University of Amsterdam

Bootable Linux CD / PXE for the remote acquisition of multiple computers

Dennis Cortjens
dennis.cortjens@os3.nl

REPORT

5th of July, 2014

Abstract

In the field of digital forensics the acquisition of multiple computers in large IT infrastructures have always been a complex and time consuming task. Especially when one doesn't know which computer to investigate and therefore needs to acquire them all. Triage software has increased the efficiency in cases like this. The software gives an indication which computers to acquire, but one still needs to disassemble and acquire storage devices of the specific computers on the crime scene. In this study two concepts of automated remote acquisition of multiple computers are researched and tested on performance. One of the concepts (based on iSCSI) is developed into a proof concept, called the Remote Acquisition Boot Environment (RABE). Although it is not yet feasible for the remote acquisition to succeed the traditional method of acquiring computers, it could make the remote acquisition a time efficient solution in the near future.

Acknowledgement

I would like to thank the Netherlands Forensic Institute's digital technology team for their hospitality and pleasant and stimulating work environment. I want to extend my gratitude to Zeno Geradts and Ruud Schramp for their trust and support.

Contents

1 Introduction

Over the last 10 years our world has been more and more digitized. We have access to computers at school, our work and in public places. This has led to an increase of large IT infrastructures with multiple computers (clients and servers). Within these infrastructures the system administration is automated or can be done remotely. Unfortunately, this is not always the case in all IT fields.

1.1 Problem

In the field of digital forensics the acquisition of multiple computers in large IT infrastructures have always been a complex and time consuming task. Especially when one doesn't know which computer to investigate and therefore needs to acquire them all. Triage software has increased the efficiency in cases like this. The software gives an indication which computers to acquire, but one still needs to disassemble and acquire storage devices of the specific computers on the crime scene. At companies, data centres and universities this is quite an issue.

It's hard to automate the full acquisition process, but this could be automated. There is software available that could perform such tasks, but this software doesn't give a clear insight on its process and is quite expensive. With the available open source Linux distributions and software there could be a solution for this problem. This will make the acquisition within large IT infrastructures easier and faster, while maintaining the forensic value and validity.

1.2 Position

A study on the remote acquisition of computers with a bootable Linux CD / PXE environment resulted with no information on the matter. However, there is some material closely related to the subject.

In 2004 Owen OConnor wrote an article on deploying forensic tools via PXE. He described PXE as an automated and parallel solution for investigating computers in a network without the need for investigating each computer in turn. He mentioned KNOPPIX as a working environment and also addressed some DHCP issues with PXE. He focused on the triage part of a forensic investigation. [1] This article is somewhat outdated, but provides some theoretical basics for this research.

In 2013 Martin B. Koopmans and Joshua I. James wrote a paper on automated network triage. They described a working client-server network triage environment based on PXE booting and created a tool for investigating client computers in large IT environments, called the Automated Network Triage (ANT). They also focused on the triage part of a forensic investigation. [2] This paper provides a technological blue print for one of the concepts in this research.

1.3 Scope

The main question for this research is:

Can a bootable Linux CD / PXE be build for the remote acquisition of multiple computers and how does it perform compared to the traditional method?

This question is researched by the following sub questions:

1. Which Linux distribution will be suited as a bootable environment?
2. How will the bootable environment acquire storage devices securely across the network?
3. What is needed within the bootable environment?
4. Which settings need to be configured for the bootable environment?
5. Which concept will have the best performance?
6. Could the remote acquisition across a LAN be worthwhile?
7. Could the remote acquisition across the internet be worthwhile?
8. Are there other ways for the remote acquisition?

This research focuses on the client side of the remote acquisition. However, a basic server side is build for the proof of concept.

