Workshop: Introduction to CLIP OS development

CLIP OS development team

Agence nationale de la sécurité des systèmes d'information (ANSSI)

December 6, 2018

Workshop goals

- Discover CLIP OS source tree and build tools
- Build and run your first CLIP OS image
- Add web server (nginx) support

Installing dependencies

Installing dependencies:

- Supported distributions:
  - Arch Linux,
  - Debian (testing & unstable),
  - Fedora 28 & later,
  - Ubuntu 18.04 & later,
- See https://docs.clip-os.org/toolkit/setup.html#dependencies-installation-on-supported-linux-distributions
- Full details at https://docs.clip-os.org/toolkit/setup.html

Retrieving the source

Retrieving the source (shortcut for this workshop):

```
$ mkdir clipos
$ cd clipos
$ wget http://192.168.10.10/clipos_poss2018.tar
$ wget http://192.168.10.10/clipos_poss2018.tar.sha256sum
$ sha256sum -c clipos_poss2018.tar.sha256sum
$ umask 022
$ tar xf clipos_poss2018.tar
```

Full download (after this workshop):

- Hosted on GitHub: https://github.com/CLIPOS
- See https://docs.clip-os.org/toolkit/setup.html#how-to-fetch-the-entire-source-tree

Source tree

Demo

Build environment

- Enter build environment:

```
$ cd clipos_poss2018
$ source toolkit/source_me.sh
...
...
(toolkit) $
```

First project build

- Start automated build:

```
(toolkit) $ sujust all
```

- Should take about 10 ~ 20 minutes
- Will take about 3 or 4 hours without pre-built package cache (included in the archive for the workshop)

Test with QEMU/KVM

QEMU/KVM as default test target (automated setup)

- Create QEMU image disk and start VM:

```
(toolkit) $ sujust qemu
```

Instrumentation

- Default builds are production builds: no root access & no debug features
- Two instrumentation levels available:
  - development:
    - production-like binaries and configuration
    - full root access
    - debuging tools and debug settings enabled
  - debug:
    - debug symbols and tools
    - non-default settings
    - etc.

Instrumentation

- Increase Core instrumentation level to get root shell access:

```
$ cp toolkit/instrumentation.toml.example instrumentation.toml
$ vim instrumentation.toml
$ cat instrumentation.toml
development = [
    "clipos/core",
]

debug = [
]
```

Second build

- Rebuild everything (will use cached packages):

```
(toolkit) $ sujust all
```

- Rebuild QEMU image and launch VM:

```
(toolkit) $ sujust qemu
```

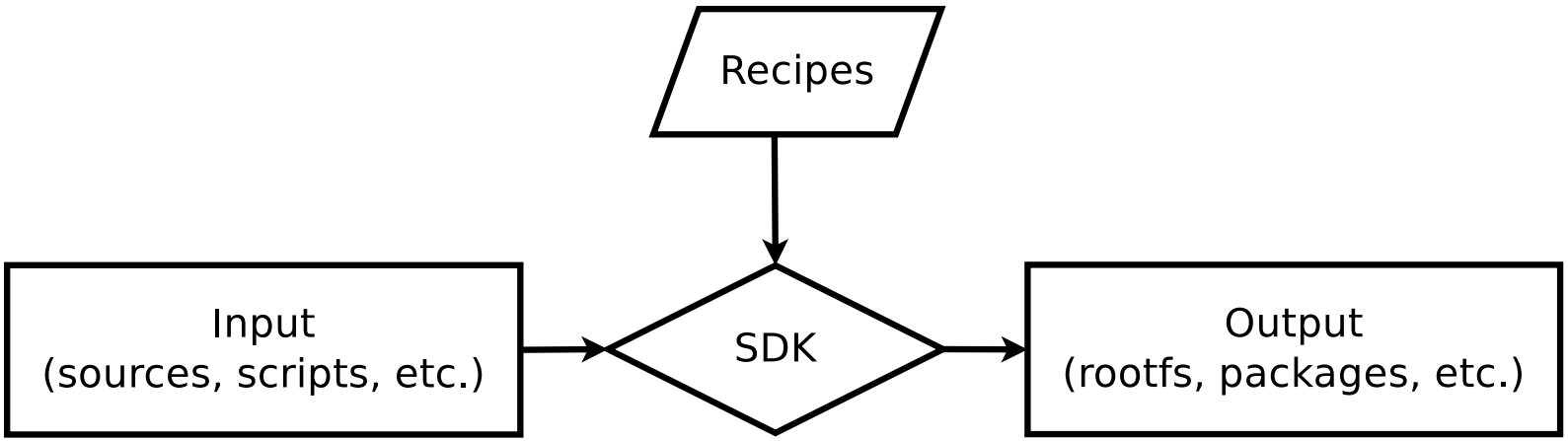
- Login as root (no password)

