

How to Create a Custom Live CD for Secure Remote Incident Handling in the Enterprise

Abstract

This paper will document a process to create a custom Live CD for secure remote incident handling on Windows and Linux systems. The process will include how to configure SSH for remote access to the Live CD even when running behind a NAT device. The combination of customization and secure remote access will make this process valuable to incident handlers working in enterprise environments with limited remote IT support.

Table of Contents

Abstract	1
1. Introduction	5
2. Making Your Own Customized Debian GNU/Linux Based System	7
2.1. The Development Environment	7
2.2. Making Your Dream Incident Handling System	9
2.3. Hardening the Base Install.....	11
2.3.1. Managing Root Access with Sudo.....	11
2.4. Randomizing the Handler Password at Boot Time	12
2.5. Installing Incident Handling Tools in Your Development System	14
2.5.1. Rootkit Hunter and Chkrootkit	14
2.5.2. Perl Scripts for Windows Forensic Analysis	14
2.6. Using NX Server/Client for Remote GUI Access over SSH	15
2.7. Installing the X Server and Window Manager	18
3. Using SSH for Secure Remote Access	21
3.1. Creating SSH Keys	21
3.2. Remote Access: The Back End Mothership	22
3.3. SSH Remote Port Forwarding	23
3.3.1. SSH Local Port Forwarding: A Review	23
3.3.2. SSH Remote Port Forwarding.....	23
3.4. Remote GUI Access with NX Server and Port Forwarding.....	25
4. Building the Mothership.....	26
4.1. Hardening the Handler Account on the Mothership	27
4.2. Creating the Handler Account's Home Directory	28
4.3. Using SSH to chroot the Handler Account.....	30
4.4. Setting Up Syslog-ng for Remote Logging Between RIHCD and Mothership	32
5. Getting RIHCD to Phone Home.....	35
5.1. Automating SSH Access at Startup	35
5.2. Automatically Pulling Down New Files from the Mothership at Boot Time.....	37
6. What is Knoppix?	39

- 6.1. Anatomy of a Knoppix CD:..... 39
- 6.2. Overview of the Knoppix Boot Process 43
- 7. **Applying Customized Knoppix Magic to Make Your RIHCD** 45
 - 7.1. Preparing the Mastering Environment..... 45
 - 7.2. Cleaning Up the Base System Before Mastering 46
 - 7.3. Creating the Base System Compressed Loopback File 48
 - 7.4. Editing isolinux.cfg to Customize or Remove Boot Options 49
 - 7.5. Editing the boot.msg file to Customize the Start Up Splash Screen 50
 - 7.6. Unpacking minirt.gz to Edit and Customize the init Start Up Script..... 51
 - 7.7. Editing Initialization Scripts Outside of /boot..... 52
 - 7.7.1. Editing the /etc/inittab File 53
 - 7.7.2. Editing the /etc/init.d/knoppix-autoconfig Script 54
 - 7.7.3. Editing the /etc/init.d/knoppix-startx Script 54
 - 7.8. Creating the Knoppix-Specific Compressed Loopback File 55
- 8. **Putting the CD in RIHCD**..... 56
 - 8.1. Generating SHA1 Sums for CD Integrity Checking..... 57
 - 8.2. Creating the Final .iso Image 57
- 9. **Additional Projects and New Directions for RIHCD** 58
 - 9.1. Additional Software to be Accessed Without Booting from the RIHCD 58
 - 9.1.1. Microsoft Sysinternals 58
 - 9.1.2. Statically Linked Linux Binaries 58
 - 9.2. Putting RIHCD on a USB Thumb Drive 58
 - 9.3. Making the RIHCD an Automated Incident Analysis Disc 59
 - 9.4. Making Modular Packages of Optional Software for RIHCD 59
 - 9.5. Running a Squid Proxy on the Mothership System 59
- 10. **Conclusion** 59
- 11. **References** 61
- 12. **Appendix** 62
 - 12.1. Debian Packages to Include When Building the RIHCD Development System..... 62
 - 12.2. Analysis of Knoppix init Script 64
 - 12.3. Analysis of Knoppix knoppix-autoconfig script..... 66
 - 12.4. etc/inittab Used by RIHCD 68

1. Introduction

Linux based Live CDs are plentiful in various flavors and popular with a wide range of users. New users can boot and run Linux without permanently giving up the familiarity of Windows, while advanced users can boot Live CDs to analyze a compromised system in a safe environment.

Incident handlers and computer forensics experts rely on Live CDs to provide a clean environment in which to run analytical tools. And depending on the task at hand, any number of dozens of existing Live CDs will provide the basic tools needed. However, no off-the-rack solution is going to fit you as well of a tool that is custom tailored to your enterprise's unique operating environment.

Consider the problems faced by an experienced incident handler working in a large, geographically disperse enterprise. When a system is compromised at a remote office, the on-site IT staff might not be up to the challenge of a full post-mortem analysis. They also might not be up to the challenge of booting an existing Live CD distribution and configuring SSH for secure remote access? What if the compromised system is behind a network address translation (NAT) firewall and is not directly accessible?

The solution is to build a Live CD that is designed specifically for incident handling within your enterprise, and the secret ingredient is SSH remote port forwarding through a separate dedicated system. Although this may sound daunting to the uninitiated, all of the building blocks already exist and are familiar to most readers.

Please note that throughout this paper, I will frequently refer to this customized disc as a Live CD, although all of the steps described will work just fine to create a Live DVD. Since fewer systems will boot a DVD than will boot a CD, a Live CD may be preferred. But because of its larger storage capacity, a bootable DVD allows for significantly more latitude when it comes to what software gets included on the disc.

This paper is aimed at incident handlers and systems administrators who are already experienced installing and securing Linux (Debian GNU/Linux, specifically). I also assume that the audience has basic experience with BASH shell scripting, the day-to-

day aspects of using SSH, and hands-on experience installing and using various incident handling tools. In addition, this paper assumes that the incident handler has already established a collection of favorite analytical tools.

Many existing Live CDs try to be all things to all users (or at least most things to most users). They are chock full of applications that will never be used during routine incident handling (e.g. games, office applications, a variety of window managers, etc.). Building your own Live CD means including only what you need and skipping everything else. It also means having the ability to target the disc to a specific purpose and operational environment.

If you have remote IT staff that are able to handle basic incident handling procedures, providing a GUI and desktop will help them help you. If not, perhaps you want to build a CD that provides remote access only. Consider spicing this option up with your own custom shell scripts (or some great ones that others have written) for system analysis, and you can have a fully automated, remotely deployable incident analysis kit.

The basic building blocks for this project are your own custom Debian GNU/Linux install, the Knoppix Live CD, SSH authentication using public keys, and SSH remote port forwarding. The keystone is a dedicated system that the Live CD automatically logs in to at boot time.

The specific application of this Live CD warrants attention beyond the basics if it is to be used securely in an enterprise environment. Consider that once a Live CD is released, it is out of your control and has the potential to be abused by rogue users. Also consider that this CD will provide remote access to any system running it. If user credentials are burned onto the disc, it is possible for an attacker to use these credentials to remotely access any system running the CD. And since the CD will automatically log in to a separate dedicated system, steps must be taken to secure this dedicated system from rogue users.

Sections 2-5 may at first glance look like a run-of-the-mill Debian how-to, but the procedures that are detailed are vital to ensuring that the Live CD can be widely

distributed and not present any additional risk or vulnerability to the systems that are using it, all while providing secure remote access.

If you've ever taken a crack at remastering Knoppix, you already know that there is a slew of online guides and documentation to help you along. This paper is not a traditional remastering how-to. Instead it aims to help you make your own custom Debian install bootable from a CD using Knoppix functionality. In other words, you will learn to *master* Knoppix, not take the existing version of Knoppix and customize it to your tastes (*remastering* Knoppix). According to Kyle Rankin, author of *Knoppix Hacks* “Traditional Knoppix remastering is a time-consuming, error-prone, complex process full of trial and error, but there is a better way.” (Rankin, 2008) Indeed. There are a number of pitfalls to be aware of when remastering Knoppix; by starting from scratch and mastering Knoppix, many of these pitfalls can be avoided.

This process does make extensive use of the unique features found in Knoppix. These include the compressed loopback (or cloop) filesystem, which allows nearly 2 GB of data to be compressed into less than 650 MB and mounted and read just like traditional loopback file; and aufs (Another Union File System), which can mount multiple disparate file systems into one merged directory structure, in this case merging read-only iso images with a read/write ramdisk on top to make the entire file system writable.

2. Making Your Own Customized Debian GNU/Linux Based System

Every incident handler is going to have their own “jump bag” full of tools that they reach for when analyzing a system. Creating your own customized Linux install and making it bootable and remotely accessible aims to make that jump bag a single CD that can be distributed globally and securely accessed from anywhere. In this paper, this CD will be called the remote incident handling CD (RIHCD).

2.1. The Development Environment

Creating your custom Linux install and making it into a bootable CD is going to require a dedicated system (real or virtual) with Internet connectivity for development. If your jump bag full of incident handling tools already includes a modestly sized, highly

customized Debian GNU/Linux install, your existing system can be used for development just as well with the addition of a couple of dedicated disk partitions (e.g., an external hard drive).

The basic layout of hardware needed for this project is a system that has two hard drives and at least one gigabyte of RAM (Neggus, 2007) and can already boot from CD. As with any system development, a faster processor, faster IO, and more RAM will make the disc creation process faster. The minimum specifications that I would recommend would be 2 x 8 GB hard drives, 2 GB of RAM, and at least one processor that is 1 GHz or faster.

A spare workstation with the above specs (or better) can be used for development, or a virtual machine with the above specs (or better) can work just as well, provided the host system still has enough overhead to operate.

My own Live CD development is usually done using a virtual machine in VMware Workstation in Linux. For the demonstrations in this paper, I'm using VMware Fusion on a MacBook Pro. This gives me the ability to reboot the development system multiple times and from multiple boot media even when working remotely (using NX Server and NX Client for remote desktop access). Any virtualization product should offer the same benefits. Using a virtualization product that can make snapshots of a hard drive state eases the ability to fork development of the Live CD into multiple versions for use in multiple environments.

The first hard drive is for installation of the base system, the system that will eventually become a bootable CD. The size requirements for the drive are modest by today's standards since we require only enough space for a relatively basic, although highly customized, Linux install. In my development environment, I've found that 8 GB per drive works well, although you could probably get by with 4 GB for the base install drive.

The base install hard drive should have three partitions:

/boot	512 MB
swap	1 GB or more
/	2 GB or more

An extra partition could be used on this disk or the second disk if optional extra software is to be included (e.g., if multiple versions of the same CD are going to be produced).

The size of the swap partition depends on how much RAM is available in your development system. The total RAM plus swap should be about equal to the size of the disc you wish to create. (Neggus, 2007)

Keeping all of /boot on a separate partition is always good practice, and is good practice here since it will ultimately be replaced with /boot from Knoppix (slightly modified with your customizations, of course). Keeping it on the same partition as the rest of the operating system is technically possible, but can be procedurally cumbersome later in CD development.

The rest of the space is for your customized Debian install. If you're planning on building a CD, you won't need more than 2 GB for this partition since you're aiming for a final compressed loopback file of less than 650 MB. If you're planning on building a DVD, the compressed loopback file can be much larger, so your custom Debian install and this disk partition itself can be larger as well.

The second hard drive is for the CD mastering process. It will accommodate a copy of your base install as well as the resulting compressed loopback (cloop) files and the final bootable CD file; ultimately, it will need to hold more data than the first hard drive that holds the OS only, so plan accordingly. It only needs to have one partition, although an optional second partition will come in handy if you plan on creating multiple versions of the same CD.

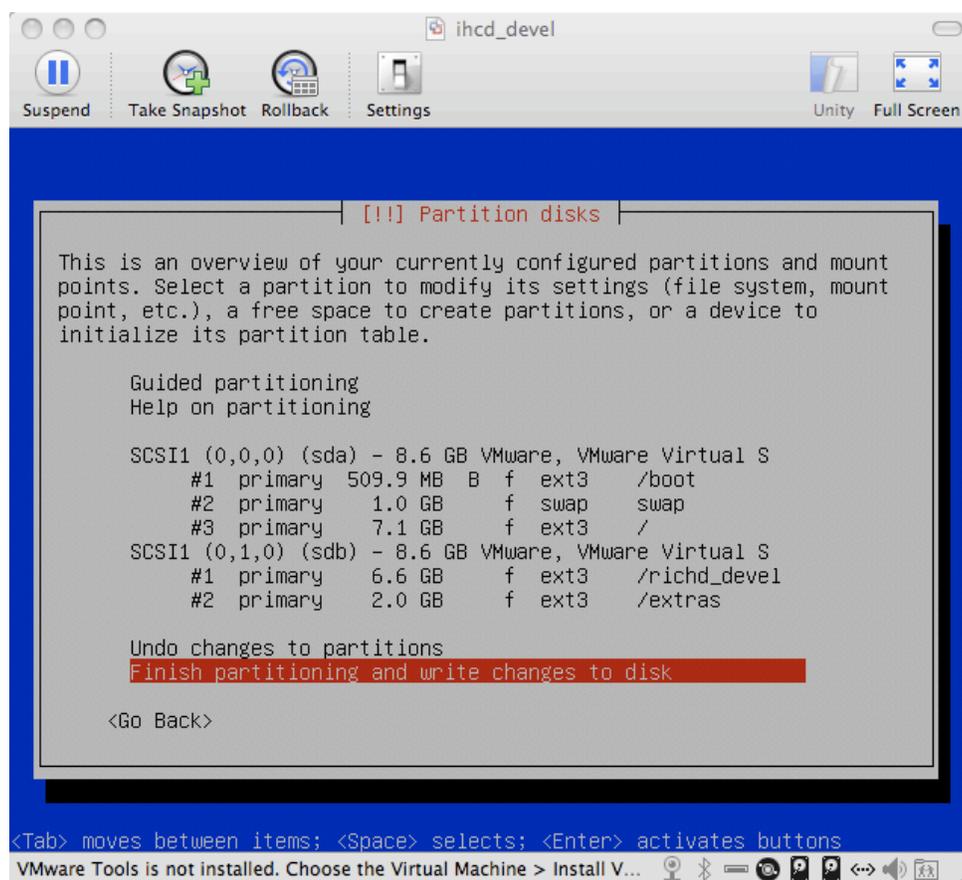
2.2. Making Your Dream Incident Handling System

Since Knoppix is based on Debian GNU/Linux, it makes sense to use this Linux distribution for the base system install. Although the procedures used in this paper may be applicable to other Linux distributions, that kind of hacking is left as an exercise to the reader.

The i386 distributions are preferable since they will run on both x86 and x86_64 hardware. If software packages that are pre-compiled for the x86_64 (or AMD64) were installed, the CD would be not work on older 32 bit hardware.

When partitioning the drive during installation, select “Manual” instead of “Guided” partition. None of the guided partitioning options give you a separate /boot partition, which you’ll want when it comes time to create your loopback images for the CD.