It creates a CD / PXE bootable environment with the following requirements:

- Requires minimal resources from the computer (client side)
- Runs on most hardware (universal)
- Sets configuration settings before CD / PXE image creation
- Requires no configuration after CD / PXE image creation
- Requires no configuration after bootable environment is loaded
- Distributes physical storage devices securely across the network

1.4 Hypothesis

The assumption/hypothesis for this research is:

The remote acquisition of multiple computers (in general) is slower then the traditional method and across the internet it is slower then across a LAN. However, if the acquisition is performed remotely without being on location, it can be done parallel to other activities. This could make it a time efficient solution for partial and sparse acquisition in the future.

2 Background

2.1 Bootable Linux CD

A bootable Linux CD or so-called live image is a complete Linux operating system that can boot from an optical disc, USB stick or Preboot eXecution Environment (PXE). It runs in the computer's memory and allows an operating system to run without installing or making changes to the computer's original configuration and files. [3] There are many live image of known Linux distributions. Well-known live images are KNOPPIX and Ubuntu which can be used for various purposes. [4]

Live images can be adjusted to run special (start-up) scripts and contain special drivers and software. The process of adjusting the contents of a live image is called remastering. [5]

Although live images run almost fully in the computer's memory, running an operating system from an optical disc is slower than running from a flash or hard disk drive. Therefore a live image is rarely used as the default operating system of a computer. However, it is often used for system recovery and restore, because it can recover files and restore configurations without the need for the broken or corrupt operating system. Nowadays, it is also used in computer forensics. A live image provides the digital forensic investigator a working environment that doesn't change the computer's original configuration and files, especially when the live image doesn't mount the storage devices automatically. Thus, forensic images of storage devices can be made without disassembling the computer.

2.2 Preboot eXecution Environment (PXE)

The Preboot eXecution Environment (PXE) is a boot system on most modern computers with cabled network connectivity. It is a combination of several network protocols (IP, UDP, DHCP and TFTP) which provide a client-server boot environment. It is therefore often referred to as a network boot. [6]

The PXE process consists of the following steps:

1. The PXE firmware of the client computer initiates a PXE session by broadcasting an extended DHCPDISCOVER with PXE options.
2. The client receives an extended DHCPOFFER with PXE info, containing a client IP address and the IP address of the PXE server.
3. The client contacts the PXE server by broadcasting an extended DHCPREQUEST with PXE options and continues the DHCP handshake.
4. The client receives an extended DHCPACK with PXE info, containing the IP address of the TFTP server and the path of the file(s) and finishes the DHCP handshake.
5. The client contacts the TFTP server by unicasting a RRQ.
6. The client downloads the file(s) from the TFTP server.

The steps of the PXE process are illustrated in figure 1.

