DevBox and HackBox

Last Revised: August 8, 2016

Contents

1 Introduction

When doing operating system work you ideally want to run your experimental software on a different computer than the one you use for development. An error in an operating system module or driver has the potential to corrupt the entire machine; testing on your development system may lead to headaches if such corruption should occur. Also it is generally not possible to debug an operating system while running the debugger tool on the same machine. Two machines are needed so the debugger and the debugee can be properly separated.

In this document I call the development system "DevBox." It is a full desktop system with all the usual tool and conveniences. It is on DevBox where you spend most of your time working and compiling your programs. It is on DevBox where you run source level debuggers.

The experimental system I call call "HackBox." It is not a full desktop system and instead is minimally configured. The smaller, lighter configuration makes testing and error recovery easier.

This document describes how to set up the preconfigured DevBox and HackBox virtual machines I provide for my operating systems course. I describe how to install the VirtualBox VM software, how to import and run the DevBox and HackBox virtual machines, and how to exercise the system to verify that it is set up properly. At the end of this document I include some notes on how DevBox and HackBox were built. That information is for reference only and not essential for using the systems.

2 Installing VirtualBox

The lab machines should already have the VirtualBox VM software installed. If you are using a lab machine you can skip this section and continue with Section 3. If you intend to run DevBox and HackBox on your personal machine you will need to first install VirtualBox on your system.

DevBox and HackBox together consume significant resources. Before trying to install them on your personal machine you should be sure you have at least 4 GiB of memory and at least 50 GiB of free disk space. You also need to have hardware virtualization support turned on in the BIOS of your host computer. Most new machines come with this feature turned on by default. Older machines may require you to enter the BIOS and activate this feature. If your machine is very old it may not support hardware virtualization at all. Finally, DevBox and HackBox assume that your host has at least two cores.

If the host does not meet the requirements above it may be possible to reconfigure the virtual machines so that they will work anyway. However you may have to sacrifice some features or endure suboptimal performance in that case.

If you still wish to proceed, begin by downloading VirtualBox from `http://www.virtualbox.org`. Be sure to download both the main installer for your system *and* the Extension Pack. The Extension Pack adds important functionality that DevBox and HackBox requires. *The virtual machines will likely not boot in their default configuration without the Extension Pack installed!*

Run the VirtualBox installer. After the installer completes double click on the Extension Pack to install that as well.

3 Booting the Guests

DevBox and HackBox are distributed as virtual appliances. This is a standard file format that includes not only the virtual hard disk (compressed) but also the virtual machine's initial configuration. Once you import the virtual appliance into your virtualization software you can boot a virtual machine by just starting it as you might start a real computer.

The virtual appliance file format is accepted by several different virtualization products. However, DevBox and HackBox assume they are running under VirtualBox, and, in particular, DevBox has the VirtualBox "guest additions" pre-installed.[1] The guest additions are a collection of software components that are loaded into the guest and that streamline the operation of the system.[2]

You may be able to run DevBox and HackBox using some other virtualization product. However, you will probably have to do some additional configuration before it will work well. VirtualBox is the only virtualization software that is officially supported for use in my operating systems class.

To import the virtual appliances into VirtualBox follow the steps below.

1. Download the file `DevBox-YYYY-MM-DD.ova` and similarly for HackBox. See my web site for the locations where these files are currently available. The names of the files contain the date when that version was released. I normally release a new version of both VMs shortly before the start of each semester.

2. Start VirtualBox and select "Import Appliance" from the File menu. Follow the prompts. This will unpack the OVA file and add the virtual machine to VirtualBox's start menu. Repeat this for both virtual appliance files.

In principle no further configuration is necessary since the configurations of the virtual machines are contained in their original OVA files. However you might review the machine settings and tweak them if desired. Note especially the amount of memory allocated to the virtual machines. *DevBox is configured to use 1.5 GiB of memory and HackBox is configured to use 0.5 GiB of memory. If your host computer has less than 4 GiB of memory you may want to adjust the configured value downward.* If your host computer is well endowed with memory you might

[1] HackBox does not have the guest additions installed to simplify its kernel configuration.