Full CLIP OS development workflow

- Dependency installation, source retrieval, environment setup
- SDK containers setup (Gentoo Hardened, Debian unstable)
- Project build steps
- Test results with the QEMU/KVM virtual machine

SDK & recipes

- SDK: Gentoo Hardened rootfs with build tools (GCC, etc.)
- Recipes: Scripts run in an ephemeral SDK container



Inside a SDK

- To launch a standalone SDK:

```
(toolkit) $ cosmk run clipos/sdk
[-] interactive run for SDK recipe 'clipos/sdk':
      bash -li
clipos-sdk /mnt/products/clipos/sdk/scripts #
```

- Inside SDK containers: repo root mounted at /mnt

```
clipos-sdk /mnt/products/clipos/sdk/scripts # ls /mnt/
assets   manifest   products   src   toolkit
cache    justfile   out        run
```

Package compilation

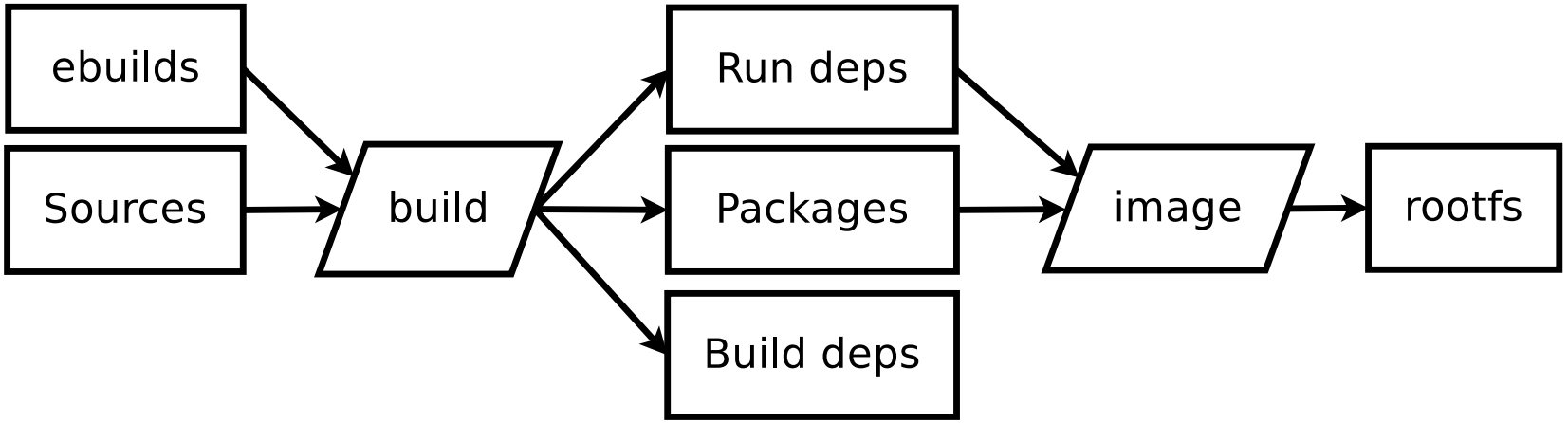
Build and image steps:

- Standard Gentoo build using profiles features
- One profile for each environment (Core, initramfs, cage, etc.)
- Portage based dependency management

Package compilation

rootfs generation in 2 phases:

- Compilation with deps & build deps: generates binary packages
- Re-installation without build deps: final rootfs



Adding nginx to Core partition

- Goal: Add nginx to the Core partition
- Core partition content handled by products/clipos/core recipe
- This recipe will use the Gentoo Hardened SDK to install clipos-meta/clipos-core and all its dependencies to the Core root.
- Will use ebuilds from upstream Gentoo (src/portage/gentoo) & the CLIP OS specific overlay (src/portage/clipos)