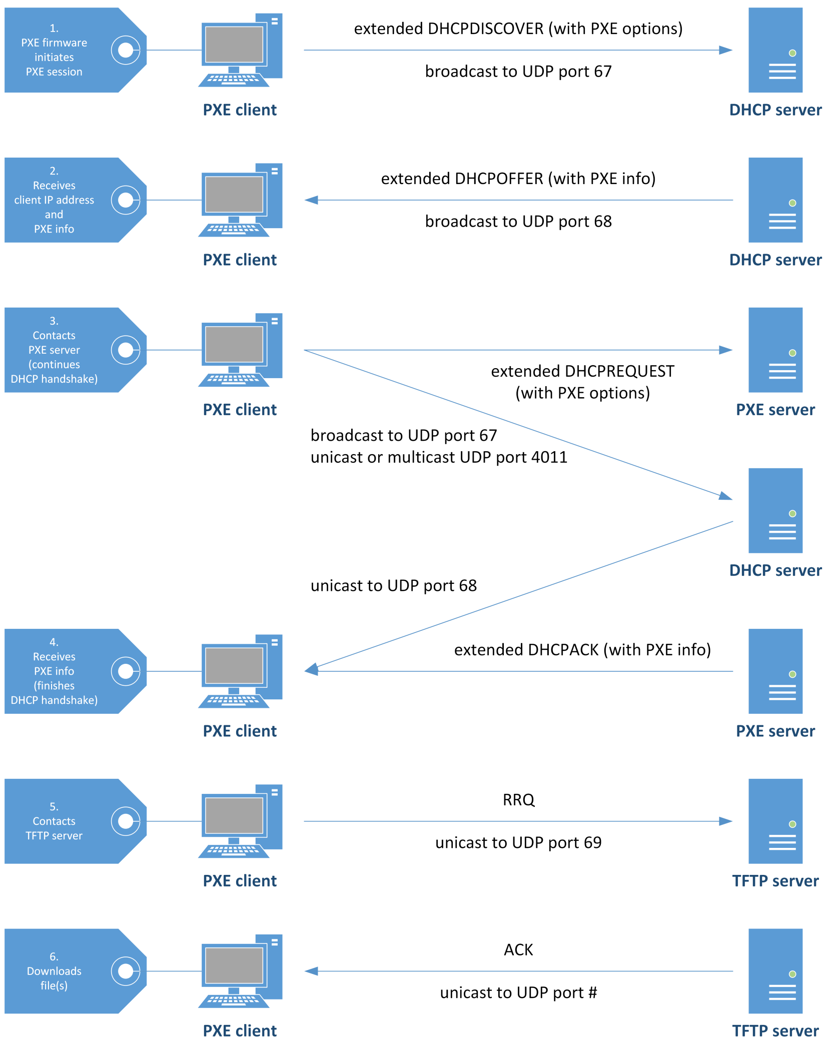

1. PXE firmware initiates PXE session
PXE client
extended DHCPDISCOVER (with PXE options)
broadcast to UDP port 67
DHCP server
2. Receives client IP address and PXE info
PXE client
extended DHCPOFFER (with PXE info)
broadcast to UDP port 68
DHCP server
3. Contacts PXE server (continues DHCP handshake)
PXE client
extended DHCPREQUEST (with PXE options)
PXE server
broadcast to UDP port 67
unicast or multicast UDP port 4011
DHCP server
unicast to UDP port 68
4. Receives PXE info (finishes DHCP handshake)
extended DHCPACK (with PXE info)
PXE client
PXE server
5. Contacts TFTP server
RRQ
unicast to UDP port 69
PXE client
TFTP server
6. Downloads file(s)
ACK
unicast to UDP port #
PXE client
TFTP server

Figure 1: PXE process

2.3 Network File System (NFS)

The Network File System (NFS) is a distributed file system, allowing access to files across a network in the same way local storage is accessed. The protocol is build on the Open Network Computing Remote Procedure Call (ONC RPC). The common port number used for NFS is 2049. It is an open standard, described in several RFCs (1094, 1813, 3530 and 5661).

The typical implementation of NFS consists of the following set-up:

1. The server runs the NFS service to make NFS shares available.
2. The system administrator determines which locations may be exported as NFS shares in the `/etc/exports` configuration file.
3. The system administrator determines which clients (by IP address or hostname) or subnet may access the shares in the same configuration file.
4. The client requests access to the NFS share by mounting the location.
5. The client can read and/or write the files as if it is local storage. [7] [8]

2.4 Internet Small Computer System Interface (iSCSI)

The internet Small Computer System Interface (iSCSI) is a standard for storage networking and also known as a Storage Area Network (SAN). It sends SCSI commands across a network in the same way as a local storage device. It can be used to distribute storage devices across a LAN or the internet as if they are locally attached disks. The common port number used for iSCSI is 3260. It is an open standard, described in several RFCs (3720 and 3721).

The implementation of iSCSI requires the following set-up:

Target The iSCSI target is the server side of an iSCSI implementation. It runs the iSCSI target service and distributes the storage device using a Logical Unit Number (LUN) and an addressing name (most of the time the iSCSI Qualified Name).