[2] The term "guest" refers to the system running inside the virtual machine. The term "host" refers to the system running the virtualization software.

consider increasing the amount available to the virtual machines. A good rule of thumb is to allocate no more than 50% of your system's physical memory to all virtual machines taken together.

After you have imported the appliances you can delete the OVA files to save disk space. However, if you have sufficient disk space you might consider keeping the files in case you need to reinitialize DevBox or HackBox from scratch. Having the OVA files on hand will save you another long download.

It is perfectly reasonable to boot both DevBox and HackBox at the same time; in fact this is necessary for the kind of work you'll be doing. Each virtual machine runs in its own window. DevBox presents a full graphical desktop; HackBox is console only. The Linux distribution running inside both virtual machines is Ubuntu, 64 bit. DevBox is using the desktop version of Ubuntu 14.04 and HackBox is using the server version of Ubuntu 14.04.

You will likely have to boot HackBox first if you want the serial consoles to communicate with each other properly. This is because HackBox's machine configuration sets up its virtual serial port in server mode; DevBox will probably be unable to connect to it unless it already exists when DevBox starts. *TODO: Check this!*

Both DevBox and HackBox has a user account named "student" with a password of "frenchfry." You should log in as this user. The student user can use `sudo bash` to create a root shell when necessary.

4 Snapshots

One major benefit to doing your development inside a virtual machine is that you can use VirtualBox to "snapshot" your system just before attempting any kind of dangerous or complicated operation. When you create a snapshot VirtualBox remembers the entire state of the system. *Any* change made after the snapshot is provisional. If the system becomes corrupt, you can just restore to the snapshot and undo all changes made since the snapshot was taken.

The undoing of changes is complete. The process does not rely on the correct operation of the guest system. Even if the data on the (virtual) hard disk is totally shredded, restoring to a snapshot will reset every detail of the system back to the state it had when the snapshot was taken.

With this protection in place you are free to experiment without concern of causing irreparable damage. For example, if you want to try building and installing a new version of the C library... go ahead! Take a snapshot first and if the result is a major disaster you can just roll back to where you started and try again. In the worst case scenario you could delete your virtual machine and re-import it from the original OVA file. Of course this rolls back all changes you ever made to the system but the base configuration will be fully restored.

It is likely you will snapshot HackBox frequently during your work. This is because HackBox will be running experimental kernels and kernel modules and is thus subject to random, spectacular failures. In contrast DevBox should remain fairly stable since it only runs software blessed by Ubuntu and well established third party products. Because of HackBox's minimal configuration, snapshots of HackBox should be small and quick to make. This is a nice side effect of using two machines in this way.

5 Basic Testing

It is nice after installing DevBox and HackBox to do some simple operations to verify that they are working for you in a useful way. Keep in mind that none of the steps described in this section are necessary for DevBox or HackBox to work. They are only intended to give you an opportunity to exercise the system in a relevant manner.

1. If any virtual machine is running, shut it down. If you are not running VirtualBox, start the VirtualBox application. Either way you should be looking at VirtualBox's window that lists your

configured virtual machines, but you should have no virtual machine actually running.

2. In the VirtualBox console select the HackBox virtual machine and take a snapshot of its current state.

3. Boot HackBox. Log in as the user student.

4. Boot DevBox. Log in as the user student. Open a terminal window.

5. Check that you have network connectivity between the two systems. For example, on DevBox issue the command:

   ```
   $ ping hackbox
   ```

   Run the corresponding command on HackBox to ping DevBox.

   Both virtual machines can communicate across an internal network managed entirely by VirtualBox. They have statically assigned IP addresses (192.168.56.101 for DevBox and 192.168.56.102 for HackBox) with appropriate entiries in `/etc/hosts` allowing the machines to communicate via unqualified names. This ping check allows you to verify that this network connectivity is functional.