Adding nginx to Core partition

- To install nginx to this root, we need to add it as a dependency to clipos-meta/clipos-core:

```
$ vim src/portage/clipos/clipos-meta/clipos-core/c*.ebuild
```

- Try rebuilding core packages only:

```
(toolkit) $ cosmk build clipos/core
```

- Then we can rebuild everything and test that nginx is included:

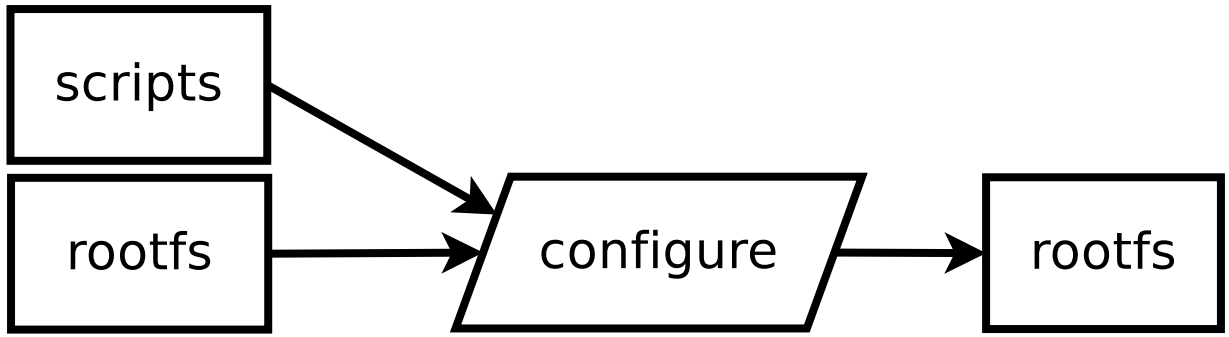
```
(toolkit) $ sujust all
(toolkit) $ sujust qemu
```

```
(qemu)    $ systemctl cat nginx
```

How do we configure CLIP OS?

Configuration step (`configure`):

- Arbitrary configuration step (shell scripts)
- Configuration & customization only: must not compile anything
- Mutate the rootfs output from `image` step to produce final rootfs
- QA checks (strip all SUID, etc.)



Adding nginx user to Core partition

How to add nginx user to Core:

- /etc/passwd & /etc/group on Core partition which is RO
- nginx can not be setup during build & image steps
- Use Core configure step to add nginx user before the rootfs is made RO

Adding nginx user to Core partition

- Add nginx user during Core configuration step:

```
$ vim product/clipos/core/configure.d/21_nginx_user.sh
```

- Tip: Use dbus user configuration as example:

```
$ cat product/clipos/core/configure.d/21_dbus_user.sh
```

- Add the new script to the recipe configuration:

```
$ vim product/clipos/core/recipe.toml
```

- Rebuild (skipping build & image steps) and test:

```
(toolkit) $ sujust cbq
```

Adding nginx config to Core partition

Setting up nginx.conf in RW state partition:

- Configuration that may be changed at runtime must be on RW partition
- State partition will be (almost) empty on first boot
- Store default nginx.conf in RO Core and copy it to RW State on first boot
- Use systemd-tmpfiles configuration

```
$ man 5 tmpfiles.d
```

Adding nginx config to Core partition

- Copy and modify current config from rootfs:

```
$ cp out/clipos/5.0.0-alpha.1/core/image/root/etc/nginx/nginx.conf ...
```

- Create nginx specific configure script:

```
$ vim product/clipos/core/configure.d/60_nginx.sh
```

- Add new custom config in factory folder:

```
install -o 0 -g 0 -m 0750 -d "${CURRENT_OUT_ROOT}/usr/share/factory/"
install -o 0 -g 0 -m 640 .../nginx.conf "${CURRENT_OUT_ROOT}/usr/share/
    factory/nginx.conf"
```

- Rebuild (skipping build & image steps) and test:

```
(toolkit) $ sujust cbq
```