Initiator The iSCSI initiator is the client side of the iSCSI implementation. It discovers iSCSI targets and connects to the LUN and addressing name the same way as a local SCSI storage device does. [9] [10]

3 Research

3.1 Approach

To determine whether or not a bootable Linux CD / PXE can be build for the remote acquisition of multiple computers, research is done on the possible concepts. The concepts need the basics for the remote acquisition environment. These basics are provided by the sub research questions which determine the suitable Linux distribution, software and settings for the environment. The concepts are tested across a LAN and the internet. Further more, the concepts are tested on performance to appoint the better concept for future development after this research. Eventually the results determine whether or not the hypothesis of this research is correct and what is to expect for these concepts in the near future.

3.2 Concepts

Within a remote acquisition environment one is able to boot a computer (client or server) with a live image from a PXE (preferred), an USB-stick or optical disc. The image needs to be configurable according to network settings of IT infrastructure. The computer boots a Linux environment and starts a VPN connection to communicate securely with a (forensic) server. From this point the storage devices should be acquired on client-server bases.

The literature study for this research showed a concept used in a similar situation for triage. In the research of Martin B. Koopmans and Joshua I. James an automated network triage environment was described in which a client computer boots a preconfigured live image, loads key words from a NFS share on the server and starts to perform the triage. The discovered evidence is copied to the NFS share on the server for further investigation. [2] For this research two concepts were developed. The first one is based on the study with a NFS share on a server and the second one on the iSCSI standard.

3.2.1 NFS

This concept is based on the use of a NFS share on the server. The computer boots a live image from a PXE or an optical disc and loads the Linux environment. Within this environment it starts an OpenVPN connection to communicate securely with the server. When the connection is established, the acquisition starts automatically if preconfigured or manually by accessing the client itself. Both situations have their requirements. The automated acquisition requires the server to be online at boot and doesn't give any control of the acquisition process. The manual acquisition requires direct or remote access to each of the client to be acquired.

This concept is illustrated in figure 2.