   Both virtual machines also have a second network interface defined that connects via a network address translator to the host's physical network interface. This gives both machines access to the Internet for updates and other purposes.

6. On DevBox issue the command:

   ```
   $ ssh hackbox
   ```

   This should connect to the ssh server on HackBox over the internal network and allow you to log in again as student. You can use the `scp` command to transfer files between the two systems. Note that DevBox is not running an ssh server so all file transfer commands must be initiated from the DevBox side.

   Once you've demonstrated that ssh is working you can log out.

7. On DevBox issue the command:

   ```
   $ minicom
   ```

   Both HackBox and DevBox also have virtual serial ports defined. Those ports (`/dev/ttyS0` in both cases) have been connected together by VirtualBox. HackBox is running a `getty` process on its ttyS0 serial port to support logins.

   Minicom is a simple terminal program for Linux that can be used to interact with serial ports. After starting minicom you should be able to hit Enter and see a login prompt from HackBox. Try logging in to verify that this works. We will make use of this serial connection for kernel debugging purposes (the network will be non-functional during debugging).

   When you are done experimenting with this, log out of HackBox and then type CTRL+A followed by X in minicom to exit minicom.

8. Check that you are using the correct kernel on HackBox by issuing the command (on HackBox's console):

   ```
   $ cat /proc/version
   ```

   This command reads a file in the "magic" proc file system that, in this case, reports information about the kernel version you are using. It should say 3.18.14. This is the pre-compiled experimental kernel we will be working with.

9. On DevBox use the file manager to browse

   ```
   /home/student/linux-3.18.14
   ```

   This is the source code for the experimental kernel. You will become familiar with its organization in the future, but a quick look now is a good first step toward that goal.

10. On DevBox in a terminal window change to

    ```
    /home/student/linux-3.18.14
    ```

    and issue the command `make menuconfig` to view the kernel configuration menu. The configuration you are looking at is the configuration

used the last time the kernel was compiled. It represents the current configuration of the experimental kernel running on HackBox.[3] Look around to see what kinds of options are available. If you change anything, do *not* save your changes. If you save any changes you will end up doing a full kernel rebuild the next time you try to compile it. That takes a rather long time.

Once you exit the configuration menu, look at the file `.config` and the file

`include/generated/autoconf.h`

to see the results of the configuration process. The `.config` file is used by the kernel build system to control which source files need to be compiled and how. The `autoconf.h` header is included into source files that need to distinguish between various configuration options.

11. Edit the file `init/version.c`[4] and change the value of `linux_proc_banner` to include some distinctive text. For example you might add something like "experimental" to the existing banner. This will change the contents of `/proc/version` once the modified kernel is running.

12. Issue the command `make` at the root of the kernel source tree to build a new kernel reflecting your changes. This should not take too long because the kernel has already been built and `make` should realize that most of the object files are up to date (even so it can take several minutes to do this). If `make` appears to be recompiling everything something is wrong (possibly you accidentally saved changes when looking the configuration earlier). In that case just cancel the build and don't worry about this step.

13. Copy the file:

    `arch/x86/boot/bzImage`

    to HackBox and on top of the file:

    `/boot/vmlinuz-3.18.14`

    on that system. This replaces the boot experimental kernel with the version you just compiled containing your changed banner. It's good practice to make a backup copy of `vmlinuz-3.18.14` first.

14. Reboot HackBox into the experimental kernel and verify that `/proc/version` has changed as expected. You can use the command:

    ```
    $ sudo shutdown -r now
    ```

    to reboot HackBox immediately.

When you are done with the steps above you can shutdown HackBox using a command such as:

```
$ sudo shutdown -h now
```

Here the `-h` option means "halt." After the system shuts down you can restore to the snapshot to undo any changes made during this session and put the system back into its initial pristine state. You may find it useful to do this after each experiment.[5]

Although you will be recompiling the kernel in some labs, most of the programming you'll be doing will actually be in the form of external kernel modules. These are modules that are not part of the normal distribution and that are always loaded dynamically. Details about how to do this will be provided in the appropriate lab.