The partitioning scheme that I’ve used is shown in the following figure:



When prompted during installation, create a user account. I call mine “handler.”

When prompted for which kernel to install, you’ll want to select the most recent 686 kernel. At the time of this writing, that’s linux-image-2.6.26-2-686. Eventually, the final CD will be using the 2.6.28.4 kernel that comes with Knoppix (although like everything with this project, the kernel can be customized if desired).

When selecting which initrd to install, you can select the "generic: include all available drivers" option, because ultimately, a customized version of the initrd from Knoppix will be used. Likewise, the Knoppix boot loader (isolinux) will be used to boot the CD, so choosing GRUB vs. LILO as a boot loader makes no difference.

2.3. Hardening the Base Install

Consider the possibility that this CD is going to be widely distributed, regardless of whether you want it to be or not. Once you have the first remote user download and burn a copy, the number of subsequent copies made and how they are used is out of your control. Imagine that there are a thousand copies of this CD floating around your enterprise as well as the outside world.

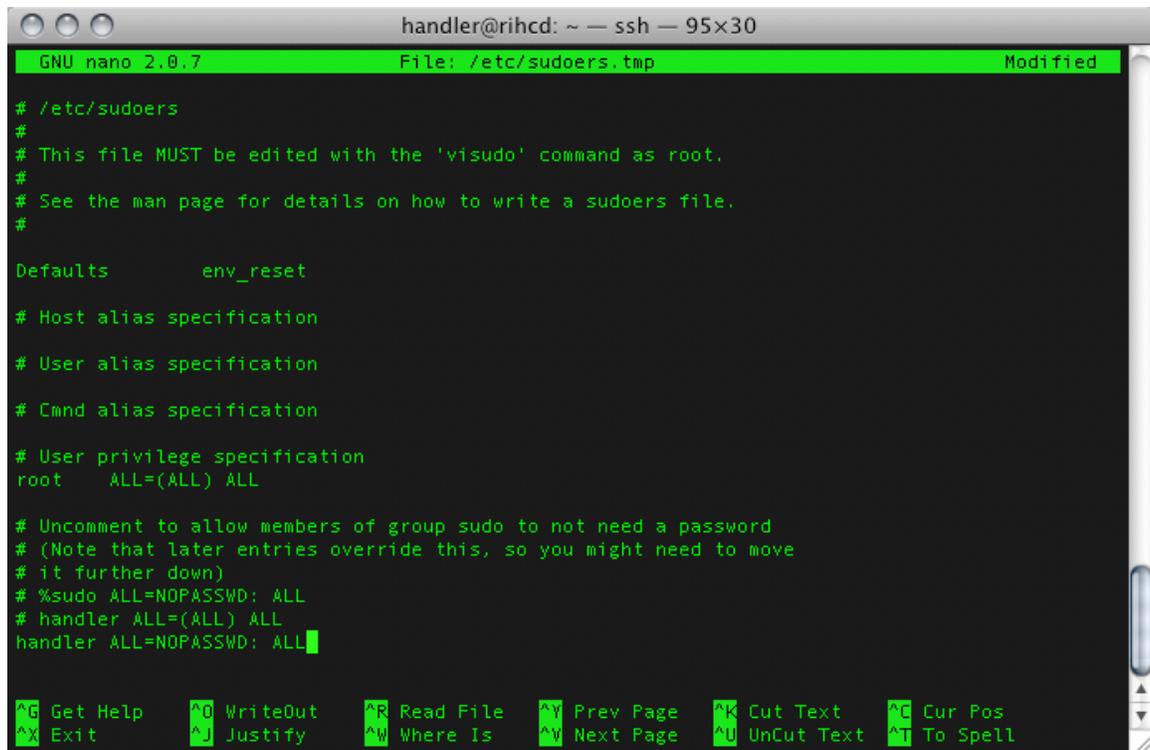
Although this scenario sounds far-fetched, it must be taken seriously and considered at every point in the design process, because once you release this disc, you are setting it free with the potential to be used and abused by anyone who picks it up. This is a point that we will return to time and time again when designing and building this CD.

2.3.1. Managing Root Access with Sudo

If you haven't already done so, make sure that root logins are not allowed by removing the hashed password from root's `/etc/shadow` entry and replacing it with a "!":

```
root:!:14487:0:99999:7:::
```

We're also going to ensure that the user account created during installation (e.g. "handler") has sudo access. Use the `visudo` command to edit the `/etc/sudoers` file:



```
handler@rihcd: ~ -- ssh -- 95x30
GNU nano 2.0.7 File: /etc/sudoers.tmp Modified
# /etc/sudoers
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL) ALL

# Uncomment to allow members of group sudo to not need a password
# (Note that later entries override this, so you might need to move
# it further down)
# %sudo ALL=NOPASSWD: ALL
# handler ALL=(ALL) ALL
handler ALL=NOPASSWD: ALL

^G Get Help      ^D WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify      ^W Where Is     ^N Next Page    ^U UnCut Text   ^T To Spell
```

In this example, the handler account is allowed to run sudo without being prompted for a password. This design copies that of most Live CDs, including Knoppix, and is tremendously helpful for running automated scripts, or allowing end-users to operate more easily. It may not be ideal for every operational environment.

Naturally, allowing the only user account unchecked root access seems like a security risk, especially since the CD will ultimately be remotely accessible via SSH and password authentication. Anyone who knows the handler password could remotely access a system running RIHCD and have their way with it.

To prevent this kind of abuse, the handler account password will be reset to a random value at boot time. The password can also be aged so that it has already expired; when the handler account logs in, the user must immediately change this password. (Schroder, 2009)

2.4. Randomizing the Handler Password at Boot Time

Take a look at the `mass_passwd` script available from http://tuxcomputing.com/cookbook/mass_passwd. This script will create random passwords for any number of users. In our case, we only want to generate a password for

one user, the handler account. Grab a copy of the script and put it into `/usr/local/sbin` or somewhere else appropriate. Make sure it's executable and call it from `/etc/rc.local` where it will be run at boot time. Also, don't forget to show the contents of the text files created by this script if you want people to be able to log in using the handler account's password. Writing this output to `/etc/motd` will ensure that the handler account can read this information, which will be necessary when this user is prompted to change their password.

```
$ wget http://tuxcomputing.com/cookbook/mass_passwd
$ sudo mv mass_passwd /usr/local/sbin/mass_passwd.sh
$ sudo chmod 755 /usr/local/sbin/mass_passwd.sh
$ sudo vim /etc/rc.local
```

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

/usr/local/sbin/mass_passwd.sh handler
cat /mass_passwds/handler.passwd.txt
cat /mass_passwds/handler.passwd.txt > /etc/motd

exit 0
```

You'll probably want to edit the `mass_passwd.sh` file itself to change the default message that gets printed out to `/mass_passwd/username.passwd.txt` (You might even want to change the location of this file.)

Double-check your work before you reboot. If you've made a typo or otherwise botched the output of the password, you'll find yourself running a system that has only one console-access account and an unknown, random password for that account. To fix it, boot from a Knoppix disc, mount the hard drive, edit your script, umount the drive and reboot.

Depending on how you plan to distribute your CD, you may wish to undertake additional system hardening. If you plan to release a bootable system with no local console access, harden away. If you plan to release a user-friendly CD to remote IT staff, you might not want to harden the OS too much, as it could be unwieldy or unusable by less experienced IT staff.

2.5. Installing Incident Handling Tools in Your Development System

Every experienced incident handler is going to have his or her own set of favorite tools for system analysis. Appendix A contains a list of software that can be installed as Debian packages. The list is not meant to be an exhaustive catalog of recommended tools, but rather a starting point of basic tools that should be applicable in most situations. Since you'll be building a highly customized system, take this list of tools and run with it.

In addition to incident handling software that can be installed as Debian packages, the following utilities are stand outs.

2.5.1. Rootkit Hunter and Chkrootkit

Rootkit Hunter, from <http://www.rootkit.nl>, and chkrootkit from <http://www.chkrootkit.org> both work as advertised and will find rootkits. Since Debian packages for these two pieces of software usually lag behind the latest release from the project's web sites, it's beneficial to install these from source so that the freshest versions are used.

2.5.2. Perl Scripts for Windows Forensic Analysis

Since Perl is available for Windows as well as Unix, a number of Perl scripts have been written for Windows forensic analysis that can easily be ported to Unix. For examples of these scripts, Harlan Carvey's, *Windows Forensics and Incident Recovery*

and *Perl Scripting for Windows Security: Live Response, Forensic Analysis, and Monitoring*.

Harlan's RegRipper tool will parse through Windows registry files looking for forensically significant registry entries and the dates on which they were modified. This is tremendously useful in post-mortem analysis as it can help pinpoint when a system was compromised. His evt2xls script will take Windows event log files and spit them out as .xls files to be opened and analyzed with Microsoft Excel. Once in Excel, the data can be sliced up according to the analyst's needs, zeroing in on important information while excluding the rest.

2.6. Using NX Server/Client for Remote GUI Access over SSH

If you are designing your CD to provide no local console access, and instead provide only remote SSH access, it's not necessary to install a window manager to use GUI applications. Using the built-in X forwarding over SSH is one way to secure a remote GUI, but unless every remote node in your enterprise network is bathing in bandwidth and spare CPU cycles, the experience can be painful. Instead, install NX server to send highly optimized, compressed X traffic over SSH, all with much lighter system resource consumption.

NX server for Linux is provided for free by NoMachine (www.nomachine.com). Also freely available are NX clients for Windows, Mac OS X, Linux, Solaris, and a web-based client.

For this install, grab the NX Free Edition for Linux at <http://www.nomachine.com/download.php>. You'll need to download and install three separate components: NX Node, NX Client, and NX Server (Medialogic S.p.A., 2009).

Download and install each of the following packages. Check the links from the download page noted above; links may change between the time this was written and the time you read it.

```
$ cd /home/handler
$ wget http://64.34.161.181/download/3.3.0/Linux/nxclient_3.3.0-6_i386.deb
$ wget http://64.34.161.181/download/3.3.0/Linux/nxnode_3.3.0-22_i386.deb
```

```
$ wget http://64.34.161.181/download/3.3.0/Linux/FE/nxserver_3.3.0-27_i386.deb
$ sudo dpkg -i nxclient_3.3.0-6_i386.deb
$ sudo dpkg -i nxnode_3.3.0-22_i386.deb
$ sudo dpkg -i nxserver_3.3.0-27_i386.deb
```

NX installs itself into the relatively non-intuitive directory `/usr/NX/` where you'll find config files, logs, binaries, etc. After the install, look for an install log file in `/usr/NX/var/log/install`

Also note that if you've changed the default port that your SSH daemon listens on (something I always recommend), NX server won't start as it looks for SSH on port 22. To fix that, edit `/usr/NX/etc/node.cfg` and `/usr/NX/etc/server.cfg` to reflect your SSH port:

```
SSHDPort = "2222"
```

Also, edit the following lines in `/usr/NX/etc/server.cfg` accordingly:

```
SSHDAuthPort = "2222"
EnableUnencryptedSession = "0"
```

NX uses SSH for the transport mechanism as well as authentication (provided that SSH is using password authentication and PAM), so NX needs to know where SSH is running. Disabling unencrypted sessions is a good idea; otherwise, only the session negotiation is encrypted, with all other GUI traffic unencrypted (Medialogic S.p.A., 2009).

Since we're still allowing password authentication and PAM in our SSH server, these should be the only changes you'll need to make to NX. Additional changes are required if you are not allowing password authentication to SSH, and instead rely solely on public keys. This is a great way to go for SSH security because it stops brute force attacks cold, but NX can no longer use SSH's authentication mechanisms and must be configured to use its own. (Medialogic S.p.A., 2009)

After making these configuration changes, go ahead and restart the NX server:

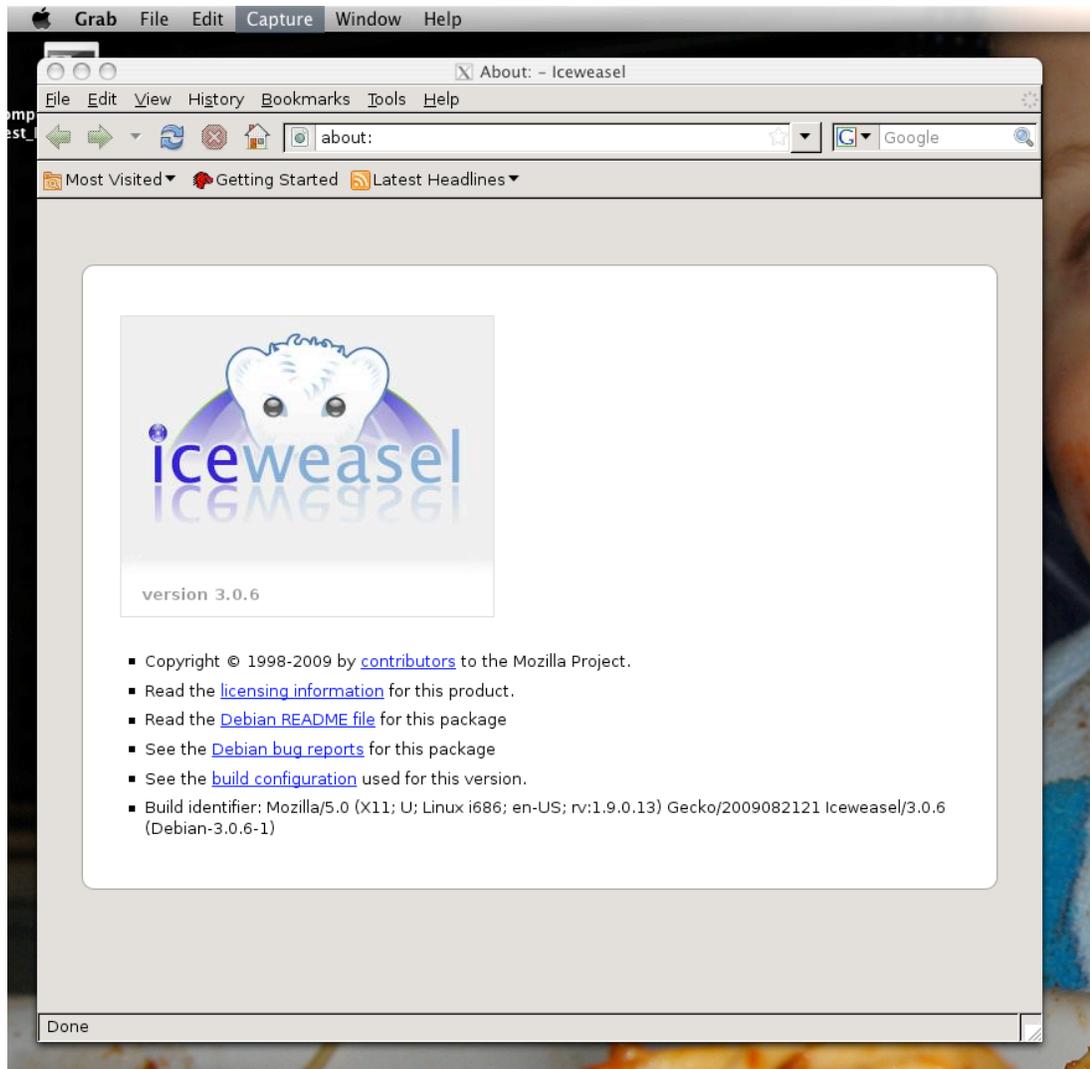
```
$ sudo /etc/init.d/nxserver restart
```

Grab a copy of `nxclient` for your favorite workstation, and try to connect to your development system. Use your "handler" account and its current password. If you have trouble connecting, check the `/var/log/messages` file on your dev system. If you want

gobs and gobs of debugging information, set the log level to 7 in `/usr/NX/etc/server.cfg`, but don't forget to set it back to 6 when you've solved your problem.

Once you've verified that your NX server install is working correctly, go ahead and remove the `.deb` files you downloaded to `/home/handler`.

Here's a screenshot of my development system running Iceweasel (Debian's non-Mozilla branded version of Firefox):



So now you've got a system that you can log into remotely and run GUI applications on your remote desktop.

Now is a good time to configure the web browser in your development system. Get it just the way you want it before it gets immutably burned to CD. This could

include installing plugins (I heartily recommend Adblock Plus and NoScript at least) and configuring bookmarks. If you've never seen [virustotal.com](http://www.virustotal.com) stop reading and check it out. Go ahead and add this bookmark now, because you'll surely want it later.

Since some newer versions of Firefox (and by extension, Iceweasel) have a rather uniquely frustrating way of handling sites with invalid SSL certificates, you'll probably want to add permanent exceptions for any self-signed certificates on servers that you may regularly use. Add them now, or face the prospect of re-adding them every time you visit these sites when booted from the CD.

So now you've got your favorite incident handling tools installed, and your web browser is configured just the way you like it. If you're building this system in a virtual machine, now is a good time to take a snapshot of the hard drive to save your work.

If your CD is going to prohibit local console access and require all users to log in via SSH, your system should be pretty much done, and you can skip the next sections regarding installing the X server and window manager. If you do want to provide a GUI and desktop to users booting the CD, read on.

2.7. Installing the X Server and Window Manager

If you're planning on making the CD usable by remote staff, you'll want a GUI desktop. And since you're trying to be mindful of space constraints, you'll want to make sure you pick something relatively lightweight.

This is where the benefits of customization are realized. By eschewing the more bloated offerings like KDE and Gnome you can save space on the CD. The fluxbox window manager is lightweight and fast and will run responsively on older, slower systems. Because it doesn't come with many extra applications built in, it will consume little disk space when you install it.

The minimal interface in fluxbox can be confusing for new users who are accustomed to seeing icons, toolbars, and especially a “Start” button. If you know you'll be building a CD for remote IT staff to use locally, and if they fall into this user category, you might consider installing Xfce (<http://www.xfce.org>). “Xfce is a lightweight desktop

environment for various *NIX systems. Designed for productivity, it loads and executes applications fast, while conserving system resources.” (Fourdan, 2009)

This paper will use fluxbox as an example, but any window manager with a small installation footprint will work just as well.

To install fluxbox:

```
$ sudo apt-get update
$ sudo apt-get install fbdesk fbpager fluxbox fluxconf
```

The additional packages extend the functionality of fluxbox. Fluxbox offers a GUI interface to configuring fluxbox; fbpager provides a way of keeping track of which virtual desktops have which windows running in them; and fbdesk allows you to create icons on your desktop. Only the fluxbox package itself is required to run the window manager, but the optional extras can be nice to have, especially fluxconf.

Now that you've got a window manager installed, you'll want to have an X server to run it on. Since you have NX Server installed and running, you can run Fluxbox remotely over NX! Just select the "New virtual desktop" and "Run the following command: /usr/bin/startflux" settings and fire it up.

To use your window manager locally, you're going to need an X server. The X.org X server comes in many different flavors for many different video cards. Since you want your CD to have drivers for a multitude of different video cards, you'll want to install as many X server video packages as possible. To get a taste of just how many packages are available, run the following command:

```
$ sudo apt-cache search xserver-xorg
```

Thankfully, all of these video card and input drivers can be installed by installing two meta-packages:

```
$ sudo apt-get install xserver-xorg-input-all xserver-xorg-video-all
```

Go ahead and let Debian install all of its recommended packages on this one. Some additional packages for 3D graphics and pretty fonts are recommended, and it's a good idea to install them too:

```
$ sudo apt-get install libglide3 xfonts-100dpi xfonts-75dpi xfonts-scalable
```

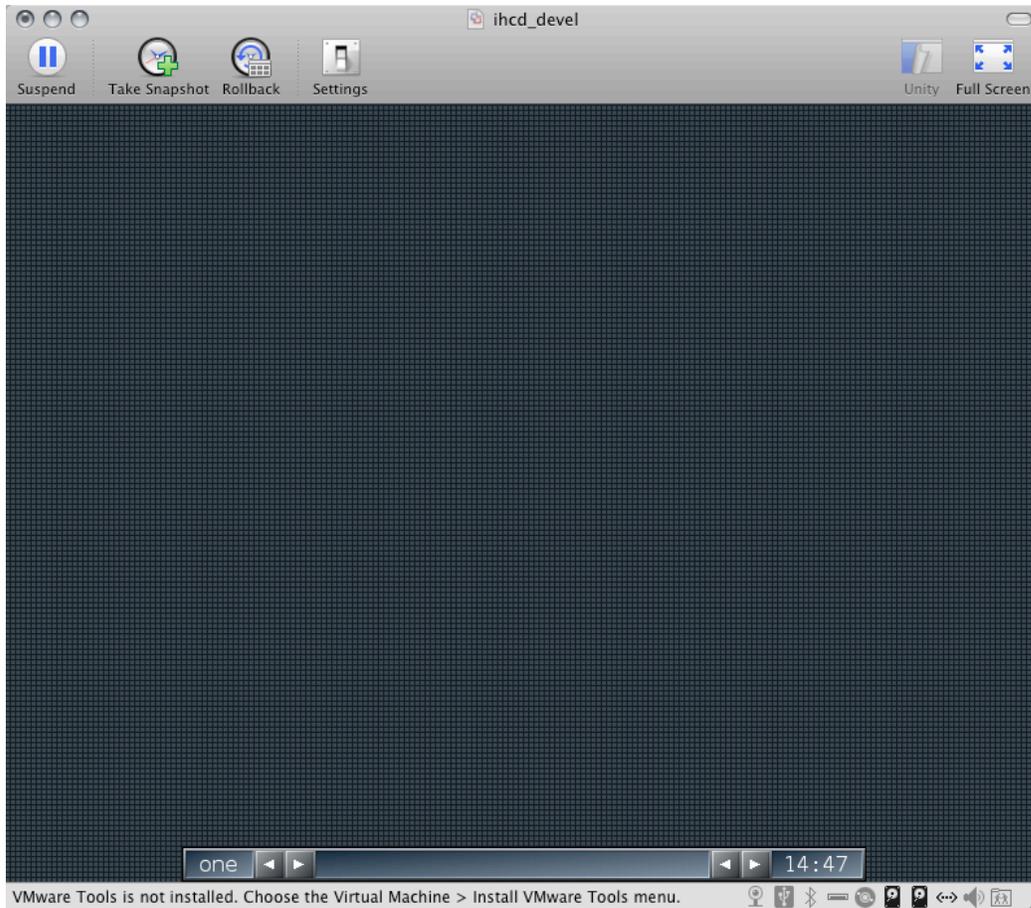
You'll also want to install the xinit package so that you can actually start the X server up.

```
$ sudo apt-get install xinit
```

Now's a good time to check your progress:

```
$ startx
```

Did you see something like this?



If not, check the `/var/log/Xorg.0.log` file for errors.

If you're used to setting up a display manager like GDM or XDM as a way to manage logging in and starting the desktop, don't worry about it here. You're only going to have one account on the disc that can log in to the local console, and the Knoppix startup script `/etc/init.d/knoppix-startx` handles starting up the GUI.

3. Using SSH for Secure Remote Access

Using SSH as a transport for the NX remote desktop software has already been demonstrated. This section will outline how to configure SSH for automated remote access using public/private key pairs and remote port forwarding.

3.1. Creating SSH Keys

If you haven't done so already, now's a good time to create a public/private key pair for the handler account on your RIHCD.

It's imperative to create the private key *without* a passphrase or password. When using a passphrase to protect your private key, the passphrase must be entered before the system can access your key. Naturally, this will hinder the automated access we'll need for port forwarding, since we don't want the system to wait for a remote user to type in a passphrase.

```
handler@rihcd:~$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/handler/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/handler/.ssh/id_dsa.
Your public key has been saved in /home/handler/.ssh/id_dsa.pub.
The key fingerprint is:
39:1c:59:66:52:53:c5:06:a3:e2:54:1b:bd:d7:90:28 handler@rihcd
The key's randomart image is:
+--[ DSA 1024]-----+
|      ..Oo+=.. |
|      BE=o.= |
|      = o. o o |
|      + + . . .|
|      S . |
|      . |
|      |
|      |
|      |
```

```
+-----+
```

```
handler@rihcd:~$
```

You should now have two new files: `/home/handler/.ssh/id_dsa` and `/home/handler/.ssh/id_dsa.pub`

3.2. Remote Access: The Back End Mothership

So now you've got your own customized Debian install with remote command line access via SSH and remote GUI access via NX over SSH. Now to make the system accessible even when behind a network address translation (NAT) device.

NAT can act as a kind of one-way filter for networks. Systems behind a NAT firewall can initiate traffic out to the Internet, and the associated responses can get back in through the firewall. But since the network “behind” the NAT device typically uses private network space (RFC 1918), any traffic that is not already associated with an existing outbound connection cannot get in to the private hosts.

NAT firewalls are ubiquitous and frequently found in homes and small offices as a Small Office Home Office (SOHO) router, firewall, or combination of these with a Wireless Access Point (WAP). They are also common in remote offices where far-flung staff make do with whatever Internet access is available. Any system behind a NAT device will not be directly accessible via SSH unless the NAT firewall is configured to specifically forward this traffic. So how to access the CD when booted behind a NAT device?

Instead of initiating the SSH connection from the outside -> in (Internet -> private NAT network) have the host behind the NAT firewall initiate the connection out (private NAT space -> Internet).

This is similar to how much malicious code operates now: gone are the days when a virus would open up a listening port on the infected host to act as backdoor access for the attacker. Virus writers know that NAT firewalls are everywhere, prohibiting this kind of access, so they design their malicious code to initiate the connection out to a dedicated system, often an Internet Relay Chat (IRC) server or a web server.

We're going to tell our remote incident handling system to phone home using SSH to a dedicated server at start up. This dedicated server can be in publicly addressable IP space, or RFC 1918 space if your enterprise network routes it internally. The SSH connection will not provide console access, but instead will utilize remote port forwarding to provide a back channel to the system running our CD.

3.3. SSH Remote Port Forwarding

3.3.1. SSH Local Port Forwarding: A Review

Many readers are familiar with SSH local port forwarding. This is where a user has takes all TCP traffic bound for a local listening port and sends it to a listening port on a remote machine, all over SSH.

For example, to check your email on an IMAP mail server without exposing your login credentials or the email itself, you could run the following command:

```
$ ssh -L1430:localhost:143 you@your.imap.email.server
```

This opens port 1430 on your localhost interface and sends all traffic sent to your system's 127.0.0.1:1430 over SSH to 127.0.0.1:143 of the remote IMAP mail server. So configure your email client to use 127.0.0.1:1430 as its IMAP server and enjoy checking your email using a plain text protocol tunneled through an encrypted channel, keeping your login credentials a secret.

SSH can also do remote port forwarding, and this is the trick we use to get access to our hosts on a NAT network.

3.3.2. SSH Remote Port Forwarding

In remote port forwarding, a user logs in to a remote machine and opens up a listening TCP port on its localhost interface. Anything that is sent to this port is shoveled back over SSH to the designated port listening on the loopback interface of the local machine, where the connection originated.

Using remote port forwarding, we configure the incident handling system to automatically SSH to a designated remote host (hereafter called the Mothership), and use remote port forwarding to send all traffic from a listening port on the Mothership's localhost to the SSH server. Provided that the remote incident handler has an account on

the Mothership, he can then use SSH to log in to the Mothership, then SSH to the listening port on Mothership's localhost interface, which is then forwarded back to the SSH server on the incident handling CD.

Consider the following (annotated and slightly edited for readability) terminal session, which starts on my laptop (silverslab) on a private network behind a NAT firewall in a neighborhood coffee shop.

```
silverslab:~ bert$ ifconfig en1 | grep "inet "  
    inet 192.168.10.15 netmask 0xfffff00 broadcast 192.168.10.255
```

Log in to the virtual machine (also behind in a NAT network):

```
silverslab:~ bert$ ssh -p 2222 handler@192.168.148.129  
handler@192.168.148.129's password:  
Linux rihcd 2.6.26-2-686 #1 SMP Fri Aug 14 01:27:18 UTC 2009 i686  
Last login: Sat Sep  5 13:47:44 2009 from localhost  
handler@rihcd:~$
```

I'm in. Now to use remote port forwarding and log in to my publicly addressable workstation.

```
handler@rihcd:~$ ssh -R3333:localhost:2222  
bert@suckerfish.infosec.utexas.edu  
bert@suckerfish.infosec.utexas.edu's password:  
Linux suckerfish 2.6.26.3suckerfish #1 SMP Mon Jul 27 16:56:53 CDT 2009  
x86_64  
Last login: Sat Sep  5 13:46:47 2009 from rrcs-71-42-142-  
91.sw.biz.rr.com  
bert@suckerfish:~$
```

Now I'm logged in to my workstation. From here I use SSH to log in to the listening port on my loopback interface that was designated above:

```
bert@suckerfish:~$ ssh -p 3333 handler@localhost  
handler@localhost's password:  
Last login: Sat Sep  5 13:48:18 2009 from 192.168.148.1  
handler@rihcd:~$
```

Now I'm logged in to my virtual machine, behind two different NAT devices.