Adding nginx config to Core partition

- Create nginx config dir & copy default config:

```
cat >> "${CURRENT_OUT_ROOT}/etc/tmpfiles.d/nginx.conf" << EOF
d /mnt/state/core/etc/nginx 0750 root nginx
C /mnt/state/core/etc/nginx/nginx.conf 0640 root nginx - /usr/share/
    factory/nginx.conf
EOF
```

- Remove default config & setup symlink:

```
rm "${CURRENT_OUT_ROOT}/etc/nginx/nginx.conf"
ln -s "/mnt/state/core/etc/nginx/nginx.conf" \
    "${CURRENT_OUT_ROOT}/etc/nginx/nginx.conf"
```

- Rebuild (skipping build & image steps) and test:

```
(toolkit) $ sujust cbq
```

Adding nginx folders to State partition

- Create /var/lib/nginx and subfolders in state partition:

```
cat >> "${CURRENT_OUT_ROOT}/etc/tmpfiles.d/nginx.conf" << EOF
d /var/lib/nginx     0750 root  nginx
d /var/lib/nginx/www 0750 root  nginx
d /var/lib/nginx/tmp 0750 nginx nginx
EOF
```

- Rebuild (skipping build & image steps) and test:

```
(toolkit) $ sujust cbq
```

Test your web server!

```
(qemu) $ ip a
...
192.168.0.XX
...
(qemu) $ echo "Hello from CLIP OS!" > /var/lib/nginx/www/index.html
(qemu) $ systemctl start nginx
```

```
$ curl 192.168.0.XX
Hello from CLIP OS!
```

Adding custom nginx service unit to Core partition

- Copy and modify current nginx service unit from rootfs:

```
$ cp out/clipos/5.0.0-alpha.1/core/image/root/.../nginx.service ...
```

- Install customized service unit:

```
rm "${CURRENT_OUT_ROOT}/lib/systemd/system/nginx.service"
install -o 0 -g 0 -m 0644 .../nginx.service \
    "${CURRENT_OUT_ROOT}/etc/systemd/system/nginx.service"
```

- Rebuild (skipping build & image steps) and test:

```
(toolkit) $ sujust cbq
```

Harden nginx service unit

- Confine nginx using systemd service hardening features:

```
NoNewPrivileges=yes
PrivateTmp=yes
PrivateDevices=yes
ProtectHome=yes
ProtectKernelTunables=yes
ProtectKernelModules=yes
ProtectControlGroups=yes
RestrictNamespaces=yes
SystemCallArchitectures=native
SystemCallFilter=~@clock @cpu-emulation @debug @keyring
    @module @mount @obsolete @privileged @raw-io
RestrictAddressFamilies=AF_UNIX AF_INET
MemoryDenyWriteExecute=yes
```

Merge configuration changes in the nginx ebuild

Integrate changes made in the configuration step in a custom nginx ebuild:

- Copy nginx ebuild & files from Gentoo portage tree into clipos overlay
- Bump nginx ebuild revision to overide Gentoo ebuild
- Import most changes from configure step
- Rebuild (including build & image steps) and test:

```
(toolkit) $ sujust all && sujust qemu
```

Further nginx hardening options

- Start nginx as non-root user with capabilities:

```
User=nginx
Group=nginx

CapabilityBoundingSet=
CapabilityBoundingSet=CAP_NET_BIND_SERVICE
AmbientCapabilities=
AmbientCapabilities=CAP_NET_BIND_SERVICE
```

- Use non-root own /run/nginx folder:

```
cat >> "${CURRENT_OUT_ROOT}/etc/tmpfiles.d/nginx.conf" << EOF
d /run/nginx          0750 nginx nginx
EOF
```

Further nginx hardening options

- Chroot child processes:
  - Start with: https://github.com/clipos-archive/clipos4_portage-overlay/blob/master/www-servers/nginx/files/nginx-1.7.6-clip-chroot.patch
  - Add CAP_SYS_CHROOT & drop all capabilities afterward (requires CAP_SETPCAP).

Thanks!

✉ clipos@ssi.gouv.fr

- Website: clip-os.org
- Docs: docs.clip-os.org
- Sources: github.com/CLIPOS
- Bugs: github.com/CLIPOS/bugs

We're hiring! (but not directly for CLIP OS)

Linux system security expert

https://www.ssi.gouv.fr/emploi/expert-en-securite-des-systemes-linux/