In addition to Linux we will also be experimenting with an operating system written by a team of VTC students for their 2008/2009 senior project. That system is called Phoenix. Using Phoenix does not require HackBox at all. Instead it runs inside a virtualization product called Bochs running inside the DevBox VM (yes, this amounts to a nested virtual machine).

If you wish to try Phoenix to see if it works for you, follow the steps below.

[3] For full details about how to compile Linux, see my companion document *Compiling Linux* on my web site.

[4] Relative paths are relative to the root of the kernel source tree unless context indicates otherwise.

[5] Avoid storing anything of importance on HackBox. Use DevBox for all files you wish to keep.

1. On DevBox change to the `Phoenix` directory and do:

   ```
   $ git pull
   ```

   to get the most recent version of Phoenix from GitHub.

2. Type:

   ```
   $ source useOW.sh
   ```

   to configure the environment of your terminal to make the Open Watcom C/C++ compilers available. You might want to change the title on the terminal to reflect this so you don't forget (use the "Terminal → Set Title" menu item on the terminal program).

3. Use the `makeandrun.sh` script in the top level Phoenix folder. That script creates a Phoenix boot disk image containing the system and several sample programs, and then launches Phoenix using the Bochs simulator. Bochs has been configured with debugging features turned on so when it starts it will produce a command prompt. Type "continue" to begin the simulation without interruption.

   When Phoenix boots you will be prompted with a menu of programs on the boot disk. Select one of them to see Phoenix in action. You can power off the Bochs simulation (upper right corner of the Bochs window) when you are done. There is no shutdown procedure for Phoenix.

6 Using KGDB and KDB

KGDB is a kernel debugger that allows you to connect `gdb` running on DevBox to the kernel running on HackBox. Using this tool is a bit intricate. Here I outline the basic procedure. See the class web page for links to additional information.

When you want to use the debugging tools you must log into HackBox by way of a serial terminal. Do *not* use SSH and do not use the console. While the HackBox kernel is being debugged the network will not function normally. Furthermore the debugger on DevBox has no way to transmit data over the console; it needs to use an old style serial port.

I recommend taking a snapshot of HackBox before your debugging session in case you accidentally trash HackBox during the session!

On DevBox use the command:

```
$ minicom
```

This command starts the terminal program. It has been preconfigured to connect to `/dev/ttyS0` which is attached to HackBox's serial port of the same name by way of a host-level named pipe. You can hit the Enter key to get a login prompt from HackBox.

When you are ready to debug, you must first activate KGDB on HackBox. Use a command such as:

```
# echo ttyS0 > \
   /sys/module/kgdboc/parameters/kgdboc
```

This informs the KGDB "over console" driver that it should use ttyS0 for communication. Without this step KGDB will effectively be turned off even if it has been compiled into the kernel. Note that this command must be executed at a root shell (using `sudo` won't work).

There are three ways to break into the kernel.

1. If the kernel takes an exception it will stop and wait for the debugger to attach.

2. You can issue a SysRq-g sequence in the terminal program.

3. You can echo the letter 'g' into the file `/proc/sysrq-trigger`.

The second option can be done from minicom by typing Ctrl+Afg. Use the following command to execute the third option:

```
# echo g > /proc/sysrq-trigger
```

After breaking into the kernel you should see a KDB prompt on your terminal. At this point the kernel on HackBox is frozen awaiting your debugging pleasure. You could enter KDB commands, but it is generally more interesting to use `gdb`.

On DevBox in a separate command window, go into the top level of the Linux source tree and use the command:

```
$ gdb ./vmlinux
```

This runs the debugger against the previously compiled kernel image. The debugger will use the image for symbolic information. Note that you must use the uncompressed kernel image and not `vmlinuz`.

