inside heads

Ivan J. (parazyd@dyne.org)

March 18, 2017

Guidelines for a heads developer.
Revision 1.2
`https://heads.dyne.org`

Preface

In a world of bloat, pain, and undocumented things, I hope to have this booklet here to help you and me in our endeavors with heads. The general philosophy heads follows is minimalism, elegance, non-intrusion, and cleverness. Strive with those points and keep your mind sane.

inside heads is a collection of hand-picked brain farts that will be helpful for anyone wanting to develop heads. So let's dive into it...

Contents

1 Introduction

heads is a libre GNU/Linux distribution loosely based around the concepts of the Tails[1]GNU/Linux distribution. heads strives to be a minimal live distribution for activists, journalists, geeks, and privacy-aware people in general. This is mainly achieved by utilizing the Tor[2]network to the maximum. It is one of the key features: routing **all** network traffic in heads through the Tor network. This enables one to stay as anonymous as possible when using the Internet.

Though important, anonymity is not the only thing needed. Security is an important aspect as well. heads tries to achieve it by using a grsecurity[3]-patched kernel, and offering only libre software. While libre software can still be malicious, chances are, it is not. At least less possible than proprietary software which is much more difficult to audit (since one can not read its code).

All of these quirks and features will be mentioned further in the document.

[1] `https://tails.boum.org`
[2] `https://www.torproject.org`
[3] `https://grsecurity.net`

2 The heads blend

At the core of the heads build system is a file called **heads.blend**. A *blend* is a concept I developed for libdevuansdk[4]. Since you probably have no idea what libdevuansdk even is, I'll start by getting a bit into that first.

2.1 libdevuansdk

libdevuansdk is a shell script library that emerged when I began maintaining the Devuan SDK[5]. The Devuan SDK[6]is a small framework written in zsh, and currently consists of libdevuansdk as such, and wrappers that are supposed to be written around it. Currently: *arm-sdk*, *live-sdk*, and *vm-sdk*. heads' build system utilizes the live-sdk.

libdevuansdk gathers common knowledge that is utilized by the beforementioned sdks, and as the name says - is a common library of sorts. All functions that are not declared in the heads blend, are either a part of live-sdk or libdevuansdk. The concept is such that the blend can override functions of what is above it (live-sdk, and then libdevuansdk). live-sdk does the same. This makes it easy to do situation-specific overrides while not breaking the API.

2.2 Configuration

The first thing the blend does is sourcing a config file. The config file is where all the software versions, their locations, list of packages to install and remove are kept.

extra_packages is an array of Devuan package names that we append because it is already a part of libdevuansdk and live-sdk. This array holds all software installed in heads that exists as a package in the Devuan repositories.

purge_packages is an array of packages that should be purged from the system when it's the time to do it (in a certain step of libdevuansdk).

finalize_purge_packages is an array of packages that should be purged from the system

at the final blend step. It is an array specific to the blend.

[4] `https://git.devuan.org/sdk/libdevuansdk`
[5] `https://git.devuan.org/sdk/`
[6] Simple Distro Kit

2.3 From start to finish

The basic workflow of functions to bootstrap an entire system is the following (the functions whose names don't start with "blend" are either parts of live-sdk or libdevuansdk):

```
bootstrap_complete_base
blend_preinst
iso_prepare_strap
build_kernel_${arch}
iso_setup_isolinux
iso_write_isolinux_cfg
blend_postinst
iso_squash_strap
iso_xorriso_build
```

I will not cover the foreign functions here, only the blend-specific.

Currently in *blend_preinst* there is only the creation of the user "luther", so that's that.

build_kernel_arch is architecture specific (i386/amd64) and will be explained in the kernel chapter.

In *blend_postinst* there is more magic happening. Here we compile and install packages from source to the system, clone the *rootfs overlay* and do the final fixes and hacks. The rootfs overlay is also a chapter of its own.

2.4 heads' kernel

The kernel heads uses is a non-standard one as some might have expected. It's a specific version, patched with grsecurity and uses a configuration specifically tailored for heads. The sources (and the git log) can be found here: `https://github.com/headslive/linux-heads`).

Luckily, as heads is using git[7]for version control, it's not needed to have separate patches stored somewhere as they are now easily available from the git log. The deblobbing scripts from linux-libre[8], however, are downloaded from their website.

Before I mentioned a function called **build_kernel_arch**. It is somewhat like a function pointer and the function gets called depending on what architecture is declared for the current build. This function will clone the kernel git repo (or pull the latest master if it is already there), compile it, and install it along with the modules in the heads bootstrap directory. It is a function already declared in libdevuansdk, but here we override it since we are building a custom kernel.

2.5 Kernel configuration

To be written.

[7] `https://github.com/headslive/`

[8] `http://www.fsfla.org/ikiwiki/selibre/linux-libre/`

3 The rootfs overlay

If the blend is the brains of heads, then the rootfs overlay is the heart of heads[9]. This overlay is a git repository residing in the root directory of heads' filesystem. It holds all specific configuration files, scripts, sources, and other miscellaneous files that are a part of heads. `https://github.com/headslive/rootfs-overlay`

3.1 Scripts

heads-specific scripts usually reside in */usr/local/bin/heads-**. They are mostly small helper scripts that I will document here for the sake of reference.

3.1.1 heads-generate-passphrase

Generates a random passphrase for the root user on every boot. Drops it once in luther's .zshrc and shows it only once when a terminal is opened and is lost unless it's remembered or changed. Executed by */etc/rc.local*.

3.1.2 heads-init

A tiny case clause, used by shutdown-menu to either shut down or reboot.

3.1.3 heads-shutdown-menu

A script that utilizes zenity[10]to give a graphical shutdown menu from AwesomeWM.