```
handler@rihcd:~$ /sbin/ifconfig eth0 | grep "inet addr"  
    inet addr:192.168.148.129 Bcast:192.168.148.255  
Mask:255.255.255.0  
handler@rihcd:~$
```

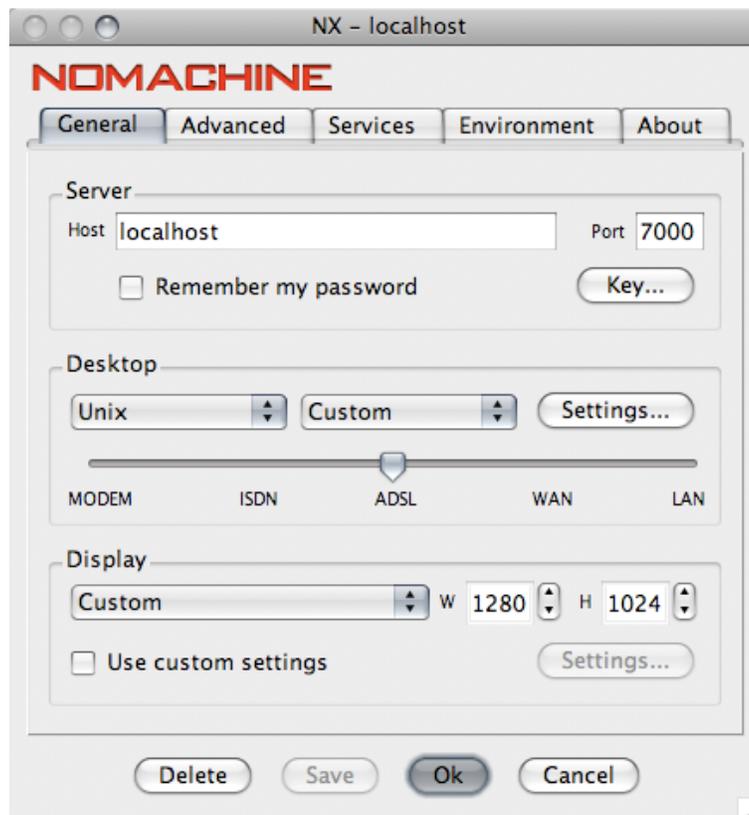
3.4. Remote GUI Access with NX Server and Port Forwarding

Now, assuming that the remote port forwarding session from suckerfish to the RIHCD is still active (anything sent to suckerfish:3333 will be forwarded over to RIHCD:2222), consider this SSH *local* port forwarding command:

```
silverslab:~ bert$ ssh -L7000:localhost:3333 bert@suckerfish
Linux suckerfish 2.6.26.3suckerfish #1 SMP Mon Jul 27 16:56:53 CDT 2009
x86_64
Last login: Sat Sep  5 14:16:26 2009 from rrcs-71-42-142-
91.sw.biz.rr.com
bert@suckerfish:~$
```

Any TCP traffic that we send to port 7000 on silverslab's localhost will be shoveled over SSH to suckerfish, which will in turn shovel it back to the SSH daemon on RIHCD.

Configure your NX client to point to localhost:7000



Now you can run GUI applications from RIHCD, even while it's in private space on a NAT network.

4. Building the Mothership

The Mothership is a system dedicated to the sole purpose of receiving remote port forwarding SSH connections from your remote incident handling CD. (In the examples above, I used suckerfish as the Mothership system.) It must be accessible by systems that boot RIHCD for remote access, as well as your own workstation.

It also serves as a kind of home base or back-end server for the booted CD. It will serve as a centralized syslog server for the disc as well as a file repository for new and updated scripts. In addition to starting a remote port forwarding SSH session at boot time, the CD can also be configured to automatically pull down new and updated files from the Mothership via rsync. This means that many new features can be implemented without creating an entirely new CD to distribute.

Although the Mothership is the keystone to SSH access to NAT systems running RIHCD, the Mothership system itself can be relatively simple and stripped down. (This system would be a good candidate for virtualization.) At a minimum, the only thing it needs to do is run SSH and allow the handler account to log in. If you don't already have a centralized logging server on your network, you can run one on the Mothership using syslog-ng and have all of your RIHCD logs stored/monitored in one location.

Running this service is best done on a dedicated system that can be locked down and closely monitored. Remember that you're not going to have much control over who boots your new CD, and it's going to be configured to automatically log in to the Mothership at boot time (although this will be locked down later with SSH chroot).

Any flavor of Unix should work for the Mothership. To keep things consistent, I'm going to stick with the same version of Debian that I used to build the base install for the CD, using a net install CD image and a minimal, expert install. If you're going to run centralized syslogging on this system, plan your disk partitions accordingly.

I won't cover actually installing this OS, but with any new install, system-hardening steps should be taken, such as turning off unnecessary services, ensuring that there are no extraneous user accounts on the system, and optionally installing and configuring IPTables to your network's requirements and your tastes. Make sure you leave the SSH port open to networks that will be booting your final CD.

If you haven't changed the default port that SSH runs on (22), please consider doing so if you want to avoid the overwhelming majority of most SSH attacks. It won't stop a dedicated attacker, but it will stop ankle-biters and automated attacks (e.g. worms).

4.1. Hardening the Handler Account on the Mothership

Anyone who can boot your CD will have the ability to log into the Mothership system with the handler account, so substantial system hardening should be done to protect against abuse from this account.

In addition to changing the default listening port for SSH you will want to configure the SSH server to not accept passwords, and instead rely on public key authentication.

We don't want *just anyone* to be able to log in to Mothership, just *anyone with a copy of the CD*. So disabling password authentication for the handler account means that an end user must at one point have a copy of the handler account's private SSH key. Sharing a known password (easier than sharing the SSH private key) won't get a rogue user access to Mothership.

The following are changes from the default values in `sshd_config` to disable password authentication (BSD, 2009):

```
# Change to no to disable tunnelled clear text passwords
PasswordAuthentication no
....

UsePAM no
```

Create the handler account on Mothership, if you haven't already done so. (It will be very helpful in the future if you ensure that the handler account on Mothership has the same numerical UID as the handler account on your base install.)

Now edit the `/etc/passwd` file and change the default shell for the handler account from `/bin/bash` to `/bin/rbash`, the restricted shell.

`/bin/rbash` is the same as calling `/bin/bash` with the `-r` switch to restrict what the user of this shell can do. `rbash` behaves identically to `bash` with the exception that the following are disallowed or not performed:

- Changing directories with `cd`
- Setting or unsetting the values of `SHELL`, `PATH`, `ENV`, or `BASH_ENV`
- Specifying command names containing `/`
- Specifying a file name containing a `/` as an argument to the `.` builtin command
- Specifying a filename containing a slash as an argument to the `-p` option to the `hash` builtin command
- Importing function definitions from the shell environment at startup
- Parsing the value of `SHELLOPTS` from the shell environment at startup
- Redirecting output using the `>`, `>|`, `<>`, `>&`, `&>`, and `>>` redirection operators
- Using the `exec` builtin command to replace the shell with another command
- Adding or deleting builtin commands with the `-f` and `-d` options to the `enable` builtin command
- Using the `enable` builtin command to enable disabled shell builtins
- specifying the `-p` option to the `command` builtin command
- turning off restricted mode with `set +r` or `set +o restricted`.

These restrictions are enforced after any startup files are read. (GNU, 2004)

4.2. Creating the Handler Account's Home Directory

Since you'll have no control over who uses the handler account to log in to the Mothership, you must severely restrict what this account can do. The first step was

specifying the rbash shell for this account, the next step is limiting what the handler account can do via SSH, its only real shell access.

```
# mkdir -p /home/handler/.ssh
```

```
# vi /home/handler/.ssh/authorized_keys2
```

Add the contents of the `/home/handler/.ssh/id_dsa.pub` file from your RIHCD base install to this file and save it. Then recursively change ownership of all files in `/home/handler` so that handler owns them, and change permissions on `.ssh/` and `.ssh/authorized_keys2`:

```
# chown -R handler:handler /home/handler
```

```
# chmod 700 /home/handler/.ssh
```

```
# chmod 600 /home/handler/.ssh/authorized_keys2
```

You should now be able to SSH without being prompted for a password from your RIHCD base system to the Mothership using the handler account. Recall that automatic SSH access will be critical for setting up port forwarding and using rsync on RIHCD to pull down updated files from the Mothership.

Did your SSH access work? Did you get a shell on Mothership? If so, that's great for testing purposes, but terrible in real life. Since we're basically allowing anyone to log in with the handler account, we don't want them to get an interactive shell on a virtual terminal. Denying a terminal is accomplished with the `no-pty` option in the `authorized_keys2` file.

Open the `/home/handler/.ssh/authorized_keys2` file and append `no-pty` to the front of the key:

```
no-pty ssh-dss AAAAB3NzaC1kc.....
```

Although this won't stop the handler account from running non-interactive commands over SSH:

```
handler@rihcd:~$ ssh -p 2200 192.168.148.131 w
17:29:48 up 45 min, 1 user, load average: 0.00, 0.00, 0.00
USER      TTY      FROM          LOGIN@      IDLE        JCPU       PCPU  WHAT
root     tty1    -             16:50      16.00s     2.16s     1.96s  -bash
handler@rihcd:~$
```

4.3. Using SSH to chroot the Handler Account

The best way to prevent unwanted commands from being run remotely is to chroot the handler account using SSH. This has the advantages severely limiting the number of executable commands that can be run as well as making the rest of the file system inaccessible to the chrooted user. (BSD, 2009)

If you've ever made a chroot jail before, you might not be looking forward to the prospect of doing it again. Creating device nodes, finding and copying library files, and replicating directory structures is all such a time-consuming mess that's easy to get wrong most of the time. Thankfully, there's the `makejail` utility (<http://www.floc.net/makejail/>), and it's available as a Debian package.

```
$ sudo apt-get install makejail
```

“The objective of `makejail` is to help an administrator creating and updating a chroot jail with short configuration files.” (Tessio, 2009) It uses `strace`, a system call tracer, to determine what libraries, files, and directories a specific binary application needs to run. It then populates a chroot directory with these files.

To run `makejail` to create a chroot environment for your handler account, you'll need a small `makejail` config file. Mine is named `chroot.py` and looks like this:

```
chroot="/chroot"  
cleanJailFirst=1  
testCommandsInsideJail=["rbash rsync"]
```

Make sure you've got a `/chroot` directory before running `makejail`:

```
$ sudo mkdir /chroot  
$ sudo makejail chroot.py
```

You may need to add a link in `/chroot/bin` for `sh`:

```
ln -s /chroot/bin/rbash /chroot/bin/sh
```

Now edit the `/etc/ssh/sshd_config` and use a `Match` user directive to specify that only the handler account is chrooted when logging in via SSH (BSD, 2009):

```
Match user handler  
    ChrootDirectory /chroot  
Match
```

Now to test SSH access for the handler account from the RIHCD base system to the Mothership:

Trying to SSH and get a shell results in a hung terminal:

```
handler@rihcd:~$ ssh -p 2200 192.168.148.131
PTY allocation request failed on channel 0
```

Trying to use SSH to get a directory listing fails, since ls is not in a chroot environment:

```
handler@rihcd:~$ ssh -p 2200 192.168.148.131 "ls -al /"
rbash: ls: command not found
handler@rihcd:~$
```

But since rsync is in the chroot directory, the handler account can still use rsync over SSH to download files from the Mothership:

```
handler@rihcd:~$ rsync -rv -e "ssh -p 2200"
192.168.148.131:/home/handler/* .
receiving incremental file list
test_rsync_file

sent 30 bytes  received 104 bytes  268.00 bytes/sec
total size is 18  speedup is 0.13
handler@rihcd:~$
```

And remote port forwarding using SSH still works:

```
handler@rihcd:~$ ssh -N -n -p 2200 -R3333:localhost:2222
192.168.148.131 &
[1] 15431
handler@rihcd:~$
```

```
mothership:~# ssh -p 3333 handler@localhost
handler@localhosthost's password:
Linux rihcd 2.6.26-2-686 #1 SMP Fri Aug 14 01:27:18 UTC 2009 i686
Last login: Tue Sep  8 15:14:46 2009 from 192.168.148.1
handler@rihcd:~$
```

If you put a directory that is write-only and owned by handler in your /chroot directory, you can have a landing pad to use for uploading files from your RIHCD. This

is great for uploading the output of analytical scripts, or any other interesting files for additional analysis or archiving.

```
mothership:/chroot# mkdir handler_files
mothership:/chroot# chown handler:handler handler_files
```

From your RIHCD base system:

```
handler@rihcd:~$ rsync -v -e "ssh -p 2200" junx.tgz \
192.168.148.131:/handler_files
junx.tgz
```

```
sent 4756 bytes  received 7063 bytes  23638.00 bytes/sec
total size is 1368156  speedup is 115.76
handler@rihcd:~$
```

If this functionality is added, remember that it can be abused by a rogue user to fill up the filesystem on the mothership, or to download potentially sensitive files that previous users have uploaded. To avoid this kind of abuse, ensure that any files that are uploaded are quickly moved to a different directory that is inaccessible by the handler account.

The chroot functionality introduced in later versions of OpenSSH is a welcome feature, allowing us to severely restrict what the handler account can and can't do. Although it was originally designed to simplify securing SFTP user accounts, using the makejail program makes setting up a chroot shell account less painful than traditional methods by orders of magnitude.

A nice side-effect of using SSH chroot is that the handler account effectively has no access to its own /home directory on the Mothership system. This greatly limits the potential for malfeasance from rogue users (e.g. rogue user can't upload or edit ~/analysta/.ssh/authorized_keys)

4.4. Setting Up Syslog-ng for Remote Logging Between RIHCD and Mothership

Syslog-ng is the “next generation” of syslog. It features remote logging over TCP (regular old syslog will only forward logs over UDP which is less reliable), flexible filtering of log entries based on content, and the ability to use macros to designate log destinations (BalaBit, 2009).

At the time of this writing, the latest version of syslog-ng available for the stable branch of Debian is 2.0.9. The latest version of the sourcecode is 3.0.4 available from <http://www.balabit.com/downloads/files/syslog-ng/open-source-edition/>

The 3.0 branch supports remote logging over TLS (Transport Layer Security) natively, while the 2.0 branch needs `stunnel` (<http://www.stunnel.org/>) to send and receive logs over SSL.

The following example will describe how to set up syslog-ng for remote logging over an un-encrypted channel. Decide for yourself if your network environment requires you to send syslog data over an encrypted channel. It probably does, but that is beyond the scope of this paper. (Either method will require the creation of encryption certificates using OpenSSL and most likely the creation of your own certificate authority.) Instead, I point you to the very excellent documentation for setting up TLS in syslog-ng 3.0 (<http://www.balabit.com/dl/html/syslog-ng-v3.0-guide-admin-en.html/bk01-toc.html>) or the very excellent recipes to do this in chapter 19 of the Linux Network Cookbook, by Carla Schroder.

You should have the latest Debian package of syslog-ng already installed on both the RIHCD base install system and the Mothership system. All that's required is a little configuration file adjustment to get remote logging started.

Edit the `/etc/syslog-ng/syslog-ng.conf` file on your Mothership system to include the following source entry:

```
source s_tcp {
    # receive 514/TCP syslog events from remote hosts
    tcp(ip(0.0.0.0) port(514) max-connections(1000));
};
```

Then add a destination directive using the `$HOST` macro to specify a specific logfile for each host:

```
# Specific dest per host
destination df_remote_hosts {
    file("/opt/logs/$HOST");
};
```

Add an entry to actually make the logging happen:

```
log {
    source(s_tcp);
    destination(df_remote_hosts);
};
```

Now restart syslog-ng:

```
# /etc/init.d/syslog-ng restart
```

If you run `netstat -anp` you should see this line to indicate that syslog-ng is listening for remote log entries:

```
tcp        0      0 0.0.0.0:514          0.0.0.0:*
LISTEN    2383/syslog-ng
```

Edit the `/etc/syslog-ng/syslog-ng.conf` file on the RIHCD base install system and add the following entry to specify the remote log server (change your IP address accordingly):

```
# remote syslog-ng server via TCP
destination d_mothership { tcp("192.168.148.131" port(514));};
```

And this entry on the RIHCD system to send the logs to the Mothership:

```
# ship all of this stuff to remote syslog-ng server
log {
    source(s_all);
    destination(d_mothership);
};
```

Save your file and restart syslog-ng, and you should see the following on Mothership:

```
mothership:~# cd /opt/logs/
mothership:/opt/logs# ls
192.168.148.129
mothership:/opt/logs# head 192.168.148.129
Sep  9 14:22:07 192.168.148.129 syslog-ng[15265]: syslog-ng starting
up; version='2.0.9'
```

This is an overly simplified configuration and it only scratches the surface of what's possible with syslog-ng. In fact, you could accomplish the same thing (albeit over UDP) with regular syslog. If you haven't used it before, I highly recommend using syslog-ng for its content-matching and filtering abilities, not to mention its remote logging and log forwarding abilities.

5. Getting RIHCD to Phone Home

Now that you've got streamlined SSH access (no password prompt), rsync over SSH, and remote logging enabled, it's time to start configuring the RIHCD base system to phone home to the Mothership at boot time.

5.1. Automating SSH Access at Startup

The `/etc/rc.local` file on the RIHCD system has already been edited to change the password for the handler account. The command to phone home can go in this same file, or into another shell script that gets called from `rc.local`.

After the password has been reset by the `mass_passwd.sh` script, the new password must be sent to the Mothership, otherwise there is no way to remotely log into the RIHCD (unless you've put your own public key in `~handler/.ssh/authorized_keys` on the RIHCD system). This is another reason why using `syslog-ng` over TLS, or SSL with `stunnel` is advisable.

Recall that the username and password combination from the `mass_passwd.sh` script is written to `/mass_passwds/mass_passwd.log`. Edit the `/etc/rc.local` file accordingly:

```
/usr/bin/tail -n 1 /mass_passwds/mass_passwd.log | /usr/bin/logger
```

The next time you restart the RIHCD system, look for the following log entry on motherfish:

```
mothership:/opt/logs# tail -n 1 192.168.148.129
Sep  9 14:49:38 192.168.148.129 logger: Wed Sep  9 12:58:46 CDT 2009
handler      aeyurool
```

This is a good place to start up the remote port forwarding SSH session. Add the following lines to the RIHCD `/etc/rc.local` file (or a different file that is called from `rc.local`):

```
# set the port for SSH forwarding
random_port=$(( $RANDOM+1024 ))
/usr/bin/ssh -n -N -p 2200 -R$random_port:localhost:2222 -i \
/home/handler/.ssh/id_dsa handler@192.168.148.131 &
```

In a large enterprise network with distributed IT staff, it's possible to have multiple CDs booted at one time, and multiple incident handlers needing to log in via

Mothership. So you won't want the remote port forwarding to be set to the same port on Mothership every time. Instead, use the built-in bash function RANDOM. "Each time this parameter is referenced, a random integer between 0 and 32767 is generated." (GNU, 2006) To ensure that a privileged port (below 1024) is not used, add 1024 to the \$RANDOM value, and use that as the remote port to forward to on Mothership.

Since /etc/rc.local runs as root, you'll need to make sure that the public key for the Mothership system is in the /root/.ssh/known_hosts file on the RIHCD system.

```
handler@rihcd:~$ sudo sh
sh-3.2# ssh -p 2200 192.168.148.131
The authenticity of host '[192.168.148.131]:2200
([192.168.148.131]:2200)' can't be established.
RSA key fingerprint is 2b:2e:d8:75:1d:97:f7:1d:5f:c1:36:48:36:c7:42:81.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[192.168.148.131]:2200' (RSA) to the list
of known hosts.
Permission denied (publickey).
sh-3.2# exit
exit
handler@rihcd:~$
```

It's doesn't matter that root wasn't able to connect (it doesn't have a public key on the Mothership). What matters is that the Mothership's RSA key is added to root's ~/.ssh/known_hosts file so that SSH can occur without a hitch when run as root from /etc/rc.local

To test the magic, reboot your RIHCD base system and walk away from it. Log into your Mothership system, and take a peek in the logs for action from the handler account:

```
mothership:/opt/logs# netstat -lanp | grep handler
tcp        0      0 127.0.0.1:17276      0.0.0.0:*
LISTEN    2621/sshd: handler
tcp        0      0 192.168.148.131:2200 192.168.148.129:35409
ESTABLISHED 2619/sshd: handler
tcp6      0      0 :::17276            :::*
LISTEN    2621/sshd: handler
unix 3      [ ]          STREAM  CONNECTED  7180      2619/sshd:
handler
```

```
unix 3      [ ]      STREAM    CONNECTED  7179      2621/sshd:
handler
mothership:/opt/logs# tail 192.168.148.129 | grep handler
Sep  9 16:11:24 192.168.148.129 chage[2776]: changed password expiry
for handler
Sep  9 16:11:24 192.168.148.129 logger: Wed Sep  9 16:11:24 CDT 2009
handler      jooquaih
mothership:/opt/logs#
```

Now log in to the RIHCD system from Mothership:

```
mothership:/opt/logs# ssh -p 17276 handler@localhost
The authenticity of host '[localhost]:17276 ([127.0.0.1]:17276)' can't
be established.
RSA key fingerprint is 35:dd:94:23:3f:4d:d7:7f:4c:73:85:c0:8a:3c:85:cd.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:17276' (RSA) to the list of
known hosts.
handler@localhost's password:
You are required to change your password immediately (password aged)
Linux rihcd 2.6.26-2-686 #1 SMP Fri Aug 14 01:27:18 UTC 2009 i686
Last login: Wed Sep  9 16:06:13 2009 from 192.168.148.1
WARNING: Your password has expired.
You must change your password now and login again!
Changing password for handler.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Connection to localhost closed.
mothership:/opt/logs# ssh -p 17276 handler@localhost
handler@localhost's password:
Linux rihcd 2.6.26-2-686 #1 SMP Fri Aug 14 01:27:18 UTC 2009 i686
Last login: Wed Sep  9 16:15:29 2009 from localhost
handler@rihcd:~$
```

5.2. Automatically Pulling Down New Files from the Mothership at Boot Time

Pre-authenticated SSH access can also be useful for downloading files from the Mothership via rsync (<http://www.samba.org/rsync/>). This makes a great way to

distribute updated analytical scripts or documentation without burning an entirely new CD to distribute. Rsync is an especially nice tool to use as it only transfers the differences between one file and another instead of the entire file from remote to local host. For small files this difference is negligible, but for big files, it is certainly noticeable.

Make a new subdirectory in your /chroot directory on the Mothership system and give the handler account (the handler group, really) read access.

```
mothership:~# cd /chroot
mothership:/chroot# mkdir handler_home_updates
mothership:/chroot# chown root:handler handler_home_updates/
mothership:/chroot# chmod 750 handler_home_updates/
mothership:/chroot# ls -al handler_home_updates/
total 8
drwxr-x---  2 root  handler 4096 2009-09-09 18:39 .
drwxr-xr-x 11 root  root    4096 2009-09-09 18:39 ..
mothership:/chroot#
```

You can make this directory read-write for the handler account, but be aware that it's one more avenue of attack: rogue CD users could exhaust your local filesystem, or cause other users of the CD to download inappropriate data from this directory.

To initiate this automatic rsync from Mothership, add another command to the /etc/rc.local file on the RIHCD base system:

```
/usr/bin/rsync -rv -e "ssh -p 2200 -i /home/handler/.ssh/id_dsa"
handler@192.168.148.131:/handler_home_updates/ /home/handler
```

This uses rsync over SSH (as authenticated by the passwordless private key) to pull down any new files from mothership:/chroot/handler_home_updates to /home/handler on the RIHCD:

```
handler@rihcd:~$ /usr/bin/rsync -rv -e "ssh -p 2200 -i \
/home/handler/.ssh/id_dsa" \
handler@192.168.148.131:/handler_home_updates/ /home/handler
receiving incremental file list
documentation.html

sent 30 bytes  received 266 bytes  118.40 bytes/sec
total size is 163  speedup is 0.55
```

```
handler@rihcd:~$
```

To test, place some new content on mothership:/chroot/handler_home_updates and reboot your RIHCD base system, then check for your file in rihcd:/home/handler

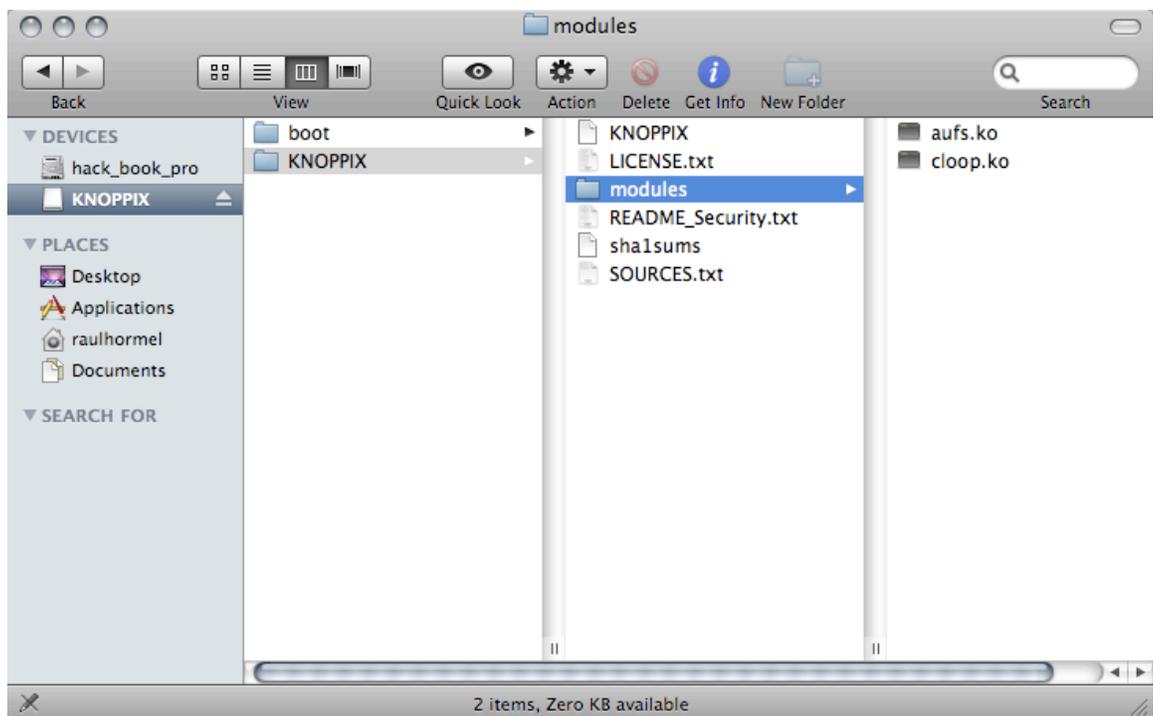
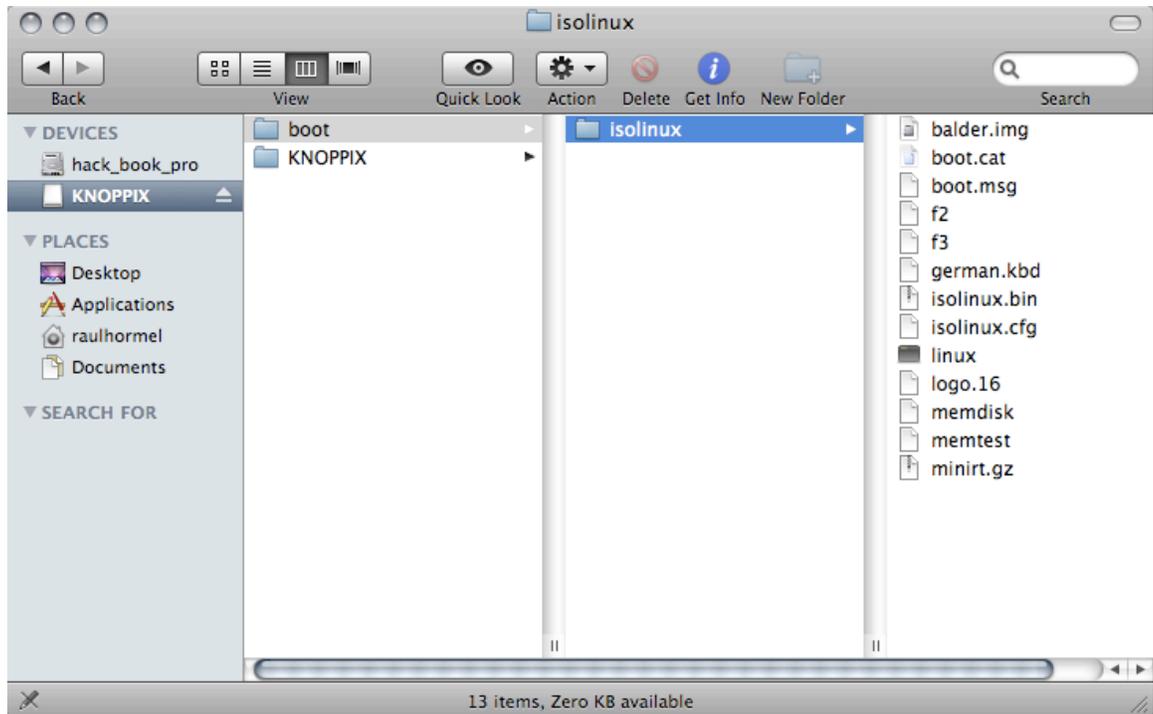
6. What is Knoppix?

Created by Klaus Knopper, Knoppix was one of the first and is now probably the most popular existing Live CD, and with good reason. It's based upon Debian GNU/Linux and automatically detects myriad hardware devices, including sound cards, video cards and monitors. And of course, because it's a Live CD, it runs completely from the read-only CD, not needing to access the local hard drive at all (although it can if you want it to).

The magic in Knoppix comes from the ability to cram a base Linux install of nearly 2 GB into single file of less than 650 MB. That single file is then mounted as a compressed loopback filesystem using the cloop kernel module developed by Klaus Knopper. That cloop filesystem is then overlaid with a read/write ramdisk using aufs (Another Union FileSystem, developed by Junjiro Okajima). The result is a full-featured Linux install that has a read/write file system without writing to the local hard drives.

6.1. Anatomy of a Knoppix CD:

If you boot a Knoppix 6.x CD, the contents of the CD itself are accesible as /mnt-system/ In the screenshots below, I show the CD when mounted (not booted) in a Mac OS X system:



For such a uniquely rich CD, it has surprisingly few files. A brief explanation of what's found:

./KNOPPIX:

KNOPPIX The main compressed loop back file containing the Debian OS

Bert Hayes, bhayes@infosec.utexas.edu

`LICENSE.txt` GNU GPL version 2 license
`README_Security.txt` Warning about built in accounts and su, root access
`SOURCES.txt` Where to get source code for KNOPPIX-specific packages
`modules` Directory containing kernel modules used at boot time
`shasums` SHA1 sums for all of these files, used during CD integrity test

`./KNOPPIX/modules:`

`aufs.ko` Another Union FS module, used to stack disparate file systems as one
`cloop.ko` Kernel module to access compressed loopbak file

`./boot:`

`isolinux` Directory containing boot files used by isolinux boot loader

`./boot/isolinux:`

`balder.img` Floppy disk image used to boot FreeDOS
`boot.cat` Boot catalog file
`boot.msg` Welcome message and version info displayed before boot prompt
`f2` Help screen, shows additional boot options (e.g. memtest, FreeDOS, etc.)
`f3` Help screen, shows boot options used for booting Knoppix
`german.kbd` German language keyboard mappings
`isolinux.bin` 2nd stage boot image, used by isolinux boot loader
`isolinux.cfg` Configuration file for isolinux boot loader
`linux` The kernel file used by Knoppix
`logo.16` Splash screen displayed as part of boot.msg
`memdisk` Another linux kernel, used by memtest
`memtest` Memory tester (memtest86 v1.65)
`minirt.gz` Initial Ram Disk (initrd) used during boot process

Many of these files are optional and can be left out of your final CD, depending on how you wish to distribute it:

`README_Security.txt`
`SOURCES.txt`
`shasums`
`balder.img`