Connect the debugger to the ttyS0 serial port with the following commands:

```
(gdb) set serial baud 115200
(gdb) target remote /dev/ttyS0
```

You can now use `gdb` debugging commands to debug the HackBox kernel. Note that you will see some "junk" appearing on the previously opened terminal. They are remote debugging protocol packets; you can ignore them.

To return to normal operation first exit `gdb` with the command:

```
(gdb) quit
```

You will see a message about terminating the remote process. You can say 'yes' here (it doesn't actually terminate the HackBox kernel). Then, back in the terminal window type:

```
$3#33
```

You won't be able to see the text as you type it. This sequence is a debugging protocol packet that tells KDB to return to the prompt. You can then resume normal operation of HackBox by issuing the 'go' command at the KDB prompt.

7 Making Backups

Most recovery operations such as restoring to a snapshot or re-importing the original OVA file will entail the lose of some or all of your work. Thus I strongly recommend that you back up your work often. This can be done by using a special backup script I created. To use it simply execute `backup` at a command prompt.

It is extremely important to understand that The `backup` script only archives the files in the `cis-4020` folder. This means the backup archives are relatively small but it also means that any files you put elsewhere on the system will not be backed up. *I therefor strongly recommend that you store all course materials in the cis-4020 folder.* Be aware that configuration changes you make, for example to the desktop on DevBox, are not backed up.

Once the `backup` script has created the archive it will ask you if you want to tranfer that archive to `lemuria`. Assuming you have an account on that system you can use `lemuria` as a repository of backups. If you do not transfer the backup file, the script will leave it in student's home directory where you can transfer it some other way (for example as an email attachment using webmail).

You should definitely transfer the backup archives off the virtual machine. The point of the backups are to save your work in case the VM is destroyed. Keeping the backup archives on the VM won't help you if you lose the state of the VM itself.

Figure 1 shows a sample backup session. Text entered by the user is show in an italic font.

Another, more elegant way to transfer a course directory to another host is to use the `rsync` program. This program only copies files that have changed and is thus often faster than transferring an entire archive (even a compressed archive). The `rsync` program will add and remove files and directories on the target as necessary and, in archive mode, it even copies file permissions and date/time information. First move to student's home directory. Then do

```
student@devbox:~$ backup
1) None
2) cis-4020
=> 2
Creating backup file for cis-4020...
Done
Transfer to lemuria? [y/n] y
Username: pchapin
Password:
backup-cis-4020-2015-08-06.tar.gz                              100%   38KB  38.3KB/s   00:00
Transfer successful. Removing ~/backup-cis-4020-2015-08-06.tar.gz
student@devbox:~$
```

Figure 1: Sample Backup Session

```
$ rsync -vaz --delete -e "ssh -p 22" \
  cis-4020 username@lemuria.cis.vtc.edu:
```

The `--delete` option tells `rsync` to remove files on the target that are not in the source. The `-z` option specifies compressed mode; this is particularly useful if you are on a low speed network connection since it will reduce the amount of network traffic required. See the `rsync` manual page for more information.

Note that the `rsync` command above uses ssh as the underlying transport. Thus you will be prompted for your password, yet your password will not appear on the network unencrypted. Note also that the `rsync` command above will create (or update) a `cis-4020` directory beneath your home directory on the remote host.

8 Shutting Down

When you are finished using the virtual machines *do not just close the VirtualBox window!* Closing VirtualBox is equivalent to pulling the power on a real machine. Instead you should shut down the guest operating system properly. On DevBox in the upper right corner of the desktop there is a menu with a "Shutdown" option. On HackBox, as the root user, issue the command:

```
# shutdown -h now
```

Alternatively you can suspend the virtual machine. On the VirtualBox menu (not inside the guest) do "Machine → Close..." In the dialog box that appears select "Save the machine state." The next time you start the virtual machine will resume from the saved state. This is usually quicker than booting the system from scratch.

9 Setup Notes

This section contains some notes on how DevBox and HackBox were initially configured. They are intended to be helpful to anyone else who wants to build the systems from scratch rather than downloading the preconfigured systems.

TODO: Finish me!