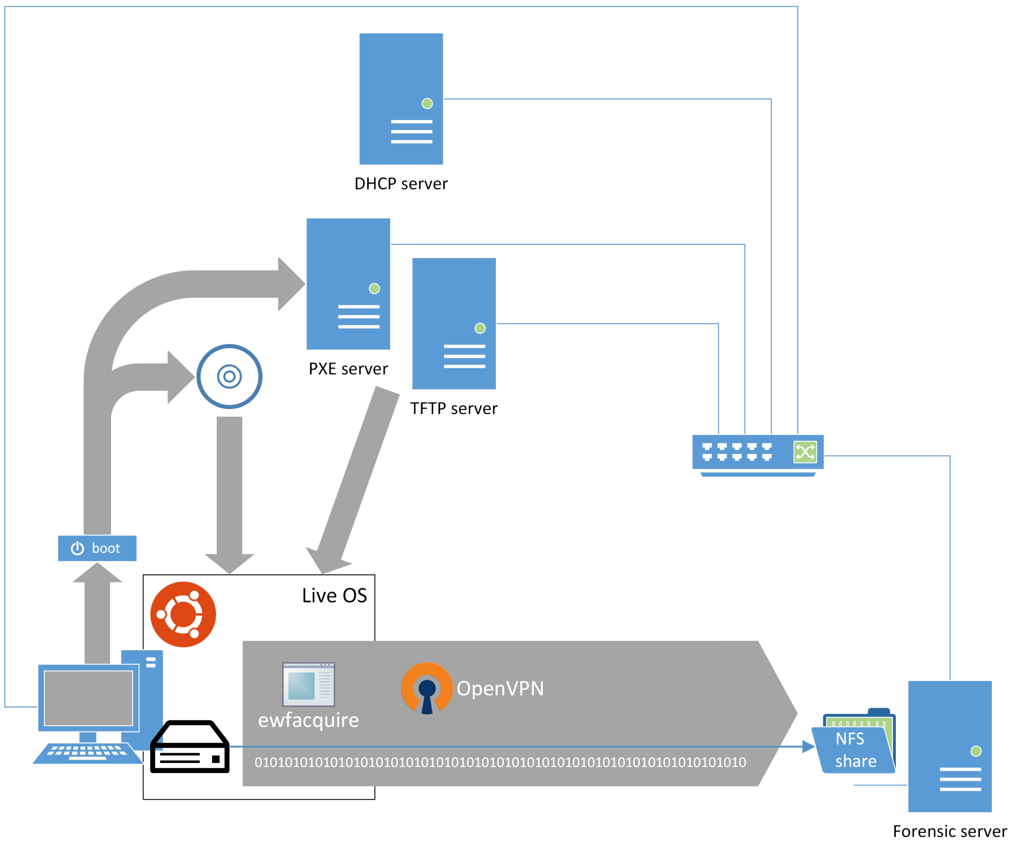


Figure 2: NFS concept

3.2.2 iSCSI

This concept is based on the use of iSCSI to distribute storage devices across a network to the server. The computer boots a live image from a PXE or an optical disc and loads the Linux environment. Within this environment it starts an OpenVPN connection to communicate securely with the server and distributes the storage devices. The acquisition of the storage devices can be started from the server at any time. The acquisition of multiple clients can be managed from a central location (the server) which is a huge advantage of the iSCSI concept.

This concept is illustrated in figure 3.