```
f2
f3
german.kbd
memdisk
memtest
```

Some of these files are absolutely critical to booting/operating the CD and must be included:

```
LICENSE.txt
modules/aufs.ko
modules/cloop.ko
boot.cat
isolinux.bin
isolinux.cfg
linux
minirt.gz
```

And a handful of these files are ripe for customization (to be detailed in section 7):

KNOPPIX	Will be created using RIHCD base install
boot.msg	Change this to provide a customized welcome message
f2 & f3	Change these (or create additional screens) for customized help
isolinux.cfg	Customize how Knoppix boots
linux	Creating your own custom kernel for Knoppix is fun and exciting
logo.16	Create a custom splash screen to brand the CD
minirt.gz	Several files zipped into one; contains init startup script

The minirt.gz file is the root file system that the Linux boot image uses, as specified in the isolinux.cfg file; it contains the minimum amount of software necessary to set up the full Knoppix system. (Neggus, 2007)

To access the contents of the minirt.gz file, unzip it and expand it using cpio:

```
# mkdir /knoppix-initrd
# cd /knoppix-initrd
```

```
# cp /mnt-system/boot/isolinux/minirt.gz .
# gunzip minirt.gz
# cpio -idvm <minirt
```

You should have a directory listing that looks like this:

```
root@Microknoppix:/knoppix-initrd# ls
KNOPPIX  bin  etc  init  mnt-system  modules  ramdisk  tmp
UNIONFS  dev  home  mnt  mnt-user  proc  sys
```

All of these listings are directories, except for the `init` file. This file (called `linuxrc` in previous versions of Knoppix) is the main system configuration script, responsible for creating the ramdisk, mounting the compressed loopback file(s), and creating the unionfs. It will be explored in detail in the next section.

It's worth noting most of the directories contained in the compressed `minirt.gz` file are empty, with the exception of `./bin`, `./etc`, and `./dev`. `./bin` contains the busybox minimal shell, and `ntfs-3g`, just enough binary programs to get your Knoppix CD up and running. `./etc` contains a skeletal `mtab` file and `./dev` has placeholder files for directories, device nodes, etc.

6.2. Overview of the Knoppix Boot Process

Since the bulk of this project is taking Knoppix boot functionality and applying it to your own Debian install, it's important to have a good understanding of the Knoppix boot process. The following outline summarizes the Knoppix 6.2 boot process:

1. Computer is turned on. The system BIOS must be configured to boot from a CD for the rest of the boot process to proceed.
2. Provided that the BIOS supports the El Torito specification (bootable CDs), the BIOS scans the CD for a boot record, which will point it to the boot catalog on the disc. The boot catalog contains a list of boot entries, where each entry in the boot catalog points to a boot image if that boot entry is selected. By default, the `isolinux.bin` boot entry is selected (Neggus, 2007).
3. `isolinux.bin` runs as configured by `isolinux.cfg`. The `isolinux.bin` file enables the user to select bootable images listed in the `isolinux.cfg` file. This file specifies the default boot values, including default kernel, initial ram disk (`initrd`), whether or not a

boot prompt is displayed, the amount of time it's displayed, which splash screens and welcome messages are displayed, and a few default kernel options.

4. As it boots, isolinux displays the welcome splash screen and the boot.msg file, followed by a boot prompt. The end user can select f2 to display a list of different bootable images (e.g. FreeDOS, memtest), or select f3 to see a list of options for booting the Knoppix bootable image itself. If no options are selected, the boot prompt times out and the Knoppix boot process proceeds with default values.

5. The linux kernel boots using /boot/isolinux/linux for the actual kernel file (more familiar as /boot/vmlinuz-2.6.31.6 if it were part of a standard install) and minirt.gz as the initial ram disk. Boot options (kernel optional parameters) are passed to the OS (kernel) from the `boot:` prompt or from isolinux.cfg

6. The kernel unzips the minirt.gz file and the init script runs. Preconditions for this script to run are: Kernel contains all drivers necessary to mount initial media, and cloop.ko and aufs.ko are located in /mnt-system/"\$knoppix-dir"/modules/ (Knopper, 2008)

The init script is a big one. It's over 700 lines long, and is written by Klaus Knopper, a man who very obviously knows his BASH shell scripting and shorthand. Its intricacies are not for the faint of heart, but it can be easily customized for this project. See the Appendix section 12.2 for a detailed breakdown of this script.

7. The last thing the init script in minirt.gz does is call the base system's own /sbin/init file, which is controlled by /etc/inittab. Knoppix uses its own inittab file which is most likely very different from the /etc/inittab in your base system. When it comes time to customize your own CD, the two files will be merged.

8. /etc/inittab specifies the default run level as run level 5. It calls /etc/init.d/knoppix-autoconfig at the startup runlevel (rcS).

9. The knoppix-autoconfig script is another wonder written by Klaus Knopper specifically for Knoppix. This script is over 600 lines long. The bulk of the script defines functions which are then called in order towards the end of the script. See the Appendix section 12.3 for a detailed analysis of this script.

10. After `knoppix-autoconfig` runs from runlevel S (at system boot), control is handed back to `init` and `/etc/inittab`. This starts sends the system into runlevel 5 and starts all designated services along the way.

11. The script `/etc/init.d/knoppix-startx` is called from `/etc/inittab` as the system enters run level 5, this script starts up the X server and window manager.

7. Applying Customized Knoppix Magic to Make Your RIHCD

With a basic understanding of how Knoppix boots, the boot scripts and techniques can be applied to the development station running your custom RIHCD install to make it a bootable CD.

If you are not COMPLETELY satisfied with your RIHCD base install, now is the time to go back and tweak it. Everything should be exactly how you want it, because it's about to be burned to a read-only CD.

As shown above, most of the Knoppix magic happens at boot time, and is controlled by files in the `/boot` directory of the CD. This is why the RIHCD base system has a separate partition for `/boot`, it's going to be replaced with the Knoppix `/boot` files that have been customized for your install.

7.1. Preparing the Mastering Environment

If the RIHCD development system is running, go ahead and shut it down, then reboot it from a Knoppix 6.x CD. No GUI is needed for any of these steps, so you may want to boot with the "knoppix 2" option, which defaults to the command line only run level 2.

Once booted, run the following commands to prepare the mastering environment:

```
# mkdir /base_install
# mkdir /extras
# mkdir /rihcd_devel
```

From the installation of Debian on the development system in section 2, my hard drives are partitioned as follows:

```
/dev/sda1  /boot
/dev/sda2  swap
/dev/sda3  /
/dev/sdb1  /rihcd_devel
/dev/sdb2  /extras
```

Mount your partitions and create a couple of new directories for mastering (the base install or development partition is mounted read-only to preclude accidental file overwrites or deletions):

```
root@Microknoppix:~# mount -o ro /dev/sda3 /base_install/
root@Microknoppix:~# mount /dev/sdb1 /rihcd_devel/
root@Microknoppix:~# mount /dev/sdb2 /extras/
root@Microknoppix:~# mkdir -p /rihcd_devel/knoppix/source/KNOPPIX
root@Microknoppix:~# mkdir -p /rihcd_devel/knoppix/master/boot
root@Microknoppix:~# mkdir -p /rihcd_devel/knoppix/master/KNOPPIX
root@Microknoppix:~# mkdir /extras/knoppix_magic
```

Copy all of your RIHCD development system from /base_install/ to /rihcd_devel/knoppix/source/KNOPPIX (this step will take several minutes to complete) (Neggus, 2007):

```
root@Microknoppix:~# cp -Rp /base_install/* \
/rihcd_devel/knoppix/source/KNOPPIX/
root@Microknoppix:~# cd /rihcd_devel/knoppix/source/KNOPPIX/
root@Microknoppix:/rihcd_devel/knoppix/source/KNOPPIX# ls
bin  cdrom  etc  initrd.img  lost+found  media  opt  root  selinux
sys  usr  vmlinuz
boot  dev  home  lib  mass_passwds  mnt  proc  sbin  srv
tmp  var
```

7.2. Cleaning Up the Base System Before Mastering

A little housekeeping is required to prepare your custom Debian install for its eventual life as a Live CD. Consider that space is a premium, so any leftover cruft should be cleared out. This means any .deb files leftover from package installation should be removed, as well as accumulated log files. You may also wish to remove any shell history files or history files from other applications (e.g. ~/.lesshst, ~/.viminfo, etc.)

If you had scratch space or working directories on your development system (rihcd_devel/ and extras/) remove them from the RIHCD base source:

```
root@Microknoppix:/rihcd_devel/knoppix/source/KNOPPIX# rmdir
rihcd_devel/ extras/
root@Microknoppix:/rihcd_devel/knoppix/source/KNOPPIX#
```

If you didn't already run "apt-get clean" when booted in your base install system, you can run it now and specify the non-default cache directory as a command line option:

```
# apt-get clean -o \
dir::cache=/rihcd_devel/knoppix/source/KNOPPIX/var/cache/apt/
```

Log files can be removed simply by deletion, but beware: if your customized disc includes services that log to /var/log, but do not use syslog (e.g. freshclam by default), these services may fail to start if their log files are not present for appending. (This problem with freshclam can be fixed by editing /etc/clamav/freshclam.conf)

You should delete the xorg.conf file from the development system source, otherwise Knoppix will assume that this is the configuration to be used when starting the GUI, and won't create a new file via the mkxorgconfig script.

```
# rm /rihcd_devel/knoppix/source/KNOPPIX/etc/X11/xorg.conf
```

Since the base system as CD will be running the 2.6.31.6 kernel that comes with Knoppix, you will need to move all 2.6.31.6 kernel modules into your source. The existing 2.6.26 kernel modules will not be needed, so they can be removed at this time, saving additional space on the CD.

```
# cd /rihcd_devel/knoppix/source/KNOPPIX/lib/modules/
# rm -rf 2.6.26-2-686
# mkdir 2.6.31.6
# cp -av /lib/modules/2.6.31.6/. 2.6.31.6/
```

If you don't want to leave your .bash_history file around to be burned to CD for all to see, go ahead and nuke those too:

```
# cat /dev/null
>/rihcd_devel/knoppix/source/KNOPPIX/home/handler/.bash_history
# cat /dev/null >/rihcd_devel/knoppix/source/KNOPPIX/root/.bash_history
(if present)
```

It's very likely that you'll be repeating this housekeeping process as you fix errors or otherwise tweak your Live CD. Consider putting all of these steps into a single shell script to speed this process in the future.

7.3. Creating the Base System Compressed Loopback File

Once the development system is cleaned up, it can be compressed into the main KNOPPIX cloop file (/KNOPPIX/KNOPPIX on the final CD).

To do this, the mkisofs command is used with the following switches and arguments, specific for this development environment (Neggus, 2007):

```
# mkisofs -R -U -V "RIHCD Base"          \
    -publisher "I Handler"              \
    -hide-rr-moved -cache-inodes -no-bak -pad \
    /rihcd_devel/knoppix/source/KNOPPIX      \
    | nice -5 /usr/bin/create_compressed_fs - 65536 > \
    /rihcd_devel/knoppix/master/KNOPPIX/KNOPPIX
```

See the man page for mkisofs for an explanation of what each command line switch is for.

create_compressed_fs is part of the cloop_utils package, responsible for taking the output of mkisofs and compressing it to a cloop file.

Running this command will consume all available system resources, not to mention quite a bit of your time, depending on the specs of your mastering station. This command should be written to a file and made executable so that it can be run time and time again without typing it in manually.

Running this against the example RIHCD development system took the base install of around 1 GB and slimmed it down to a 372 MB cloop file.

Now copy the contents of the Knoppix /boot directory into your mastering area:

```
root@Microknoppix:/rihcd_devel/knoppix/master/boot# cp -rv /mnt-
system/boot/* .
root@Microknoppix:/rihcd_devel/knoppix/master/boot# cd isolinux/
root@Microknoppix:/rihcd_devel/knoppix/master/boot/isolinux# ls
balder.img boot.cat boot.msg f2 f3 german.kbd isolinux.bin
isolinux.cfg linux logo.16 memdisk memtest minirt.gz
```

Also copy the aufs.ko and cloop.ko kernel modules that will be needed at boot time, as well as the GPL version 2 license file:

```
root@Microknoppix:/rihcd_devel/knoppix/master/KNOPPIX# mkdir modules
root@Microknoppix:/rihcd_devel/knoppix/master/KNOPPIX# cp -v \
/mnt-system/KNOPPIX/modules/* modules/
root@Microknoppix:/rihcd_devel/knoppix/master/KNOPPIX# cp -v \
/mnt-system/KNOPPIX/LICENSE.txt .
```

7.4. Editing isolinux.cfg to Customize or Remove Boot Options

How much of the original Knoppix boot functionality you remove is up to you. Knoppix boots using Isolinux (<http://syslinux.zytor.com/wiki/index.php/ISOLINUX>) which is controlled via the isolinux.cfg file. This file specifies the default boot options such as whether or not a boot prompt is offered, how long to pause at the prompt before booting, where to display help and welcome messages, and which kernel to use with which kernel options. Open this file in a text editor and take a look around.

If you never plan on using a German keyboard, you can remove the german.kdb file. Likewise, if you don't want end users to be able to perform a memory test, you could remove the memdisk and memtest files; the same goes for booting into FreeDOS. If you do remove this functionality, make sure to edit the f2 file to remove any references to these boot options.

If you want to disable these extra boot functions, you will want to comment out these boot options from the isolinux.cfg file as well:

```
#LABEL memtest
#KERNEL memtest
#APPEND foo
#LABEL dos
#KERNEL memdisk
#APPEND initrd=balder.img
```

Generally speaking, there's not much usefulness in removing functionality. Most useful customizations would be along the lines of removing the boot prompt (PROMPT 0) if building a non-interactive CD, or adding additional help screens (like those accessed by hitting F2 or F3 at boot). You may also find the need to pass specific kernel

parameters at boot time. Although this is unlikely, the APPEND line is where these extra parameters would go.

For this example build, I'm leaving the isolinux.cfg file as is, and editing the files that are called from it in order to change the boot behavior.

Pressing F2 or F3 at a boot prompt will still bring up the f2 and f3 help files, but it can be useful to edit these files, if only to replace KNOPPIX with your own brand name.

An additional boot.msg file for an additional function key (e.g. f6_boot.msg) can be created for more help screens, etc.

You can specify optional boot parameters by editing the "APPEND" lines, for example to turn off using linux swap partitions with the noswap option . The range of options that this config file can specify makes it a fun file to hack on.

7.5. Editing the boot.msg file to Customize the Start Up Splash Screen

In addition to editing the f2 and f3 files, the boot.msg file should be edited to reflect that this is your own custom install, and the logo.16 file should be changed so that the boot process displays your own snappy splash screen instead of Klaus Knopper's.

The boot.msg file from Knoppix 6.0.1 looks like this:

```
^O17^L^Xlogo.16
```

```
KNOPPIX V6.0.1                http://www.knoppix.de/  
  RELEASE: 2009-02-08
```

The control characters (^O17^L^X) specify screen colors and call for the logo.16 file to be displayed, followed by the version of Knoppix, its web site and release date for this version. (Neggus, 2007)

To use your own image file, rename the logo.16 file to back it up, and fire up your favorite image editor. (This example will use GIMP from Knoppix 6.0.1.) Find a favorite image that will look good in 16 colors at 640x400 or so. You'll have a 640x480 space to display your image, but if you don't use the whole 480 pixels in height, you'll be able to show some text below your image without pushing your image off the screen.

Here are the steps I used in GIMP to create my custom logo.

1. Open the image in the GIMP.
2. If it's not already scaled to the right size, select Image->Scale Image and scale it down to around 640x400.
3. Set the color palate to 16 colors by selecting Image -> Mode -> Indexed. Select "Generate optimum Palette" at 16 colors, then click Convert.
4. To save the file as a .ppm file, select File -> Save As Make sure that the "Select File Type" is set to "By extension" and save your file with a .ppm extension.
5. Run the ppmtolss16 Perl script to take your .ppm file as input and spit out a new_logo.16 as output:

```
ppmtolss16 < yerfile.ppm > logo.16
```

6. Copy the new logo.16 file into /extras/scratch_space/boot/isolinux/

7.6. Unpacking minirt.gz to Edit and Customize the init Start Up Script

The next fun file to hack on is the init file in the compressed file minirt.gz.

Here are the steps needed to open the compressed file for access to the init file (Bleßmann, 2009):

```
# gunzip -c /mnt-system/boot/isolinux/minirt.gz >/tmp/minirt
# mkdir /tmp/miniroot
# cd /tmp/miniroot
# cpio -idmv -I /tmp/minirt
# ls
# vim init
```

Recall that the init file runs at boot time, just after the kernel boots and mounts all disk images into the unionfs. This file is a big hairy beast and for anyone but the very knowledgeable and patient, most of its contents are best left as is.

There are a couple of minor edits that can be made for your environment. For example, to brand your CD:

```
Line 11: DISTRO="RIHCD"
```

Mount any disk partitions read-only (this could also be accomplished by adding "forensic" to the list of boot parameters via the APPENDS line in isolinux.cfg):

```
Line 283: RW="ro"
```

Once you're done making edits, to recompress this initial ram disk back to a minirt.gz file, run the following:

```
# cd /tmp/miniroot
# find . | cpio -o -H newc > /tmp/minirt.new
# cd /rihcd_devel/knoppix/master/boot/isolinux
# gzip -9 -c /tmp/minirt.new > minirt.gz
```

If you are knowledgeable and patient, the init script has a ton of potential for hacking and additional customization. For the purposes of this paper, I'm changing only the bare minimum required for branding and booting the base system.

7.7. Editing Initialization Scripts Outside of /boot

Once the initial ramdisk `init` script finishes running, all file systems are mounted, *then* the `/etc/inittab` file is run, calling all of the other system initialization scripts (knoppix-specific and otherwise) along the way.

To edit the knoppix-specific files that are outside of `/boot`, create a new working directory, e.g. `/extras/knoppix_magic` and copy all knoppix-specific scripts and files into this directory. At a minimum, these files are:

```
/etc/inittab
/etc/init.d/knoppix-autoconfig
/etc/init.d/knoppix-startx
/etc/init.d/knoppix-halt
/etc/init.d/knoppix-reboot
/sbin/hwsetup
/sbin/mkxorgconfig
/usr/sbin/rebuildfstab
/usr/bin/flash-knoppix
```

(Note that Klaus Knopper has written a multitude of other scripts and utilities, many of which are distributed within Knoppix, but are not absolutely necessary for system booting and hardware configuration. See <http://debian-knoppix.alioth.debian.org>)

The files listed above should be copied into the destination directory with the full source path intact, so that you eventually have `/extras/knoppix_magic/etc/init.d/knoppix-autconfig` etc. Using the `--parents` switch for the `cp` command will do this:

```
# mkdir /extras/magic_knoppix
# cp --parents /etc/init.d/knoppix-* /extras/knoppix_magic/
# cp --parents /etc/inittab /extras/knoppix_magic/
...

```

7.7.1. Editing the `/etc/inittab` File

One of the last things that the `init` script from the initial ram disk (`minirt.gz`) does is call the base system's own `/sbin/init`, which is controlled by the base system's `/etc/inittab` file.

Lines 703-725 of this `init` script handle how the `inittab` file is copied, based on whether the “`adriene`” or “`secure`” boot options are given. The base install system that I've described has neither the `inittab.adriene` or the `inittab.secure` files, so this part of the `init` script uses the only `inittab` file it can find, that of the base system (after all disk images are mounted into the `unionfs`). To add the Knoppix `inittab` functions to your own `inittab` file, copy your base system `inittab` file to `inittab.bak`, and edit the original. (The `.bak` file will not be burned to the CD. See section 7.7)

```
# cd /rihcd_devel/knoppix/source/KNOPPIX/etc/
# cp inittab inittab.bak
# vim inittab

```

Edit the following lines to read:

```
id:5:initdefault:
si::sysinit:/etc/init.d/knoppix-autoconfig

```

And add a section to automatically start X via `/etc/init.d/knoppix-startx`:

```
# X mode
x0:5:wait:sleep 2
x1:5:respawn:/etc/init.d/knoppix-startx start >/dev/console 2>&1

```

Add a section to handle shutdown or reboot with knoppix scripts:

```
# Halt or Reboot
z0:0:wait:/etc/init.d/knoppix-halt

```

```
z6:6:wait:/etc/init.d/knoppix-reboot
```

Note that a big difference between the inittab file that Knoppix uses and the one that your Debian base install uses is that Knoppix does not call the SystemV init scripts (see lines 22-31 of the knoppix /etc/inittab file). In the case of the RIHCD, it makes sense to use these init scripts, since there is a wide range of services that need to be started (e.g. SSH, freshclam, etc.). See Appendix A for a listing of the /etc/inittab file used in this example RIHCD.

7.7.2. Editing the /etc/init.d/knoppix-autoconfig Script

The knoppix /etc/inittab starts system services by running the knoppix-autoconfig script at runlevel S, system initialization time, instead of running the scripts in /etc/rcS.d in your base system.

Opportunities for extensive customization of your CD via the knoppix-autoconfig script are abundant but are best left for those with experience writing BASH scripts. A few minor edits are called for, however:

```
# vim /extras/knoppix_magic/etc/init.d/knoppix-autoconfig
```

Line 290: Set your time zone accordingly, eg:

```
TZ="America/Chicago"
```

Comment out line 458 and 459 because with syslog-ng there is no /sbin/klogd, nor is there a /sbin/syslogd. If we attempt to start syslog-ng at this point, it will fail since it has no network connectivity yet.

```
#!/sbin/klogd -c 1 -x  
#[-r /etc/syslog-knoppix.conf ] && /sbin/syslogd -f /etc/syslog-  
knoppix.conf || /sbin/syslogd
```

This should be the bare minimum that needs to be done to this file to make it customized for the RIHCD application.

7.7.3. Editing the /etc/init.d/knoppix-startx Script

Once the system hits runlevel 5, the /etc/init.d/knoppix-startx script is called. This script does what it advertises, but it makes a few assumptions that won't apply to the RIHCD. To make it behave properly, edit lines 32-34 to read:

```
USER="handler"  
GROUP="handler"  
SESSION="fluxbox"
```

Also, edit line 46 to replace “startlxde” with “startfluxbox”:

```
type -p "$STARTUP" >/dev/null 2>&1 || STARTUP="startfluxbox"
```

Now is a good time to make sure that the `/etc/X11/Xwrapper.config` has the following line:

```
allowed_users=anybody
```

If it's set to “`allowed_users=console`”, the `knoppix-startx` script won't be able to start the X server.

7.8. Creating the Knoppix-Specific Compressed Loopback File

Recall that the `init` shell script contained within `/boot/minirt.gz` file is what mounts the main KNOPPIX cloop file (`/KNOPPIX/KNOPPIX`) into the union filesystem. [See lines 615-643]. Take a closer look at this function, and you'll see that it also looks for and mounts any `/KNOPPIX/KNOPPIX[0-9]` files into the union file system as well. Each KNOPPIX cloop file is mounted “on top” of the previously mounted KNOPPIX cloop file. So, `KNOPPIX3` would be mounted “on top” of `KNOPPIX2` which would be mounted on top of `KNOPPIX`, etc. Overlaid on top of all of these in the union file system would be the read/write ramdisk.

This little bit of forward thinking and modular design is a gift for anyone who wants to hack on their own Live CDs and produce multiple flavors of the same base system. Not only do you have the option of including additional software by including an extra `KNOPPIX` cloop file, but you can actually put all of the `knoppix`-specific startup and shutdown scripts in their *own* `KNOPPIX` cloop file.

If you have the development system compressed into the `KNOPPIX` cloop file in `/KNOPPIX/KNOPPIX` on the CD, and you have all of your `knoppix`-specific scripts (e.g. `/etc/init.d/knoppix-autoconfig` etc.) in a compressed cloop file called `KNOPPIX1`, the `init` script will mount the cloop files into the union files sytem, overlaying the `knoppix`-specific scripts on top of the base install.

It is this modular separation of functionality that makes it possible to take *any* modestly sized base Debian install and make it into a Knoppix-based Live CD.

After the files in section 7.5 are edited and saved, you can use the same process to generate the cloop file for them that was used to create the cloop file for the base install:

```
/usr/bin/mkisofs -R -U -V "RIHCD" \
  -publisher "I Handler" \
  -hide-rr-moved -cache-inodes -no-bak -pad \
  /extras/knoppix_magic \
  | nice -5 /usr/bin/create_compressed_fs - 65536 > \
  /rihcd_devel/knoppix/master/KNOPPIX/KNOPPIX1
```

Since there's much less data to be compressed compared to the base system cloop file, this process will be very speedy.

8. Putting the CD in RIHCD

Once the KNOPPIX cloop file for the base install has been created, as well as the KNOPPIX1 cloop file for knoppix-specific files and scripts (as well as any other desired cloop files for optional software, etc.), the final CD can be burned.

At this point, the /rihcd_devel/knoppix/master/ directory should look like this:

```
root@Microknoppix:/extras/scripts# ls -R /rihcd_devel/knoppix/master/
/rihcd_devel/knoppix/master/:
KNOPPIX  boot
```

```
/rihcd_devel/knoppix/master/KNOPPIX:
KNOPPIX  KNOPPIX1  LICENSE.txt  modules
```

```
/rihcd_devel/knoppix/master/KNOPPIX/modules:
aufs.ko  cloop.ko
```

```
/rihcd_devel/knoppix/master/boot:
isolinux
```

```
/rihcd_devel/knoppix/master/boot/isolinux:
balder.img  boot.msg  f3          isolinux.bin  linux      logo.16.bak
memtest
```

```
boot.cat      f2          german.kbd  isolinux.cfg  logo.16  memdisk
minirt.gz
```

8.1. Generating SHA1 Sums for CD Integrity Checking

The “testcd” boot option will calculate SHA1 hashes for all files listed in the sha1sums file and compare them against the previously generated hash in that same file in order to ensure that no files are corrupted. To generate this list of hashes, run the following commands (Neggus, 2007):

```
# cd /rihcd_devel/knoppix/master
# find -type f -not -name sha1sums -not -name boot.cat \
    -not -name isolinux.bin -exec shasum '{}' \; > KNOPPIX/sha1sums
```

At this point, the boot.cat and isolinux.bin files for this CD have not yet been generated, so they should be excluded from hash checks.

8.2. Creating the Final .iso Image

Once all of the pieces are put together, actually creating the CD is pretty easy. The following commands will generate the final .iso image (Neggus, 2007):