3.1.4 heads-torify

The iptables black magic we use to route all network traffic through Tor. Executed by */etc/rc.local*.

3.1.5 heads-update

A script to update the system. Explained in the *updating heads* section.

3.2 Updating heads

As mentioned, the rootfs overlay is a git repository. We can utilize this to provide seamless updates to users, by pushing commits to the rootfs overlay git repository. Keep in mind heads as such is **never** going to phone home[11], but it's up to the user to choose if they will update or not. In any case, these will mostly be minor updates from heads' side, and if there are any important (read: security) updates, we will rather roll out a new release than relying on people updating their rootfs overlay.

However, since git is awesome (read: decentralized), it's easy to host one's own rootfs overlay if one wants to customize their heads setup, have a certain kind of persistence in the system or just easily grab files when one boots into the live system. The possibilites are endless.

[9] such organs, much anatomy

[10] `https://help.gnome.org/users/zenity/stable/`

[11] `https://en.wikipedia.org/wiki/Phoning_home`

4 Building heads

A little bit of spoonfeeding is always nice. Let's go through the process of building a heads ISO... Learning this process is essential, but don't worry, it's the easiest process in the world of building ISOs.

4.1 Obtaining the build-system

I use a Devuan based system to bootstrap heads images, and I can't promise it's going to work anywhere else (It should, but I haven't tried it ever, nor do I plan to). So let's dive into this beautiful process of obtaining the heads build system and baking our ISO :)

First off, we need to grab the *build-system* git repository. It contains git submodules, so we append –recursive to the args:

```
$ git clone https://github.com/headslive/build-system.git --recursive
```

This will give us all we need. Check out the README to find out what dependencies you need to install to run it on your system. If you are on the master git branch, you are using the development version - that's what you should use for testing and development, but if you require an image for daily usage, you should probably checkout the latest tag you can find in the git history.

4.2 Firing up live-sdk

Okay, so once we have our git repo cloned and possibly checked out, we go into the *live-sdk* directory of the repo. From there you start a vanilla zsh shell, source live-sdk to enter its environment and initialize the blend to build heads:

```
$ cd live-sdk
$ zsh -f
$ source sdk
$ load devuan amd64 heads
```

Once done, you will be notified all is loaded and ready to run. There is a helper command that will run the following functions in sequence, and exit if one of them exits. It is called **build_iso_dist**.

4.2.1 bootstrap_complete_base

This function is the one that provides us with a vanilla base system that is as minimal as it can be, and then we build up upon it. It uses our *extra_packages* array and installs our packages. The function also creates a tarball of the minimal system so it is not needed to debootstrap for every build. If the tarball is found when starting a build, it will be extracted and updated.

4.2.2 blend_preinst

This one was mentioned before. It is blend-specific, and currently only adds the "luther" user.

4.2.3 iso_prepare_strap

A function that will install the packages needed for a liveCD to function properly using *apt*.

4.2.4 build_kernel_arch

Mentioned before as well. This will clone (or update) the latest *linux-heads* sources, compile them, and install them into the heads bootstrap directory (our filesystem). The function exists in libdevuansdk, but is overriden with the heads blend.

4.2.5 iso_setup_isolinux

After the kernel is installed, we have to copy it for isolinux to use it. This function also copies the needed isolinux binaries where they have to be.

4.2.6 iso_write_isolinux_cfg

This function drops the isolinux configuration file into the isolinux directory. Exists in libdevuansdk, but is overriden with the blend because of branding.

4.2.7 blend_postinst

Mentioned in an earlier chapter. This function calls other *blend_install_software* functions to compile and install specific software like Tor, GNU IceCat and such from source, because

their packages aren't provided in Devuan. It also clones the *rootfs-overlay* and sets it up. Finally, it executes the hacks and fixes that can be found in the *blend_finalize* function, and cleans up the filesystem, making it ready for packing. The function is blend-specific.

4.2.8 iso_squash_strap

After our filesystem is ready, this will create a SquashFS from the bootstrap directory. This is what holds our read-only live system.

4.2.9 iso_xorriso_build

Finally, this function creates a bootable ISO from our SquashFS and the ISOLINUX files we have copied earlier. The ISO can be found in *dist/* after xorriso is finished.

5 Handling bugs

Yes, a bunch of bugs.
`https://github.com/headslive/bugtracker`

5.1 The bugtracker

heads doesn't have the usual Bugzilla bugtracker or similar, it rather utilizes GitLab issues to track bugs, feature requests and such. You can find the link above.

5.1.1 Reporting bugs

To report bugs, you should be registered at `https://github.com`. Although, if you are a heads developer, my guess is you already are registered. To report a bug, you simply open a new issue in the bugtracker repository. Keep in mind it's always a good idea to search through existing bugs before reporting anything. It will make me hate you a little bit less[12]. Make sure to write your bug report as descriptive as you can, and if applicable, tag it accordingly. Take look at the existing issues as a guideline.

5.1.2 Fixing bugs

To fix bugs, search through the existing issues and find out how to resolve them. Simple as that. Once you think you have a bugfix, report it on the issue and it will be reviewed.

[12] I love you all equally though.

6 Talking heads

Discussion around and about heads can be done on IRC[13]and the mailing list[14]. These are the prefered way of communication about all kinds of topics. IRC is a good place to ask if you are not sure if you should report a bug or not - for example. The mailing list is a good place to talk about anything. But again, if it's a bug, just report it on the bugtracker.

So that's it. I hope this has helped you at least somehow. If you feel like contributing anything (time or bitcoins), please see: `https://heads.dyne.org/contribute.html`

Happy hacking!

[13] `https://heads.dyne.org/irc.html`
[14] `https://mailinglists.dyne.org/cgi-bin/mailman/listinfo/heads`