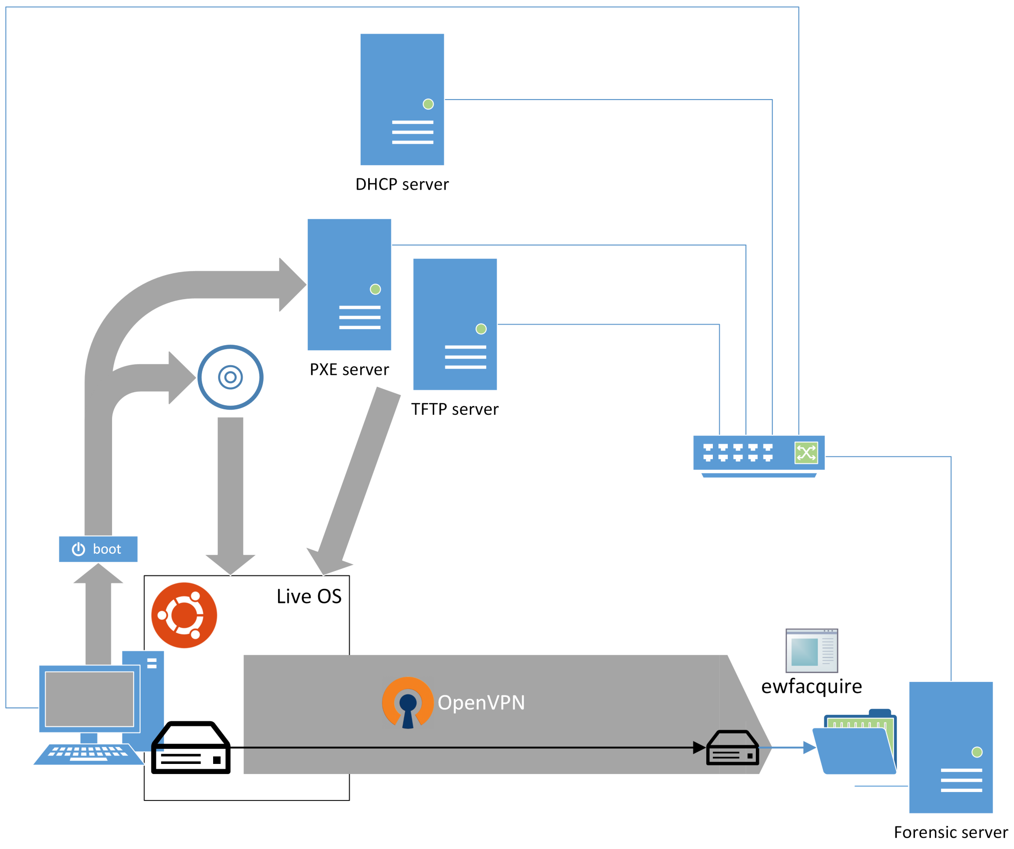


Figure 3: iSCSI concept

3.3 Proof of concept

In this research both concepts are developed for performance testing, but the focus will be on the iSCSI concept. As mentioned in section 3.2.2 this concept has a huge advantage on the NFS concept. Therefore the iSCSI concept will be developed into a proof of concept. This proof of concept will be called the Remote Acquisition Boot Environment (RABE).

3.4 Base environment

The base environment was the test environment for and during development. It consisted of a dedicated computer (also the server for the environment) and four virtual machines (VMs), each with a single task. This environment is illustrated in figure 4.

dedicated computer The dedicated computer runs a Windows operating system (Windows 8) and four virtual machines (VMware Workstation). This computer also served as the RABE server for the environment, running the OpenVPN and iSCSI initiator service.

VM1: RP2 - DHCP Server The virtual machine ran a Linux operating system (Ubuntu Server 14.04) and DHCP service. The VM was used to distribute network configuration settings (IP addresses and PXE parameters) and to test the PXE use of a RABE live image.

VM2: RP2 - PXE Server The virtual machine ran a Linux operating system (Ubuntu Server 14.04) and TFTP service. The VM was used to transfer boot files and to test the PXE use of a RABE live image (together with the DHCP server).

VM3: RP2 - iSCSI Client The virtual machine ran a Linux operating system (Ubuntu Desktop 14.04) and was configured to have three hard disk drives. The VM was used to boot the RABE live image with an optical disc, USB stick or a PXE. The hardware configuration of three hard disk drives (sda, sdb and sdc) was used to test the distribution of available storage devices on a RABE client.

VM4: RP2 - Ubuntu Remastering The virtual machine ran a Linux operating system (Ubuntu Desktop 14.04) and was configured to remaster Ubuntu live images. The VM was used to create the first live image through manually remastering the standard Ubuntu image and after the creation of the authoring to remaster and configure the RABE live images.

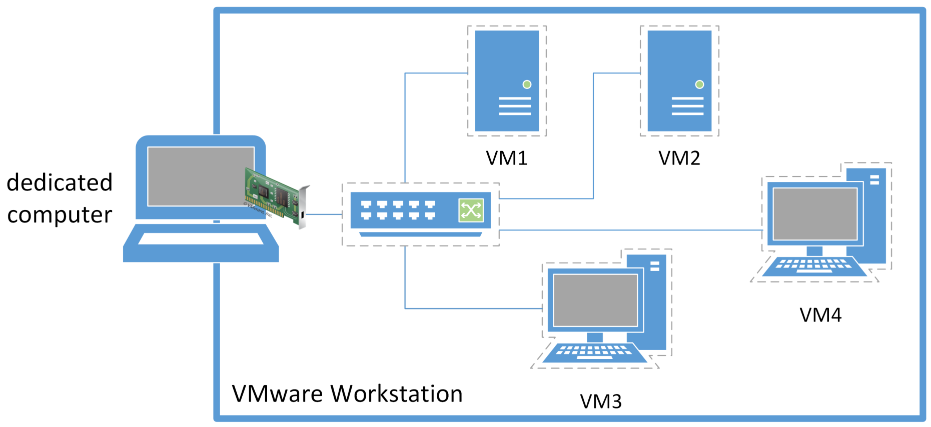


Figure 4: Base test environment

4 Implementation

4.1 Live image

The implementation started with the remastering of a standard Linux distribution live image to create a default RABE live image. At first KNOPPIX 7.2.0 was chosen as the distribution, because this is often used for forensic live images and has an easy remastering process. However, there where some problems with implementatoion of the `iscsitarget` package within KNOPPIX, so Ubuntu Desktop 14.04 was chosen as the distribution.

