    --dev /dev                                  \
    --chdir /                                   '
```

You will need to set up the locales by editing `~/sandboxes/${MYPACKAGE}/files/etc/locale.gen` and running:

```
$ bw-install locale-gen
```

Then set up pacman's keyring:

```
$ bw-install fakeroot pacman-key --init
$ bw-install fakeroot pacman-key --populate
```

Now you can install the desired `xterm` package.

```
$ bw-install fakeroot pacman -S ${MYPACKAGE}
```

If the pacman command fails here, try running the command for populating the keyring again.

Congratulations. You now have an isolated filesystem tree containing `xterm`. You can use `bw-install` again to upgrade your filesystem tree.

You can now run your software with bubblewrap. *`command`* should be `xterm` in this case.

```
$ bwrap                                         \
    --ro-bind ~/sandboxes/${MYPACKAGE}/files/ / \
    --ro-bind /etc/resolv.conf /etc/resolv.conf \
    --tmpfs /tmp                                \
    --proc /proc                                \
    --dev /dev                                  \
    --chdir /                                   \
    command
```

Note that some files can be shared between packages. You can hardlink to all files of an existing parent filesystem tree to reuse them in a new tree:

```
$ cp -al ~/sandboxes/${MYPARENTPACKAGE} ~/sandboxes/${MYPACKAGE}
```

Then proceed with the installation as usual by calling pacman from `bw-install fakechroot fakeroot pacman ...`.

Troubleshooting

Using X11

Bind mounting the host X11 socket to an alternative X11 socket may not work:

```
--bind /tmp/.X11-unix/X0 /tmp/.X11-unix/X8 --setenv DISPLAY :8
```

A workaround is to bind mount the host X11 socket to the same socket within the sandbox:

```
--bind /tmp/.X11-unix/X0 /tmp/.X11-unix/X0 --setenv DISPLAY :0
```

Sandboxing X11

While bwrap provides some very nice isolation for sandboxed application, there is an easy escape as long as access to the X11 socket is available. X11 does not include isolation between applications and is completely insecure. The only solution to this is to switch to a Wayland compositor with no access to the Xserver from the sandbox.

There are however some workarounds that use xpra or xephyr to run in a new X11 environment. This would work with bwrap as well.

To test X11 isolation, run `xinput test` *`id`* (the keyboard id can be found with `xinput list`). When run without additional X11 isolation, this will show that any application with X11 access can capture keyboard input of any other application, which is basically what a keylogger would do.

The optimal solution to eliminate the X11 weak point is to switch to a Wayland compositor.

Using portals

Warning: This takes advantage of workarounds that "trick" `xdg-desktop-poral` into thinking a sandboxed program is a Flatpak. As a reminder, running untrusted code is never safe, even in a sandbox, even with Portals

With workarounds, it is possible to sandbox programs with XDG Desktop Portals. The main advantage is with filesystem portals, as it makes it possible to not give a program access to the home directory, but still be able to access files. For security reasons however, using portals requires tricking `xdg-`

`desktop-portal` into thinking a sandboxed program is part of a Flatpak. This can be done by adding a `.flatpak-info` file to the sandbox's root filesystem.

In addition, one also needs to run `xdg-dbus-proxy` for more fine control over what portals can be accessed. This should be ran in a sandboxed environment, and as such also needs a `.flatpak-info` file. At the minimum, the proxy needs to have talk access to `org.freedesktop.portal.Flatpak`. Additional portals can be found on the Flatpak documentation.

A common use case is to allow restricting a program from having 100% access to the home directory, and instead only giving access to files and folders the user selects in a file chooser. To achieve this, `xdg-dbus-proxy` can be started with the following arguments:

```
--talk=org.freedesktop.portal.Documents
--talk=org.freedesktop.portal.Flatpak
--talk=org.freedesktop.portal.Desktop
--talk=org.freedesktop.portal.FileChooser
```

Full example:

```
APP_NAME=app.application.Name
APP_FOLDER="$XDG_RUNTIME_DIR/app/$APP_NAME"
mkdir -p "$APP_FOLDER"
set_up_dbus_proxy() {
 bwrap \
   --new-session \
   --symlink /usr/lib64 /lib64 \
   --ro-bind /usr/lib /usr/lib \
   --ro-bind /usr/lib64 /usr/lib64 \
   --ro-bind /usr/bin /usr/bin \
   --bind "$XDG_RUNTIME_DIR" "$XDG_RUNTIME_DIR" \
   --ro-bind-data 3 "/.flatpak-info" \
   --die-with-parent \
   -- \
   env -i xdg-dbus-proxy \
   "$DBUS_SESSION_BUS_ADDRESS" \
   "$APP_FOLDER/bus" \
   --filter \
   --log \
   --talk=org.freedesktop.portal.Flatpak \
   --
call="org.freedesktop.portal.Desktop=org.freedesktop.portal.Settings.Read@/org/freedesktop/portal/desktop" \
   --
broadcast="org.freedesktop.portal.Desktop=org.freedesktop.portal.Settings.SettingChanged@/org/freedesktop/porta
3<<EOF
[Application]
name=$APP_NAME
EOF
}

set_up_dbus_proxy &
sleep 0.1

bwrap \
  ...
  --ro-bind-data 3 /.flatpak-info \
  ...
  3<<EOF
[Application]
name=$APP_NAME
EOF
```

Opening URLs from wrapped applications

When a wrapped IRC or email client attempts to open a URL, it will usually attempt to launch a browser process, which will run within the same sandbox as the wrapped application. With a well-wrapped application, this will likely not work. The approach used by Firejail is to give wrapped applications all the privileges of the browser as well, however this implies a good amount of permission creep.

A better solution to this problem is to communicate opened URLs to outside the sandbox. This can be done using `snapd-xdg-open` as follows:

1. Install snapd-xdg-open-git[AUR]
2. On your `bwrap` command line, add:

```
$ bwrap ... \
  --ro-bind /run/user/$UID/bus /run/user/$UID/bus \
```

```
  --ro-bind /usr/lib/snapd-xdg-open/xdg-open /usr/bin/xdg-open \
  --ro-bind /usr/lib/snapd-xdg-open/xdg-open /usr/bin/chromium \
  ...
```

The `/usr/bin/chromium` bind is only necessary for programs not using XDG conventions, such as Mozilla Thunderbird.

New session

There is a security issue with TIOCSTI, (CVE-2017-5226) which allows sandbox escape. To prevent this, bubblewrap has introduced the new option '--new-session' which calls setsid(). However, this causes some behavioural issues that are hard to work with in some cases. For instance, it makes shell job control not work for the bwrap command.

It is recommended to use this if possible, but if not the developers recommend that the issue is neutralized in some other way, for instance using SECCOMP, which is what flatpak does: https://github.com/flatpak/flatpak/commit/902fb713990a8f968ea4350c7c2a27ff46f1a6c4

Nested namespaces

Certain applications such as Chromium already implement their own sandbox environment using suid helper files. This mechanism will be blocked when they are executed inside a bubblewrap container.

One solution is to have the application use the namespace created by bubblewrap. This can be achieved through zypak[AUR] which is also used by flatpak to run electron based apps inside an additional namespace. Example code that demonstrates how to use zypak with Chromium/Electron can be found at [1].

See also

- GitHub repository
- Seccomp BPF (SECure COMPuting with filters)
- Additional bubblewrap examples