```
# cd /rihcd_devel/knoppix/master
# mkisofs -pad -l -r -J -v -V "RIHCD" -no-emul-boot -boot-
load-size 4 \
    -boot-info-table -b boot/isolinux/isolinux.bin \
    -c boot/isolinux/boot.cat -hide-rr-moved \
    -o /rihcd_devel/knoppix/rihcd.iso /rihcd_devel/knoppix/master
```

See the mkisofs man page for full details on what the command line switches do.

The resulting .iso file, /rihcd_devel/knoppix/rihcd.iso can be transferred from the development system using a variety of methods, e.g. FTP, netcat, SSH, etc.

Actually burning the .iso image to a CD is left as an exercise for the reader, since everyone has their own favorite software/system for burning.

9. Additional Projects and New Directions for RIHCD

Once the RIHCD has been successfully created, any number of variations and additional customizations are possible.

9.1. Additional Software to be Accessed Without Booting from the RIHCD

Any files that are placed in `/rihcd_devel/knoppix/master/` can be accessed from the CD in a running system (not already booted from the RIHCD). This can be a good spot for documentation, or additional utilities for analyzing a potentially compromised system that has not yet been rebooted.

9.1.1. Microsoft Sysinternals

For a CD that is designed to be used interactively by on-site incident handlers, including all of the Windows Sysinternals tools (<http://technet.microsoft.com/en-us/sysinternals/default.aspx>) in this directory will allow analysts to run system analysis tools on a live, running system. There is a great deal more information that can be gleaned from a running system than from one that's been rebooted.

9.1.2. Statically Linked Linux Binaries

Statically linked binary files of common Linux commands burned to a read-only medium are incredibly useful tools to analyze a compromised Linux system for rootkits. Statically linked binaries do not rely on possibly corrupt libraries on the host system, and instead, have all necessary code compiled directly into the executable binary file. Installing rkhunter (http://www.rootkit.nl/projects/rootkit_hunter.html) with all of its requisite tools (e.g. dd, ls, w, tcpdump, lastlog, less, more, cat) makes a great addition to this CD. A good guide to creating these binary files is provided by David Hoelzer, at <http://enclaveforensics.com/Blog/files/b85f4d072b82d7183c144ce38d634229-18.html>

9.2. Putting RHICD on a USB Thumb Drive

The Knoppix 6 CD comes with the flash-knoppix script, which will install the contents of the Knoppix CD to a USB thumb drive, or other media. This script relies on the following Debian packages which may not be installed on your RIHCD development system: syslinux, diaglog, and sfdisk. It also relies on ms-sys, which can be obtained

from <http://ms-sys.sourceforge.net>. As more and more systems are bootable from removable media such as thumb drives, this makes a great way to keep your custom incident handling system with you wherever you go.

9.3. Making the RIHCD an Automated Incident Analysis Disc

By combining the disabling of the boot prompt in the `isolinux.cfg` file with a careful implementation of initialization scripts and system analysis scripts, a CD can be created which neither requires nor accepts any user input from the console, runs a series of analytical tools, and reports the findings back to the Mothership system.

9.4. Making Modular Packages of Optional Software for RIHCD

The same methods to create the compressed loopback (cloop) file of knoppix-specific scripts in section 7.2 can be used to create additional cloop files containing optional extra software. For example, install your favorite suite of pen-testing tools to `/opt` and then use `mkisofs` and `create_compressed_fs` as shown in section 7.2 to create a `/KNOPPIX/KNOPPIX3` file that can be optionally included with the RIHCD base system.

9.5. Running a Squid Proxy on the Mothership System

In well-managed network environments, the IP addresses of compromised systems are put into a quarantine network where they are unable to reach the Internet (and conversely, the attacker on the Internet can no longer reach the compromised system). To provide web access to compromised systems running the CD in the quarantine network, set up a Squid proxy server (also in the quarantine network, but allowed to send traffic out) and configure the web browser on the CD to use this proxy. To avoid abuse of the proxy, enable logging, configure the proxy to require authentication and only allow access to a whitelist of URLs.

10. Conclusion

Live CDs such as Knoppix can be valuable tools for post-mortem analysis of compromised systems, but they can be restrictive because of the limited number of forensic or analytical tools included. Even forensic-specific Live CDs like Helix, which

provides an extensive collection of analytical software, is useless when the compromised system is not immediately physically accessible.

Any experienced incident handler is going to have his or her own favorite collection of software and scripts for system analysis. Aggregating all of these tools in a custom Linux install is something that typically happens over time as the incident handler maintains their own workstation, or jump bag. Making this collection of tools remotely deployable solves the problems of analyzing compromised systems that are physically unreachable by the incident handler. Making the remotely deployable suite of tools remotely *accessible* by the incident handler solves the problem of directly accessing a compromised system that is on a NAT network.

Knoppix implements a unique combination of initialization scripts and kernel modules (cloop and aufs) that make a modestly sized Linux system fit onto a bootable CD that can be run without any local hard drive interaction.

The methods used by Knoppix, especially Knoppix 6.x, are modular enough to be applied to virtually any customized Debian GNU/Linux install, making a custom bootable system that can be deployed as easily as downloading and burning a .iso file.

While there are limits to what can be done with a bootable CD, either because of space constraints or network restrictions, many of these limits can be circumvented by incorporating an additional networked system that is reachable by systems booted from the CD.

If the operating system on the custom CD can use SSH to login to this dedicated system, then remote port forwarding can be used to enable access, both command line and GUI, back to the system running the CD, even if it is on a NAT network. Additionally, the dedicated system can be used as a file repository, both for the results of analytical scripts to be uploaded from the compromised system, as well as for software packages and new analytical scripts to be downloaded from the bootable CD when it starts up.

The tools and techniques employed by Knoppix to make it a bootable OS are as open to hacking and customization as Linux itself, and provide a rich starting point for an

inspired systems hacker to build multiple distributions of bootable systems for multiple implementations.

11. References

- Rankin, K (2005). *Knoppix Hacks*. Sebastopol, CA: O'Reilly Media, Inc..
- Rankin, K (2008, May 1). *Remaster Knoppix without Remastering*. Retrieved from <http://www.linuxjournal.com/article/10075>
- Neggus, C (2007). *Negus Live Linux Series Live Linux® CDs: Building and Customizing Bootables*. Boston, MA: Prentice Hall.
- Debian GNU/Linux (2008), man page for `shadow` retrieved on Debian GNU/Linux 5.0
- Medialogic S.p.A. (2009). NoMachine NX – Download: NX Free Edition for Linux. Retrieved on 12 August 2009 from the NoMachine.com web site: http://www.nomachine.com/download-package.php?Prod_Id=1079
- Medialogic S.p.A. (2009). NoMachine NX – Documentation. Retrieved on 12 August 2009 from the NoMachine.com web site: <http://www.nomachine.com/documentation/admin-guide.php>
- Fourdan, O (n.d.). *Xfce - Desktop Environment*. Retrieved on 28 August 2009 from the Xfce.org web site: <http://www.xfce.org>
- BSD (2009), man page for `sshd_config` retrieved on Debian GNU/Linux 5.0
- Schroder, C (2009). http://tuxcomputing.com/cookbook/mass_passwd. Retrieved on August 08 2009 from the Tux Computing web site: http://tuxcomputing.com/cookbook/mass_passwd
- Srisuresh, P, & Egevang, K (2001, January). *RFC 3022 - Traditional IP Network Address Translator (Traditional NAT)*. Retrieved from <http://tools.ietf.org/html/rfc3022>
- GNU (2004), man page for `rbash` retrieved on Debian GNU/Linux 5.0
- Tessio, A (2009), man page for `makejail` retrieved on Debian GNU/Linux 5.0
- BalaBit IT Security (2009). Syslog Server | Syslog-ng, Retrieved 14 August 2009 from the BalaBit IT Security web site: <http://www.balabit.com/network-security/syslog-ng/>
- GNU (2006), man page for `BASH` retrieved on Debian GNU/Linux 5.0
- Debian GNU/Linux (2006), man page for `mkisofs` retrieved on Debian GNU/Linux 5.0
- Bert Hayes, bhayes@infosec.utexas.edu

Bleßmann, B (2009, April 22). *Nabble - debian-knoppix - cd 6.0.1 mount problem fromhd*. Retrieved from <http://www.nabble.com/cd-6.0.1-mount-problem-fromhd-td23029838.html>

12. Appendix

12.1. Debian Packages to Include When Building the RIHCD Development System.

These tools will not be included as part of the minimal build, and should be installed immediately after the installation process finishes. These tools can all be installed from the command line using `apt-get install packagename`. Most, if not all of these packages will have other packages that they depend on which must be installed as well. When prompted to install these as well, it's safe to say yes.

`sysv-rc-conf` – For system tuning: turning init.d scripts on/off at boot

`syslog-ng` – The next generation of syslog (more on this later)

`stunnel4` – Wraps TCP sessions in SSL

`pwgen` – For generating passwords

`ntpdate` – Set the date/time

`openssh-server` – For providing secure remote access

`ClamAV` – Open-source virus scanner

`tcpdump` – You love it

`xterm` – ‘Nuff said

`wireshark` – GUI packet sniffer

`iceweasel` – Non-Mozilla branded version of firefox

`rdesktop` – Windows Remote Desktop Protocol (RDP) client for Linux

`xtightvncviewer` – VNC client

`xpdf-reader` – PDF viewer

vim-full – Improved Vi (if that's your flavor)

emacs – Text editor preferred by some (you know who you are)

samba-common – For connecting to Windows SMB shares

smbclient – See above

rkhunter – Root Kit Hunter

python – Scripting language required by many tools

lynx – Text based web browser

links – Text based web browser that supports frames

ncftp – Necessary for installing perl modules via CPAN

dpkg-dev – Necessary for making your own .deb files from source

debhelper – See above

binutils – Includes “strings” command for finding ascii strings in binary files

arpwatch – Alerts on ARP shenanigans

sleuthkit – Open source forensic tool

autopsy – Web GUI for sleuthkit

k3b – CD burning software (optional, installs gobs of dependencies also)

curl – Used to grab files from HTTP, HTTPS, FTP servers (like wget)

dd_rescue – Copies data from one file (or block device) to another

ntfsprofs – NTFS utilities

whois – Get network registration information

telnet – Always nice to have

minicom – Serial terminal

ftp – FTP client

dosfstools – Utilities to create and check MS-DOS FAT file systems

foremost – Recover files using their headers, footers, and data structures

ghex – GUI hex editor

hexedit – Console hex editor

hfsplus – Tools to access HFS+ formatted volumes

hfsutils – Tools for reading and writing Macintosh volumes

ngrep – Grep for network traffic

parted – The GNU Parted disk partition resizing program

rsync – Synchronize files to/from remote/local copies

dialog – Used for displaying dialog boxes from shell scripts

syslinux – Used to create bootloader for RIHCD use on USB thumb drive

12.2. Analysis of Knoppix init Script

Below is an overview of the init script included in minirt.gz, each action is followed by the [line-number] corresponding to its location in the init script:

- Set \$DISTRO (e.g. KNOPPIX) [11]
- Set colors used during boot process [13-36]
- Specify where all knoppix files are located [41]
- Set localization - English or German? [45]
- Define system functions that are called later in this script [114-270]
- Mount /proc [276]
- Link /proc/mounts to /etc/mstab [279]
- Read command line parameters [281]
- mount /sys [290]
- Display welcome message [292]
- Load modules [299]

- Check to see if Terminal Service boot option is specified, if so, load NFS modules, start up networking, and look for terminal server [301-423]
- Establish TOTALMEM & print kernel info [425-436]
- “Return existing device names listed as regular expressions” [438]
- Look for Knoppix root (check USB devices) [450-474]
- Run check_sha1sums if requested to test disc integrity [478]
- Create tmpfs /ramdisk [490]
- Copy to RAM? Copy to hard drive? [493]
- Define functions to run commands from /KNOPPIX directory on CD [503-514]
- Define createdata() function; if booted from read/write media, prompt user with dialog box asking to create persistent disk image (encrypted or not) [534-573]
- Define mountdata() function to mount user-generated data file (knoppix.img or knoppix.aes) if present, otherwise create using createdata() function [575-613]
- Define mountknoppix() function to mount all found KNOPPIX cloop files (e.g. /KNOPPIX/KNOPPIX, /KNOPPIX/KNOPPIX1, etc.). Note: This function is critical and will be relied on when creating and mounting additional cloop files for including optional extra software on your customized CD [615-643]
- Define mountunion() function to mount disparate file systems into single merged directory using aufs. This function always mounts the KNOPPIX cloop file at the bottom of the stack of file systems in the union. [645-652]
- Set \$PATH to include /bin in minirt.gz as well as /UNIONFS/bin:/UNIONFS/sbin [671]

- “Link directories in order to create a writable system” -- moves existing directories into the union, then removes the old directory handles [675-683]
- Mount /home into unionfs [687]
- Check for updates on disc (e.g. /KNOPPIX/update*.tar.gz) and install if needed [691-700]
- Check for boot options for “secure” or “Adrienne” and handle moving /etc/inittab file accordingly to specify different initialization scripts [702-725]
- Start init (the /sbin/init file in your development system) [738]

12.3. Analysis of Knoppix knoppix-autoconfig script

The bulk of this script is spent defining functions which are then called in specific order towards the end of the script.

- Define functions
 - o checkbootparam() [17-24]
 - o progress() -- make progress bar happen [38-64]
 - o getbootparam() [66-78]
 - o startudev() [90-115]
 - o start_modules() [118-127]
 - o start_sound() [129-140]
 - o start_net() [142-158]
 - o start_swap() [160-167]
 - o netbook_specials() -- extra modules for eeepc (camera, wlan, cardreader) [169-177]
 - o waitio() – wait for IO on system to settle down [180-187]
 - o start_services() [190-201]] -- turn on: dbus, hal, netbook_specials, start_sound, checkbootparam "nfsdir" || start_net, start_swap
 - o bailout() [216-221]

- localize() [224-366]
- start_proc() – mount and tune /proc [373-384]
- check_start_debug() – did system boot with debug option? [386-389]
- check_root_fs() [391-393]
- check_start_splash() [395-400]
- start_sys() – mount /sys if not already mounted [402-405]
- start_clock() [407-412]
- check_installed() [414-416]
- start_hostname() [418-423]
- start_loopback() [425-427]
- start_fs() [429-454]
- start_log() [456-461]
- start_devpts() [463-466]
- start_hwsetup() [468-476]
- start_dbus() [478-481]
- start_hal() [483-486]
- start_acpi() [488-503]

Once the functions in used the script are defined, they are called in the following order:

- start_proc [511]
- check_start_debug [514]
- check_rootfs [517]
- start_sys [520]
- start_clock [523]

- check_start_splash [526]
- check_installed [529]
- start_hostname [532]
- start_loopback [535]
- start_fs [538]
- localize [541]
- progress – starts the progress bar [544]
- start_modules [547]
- start_udev [550]
- start_log [554]
- start_devpts [557]
- start_hwsetup [560]
- start_dbus [563]
- start_hal [566]
- start_acpi [569]
- start_sound [572]
- start_services [574]
- wait for udev to complete [580]
- clear & end progress bar [586-591]
- run /KNOPPIX/knoppix.sh if present [602-608]

12.4. /etc/inittab Used by RIHCD

```
# /etc/inittab: init(8) configuration.
# $Id: inittab,v 1.91 2002/01/25 13:35:21 miguels Exp $

# The default runlevel.
```

```
id:5:initdefault:

# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
#si::sysinit:/etc/init.d/rcS
si::sysinit:/etc/init.d/knoppix-autoconfig

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

#l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
#l6:6:wait:/etc/init.d/rc 6

# Halt or Reboot handled by knoppix
z0:0:wait:/etc/init.d/knoppix-halt
z6:6:wait:/etc/init.d/knoppix-reboot

# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

# Action on special keypress (ALT-UpArrow).
#kb::kbrequest:/bin/echo "Keyboard Request--edit /etc/inittab to let this work."

# What to do when the power fails/returns.
pf::powerwait:/etc/init.d/powerfail start
pn::powerfailnow:/etc/init.d/powerfail now
po::powerokwait:/etc/init.d/powerfail stop

# /sbin/getty invocations for the runlevels.
```

```
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
#
# Format:
# <id>:<runlevels>:<action>:<process>
#
# Note that on most Debian systems tty7 is used by the X Window System,
# so if you want to add more getty's go ahead but skip tty7 if you run X.
#
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6

# X mode
x0:5:wait:sleep 2
x1:5:respawn:/etc/init.d/knoppix-startx start >/dev/console 2>&1

# Example how to put a getty on a serial line (for a terminal)
#
#T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100
#T1:23:respawn:/sbin/getty -L ttyS1 9600 vt100

# Example how to put a getty on a modem line.
#
#T3:23:respawn:/sbin/mgetty -x0 -s 57600 ttyS3
```