The concept needed to contain some packages and services for the live image in order to boot an automated client environment in which the following aspects were implemented:

1. setting the network configuration for the `eth0` interface
2. loading firewall rules to only permit communication with the server
3. starting a VPN connection to communicate securely with the server
4. distributing all storage devices across the network
5. sending a log file with the client's IP address, VPN IP address, iSCSI targets and disk (meta) information to the server

This resulted in an client environment as illustrated in figure 5.

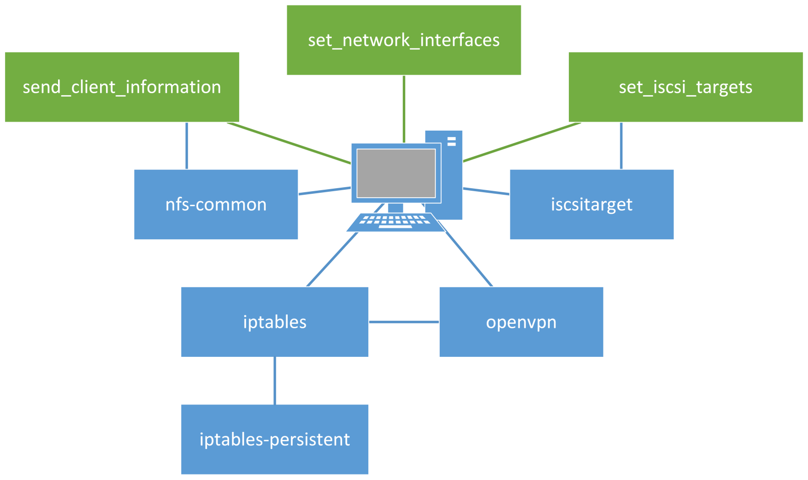


Figure 5: Client side implementation

4.1.1 Packages

The default Ubuntu Desktop 14.04 live image contained a lot of unnecessary packages, resulting in a live image of 970 MB. To reduce the size of the default live image 62 packages (including its dependencies) where purged. These where mostly GUI related packages which where discovered by research in the Ubuntu Software Center. A list of purged packages is mentioned in appendix A. For the mentioned aspects to work, the live image also needed four new packages to be installed. Eventually this led to a live image of 785 MB. The packages needed for the concept are described next.

nfs-common The `nfs-common` packages contains the basic programs for NFS. It includes the `mount.nfs` program which easily mount NFS shares. This is needed for the `send_client_information` service as described in section 4.1.2.

iptables The `iptables` package is a firewall program, installed by default in all Ubuntu distributions. It allows and blocks network IPv4 and IPv6 traffic. By default it allows all network traffic back and forth, but for the concept there is a need for the allowance of specific traffic and blocking the rest. The firewall rules are determined by the network settings of the IT infrastructure. These settings are configured with the authoring tool. The explicit firewall rules are mentioned in section 4.2. However, the firewall rules are not boot persistent. Booting a client won't automatically load the firewall rules and that's where the `iptables-persistent` packages comes in.

iptables-persistent The `iptables-persistent` package is an addition to the `iptables` packages. It starts where the `iptables` leaves off, being boot persistent. The packages runs as a service at boot and loads firewall rules from the configuration files located at `/etc/iptables/rules.v4` for IPv4 rules and `/etc/iptables/rules.v6` for IPv6 rules.

iscsitarget The `iscsitarget` package is the iSCSI Enterprise Target (IET) program. It is an open source iSCSI target program for distributing storage devices in an enterprise environment. [11] This packages is used for distributing storage devices as iSCSI targets across the network. The service needs to be enabled by setting `ISCSITARGET_ENABLE=true` in `/etc/default/iscsitarget` to be loaded at boot. The iSCSI targets need to be set manually in the configuration file located at `/etc/iet/ietd.conf`. [12] For the concept this configuration is set by the `set_iscsi_targets` service as described in section 4.1.2.

openvpn The openvpn package is the OpenVPN program. It is an open source SSL VPN program for creating a routed IP tunnel (TUN) or ethernet tunnel (TAP). The package contains the programs for client and server implementations. [13]. This package is used for starting the client side of the routed VPN tunnel (TUN) to communicate securely with the server. The automatic start of a specific VPN connection needs to be enabled by setting `IAUTOSTART="<name-of-vpn-connection>"` in `/etc/default/openvpn` to be loaded at boot. [14] The client configuration file for the concept is shown in figure 6. This figure contains only enabled settings, except for the `;comp-lzo` setting which is important. The settings are described and explained from a server perspective in section 4.3.1.

```
client

dev tun

proto tcp

remote 192.168.6.15 1194

resolv-retry infinite

nobind

persist-key
persist-tun

ca ca.crt
cert rabe_client.crt
key rabe_client.key

ns-cert-type server

cipher AES-256-CBC
keysize 256
auth SHA256

;comp-lzo

verb 3
```

Figure 6: OpenVPN client configuration file

4.1.2 Services

The live image needed new features that couldn't be provided by the installation of packages, because these features could only be achieved by scripting. These features are provided by newly created services which are described next.

set_network_interfaces The `set_network_interfaces` service configures the network settings of the interfaces at boot by replacing the default interfaces configuration file located at `/etc/network/interfaces` with a file located at `/etc/network/interfaces_new` which is created by the authoring (line 18 of figure 7). It brings the `eth0` interface down (line 13) before replacing the configuration file and brings the interface back up afterwards (line 23). Stopping the service will bring the `eth0` interface down (line 29). Restarting the service will bring the `eth0` interface down and back up (lines 35, 40). This service is needed, because removing the `network-manager` without completely removing the Ubuntu Desktop feature leaves a complicated configuration for network managing. Researching the complete removal of the Ubuntu Desktop feature could resolve this, but was beyond the scope of this research. As a workaround, this service was created.

set_iscsi_targets The `set_iscsi_targets` service searches for SCSI block devices and configures the devices as iSCSI targets in the `iscsitarget` configuration file located at `/etc/iet/ietd.conf` (lines 19-21, 26-27 of figure 8). The service extracts the MAC address of the `eth0` interface from the `ifconfig` command (line 11). It searches for SCSI block devices (sdX) on the client (line 19) and for each device an iSCSI target is configured, using the MAC address and block device as the addressing name (lines 26-27). An example configuration for an iSCSI target block device is shown in figure 9.

send_client_information The `send_client_information` service creates a log file with the network, iSCSI target and disk (meta) information of the client on the NFS share of the server. The service extracts the IP address of the `eth0` interface from the `ifconfig` command (line 19 of appendix B) and then extracts the VPN IP address of the `tun` interface also from the `ifconfig` command (line 22). It sets the log file's name and path (lines 28-29) and creates a log file with the IP address on the NFS share of the server (line 32). From that moment on it appends the VPN IP address (line 33), iSCSI targets with heading (lines 39-41) and disk (meta) information with heading (lines 47-49) to the log file. An example of the log file's content is shown in figure 10. The meta information of storage devices is not distributed with the iSCSI targets. However, it is very important for the forensic acquisition of a device. Information on the make (vendor), model (product), logical block device name, serial number (serial) and size of the device is needed to distinguish one storage device from another. This information is obtained by the `lshw` command which lists all hardware in a system and with a parameter (-c disk) only lists disk related hardware, including the meta information. Before extracting the IP addresses it mounts the NFS share on the server (line 14) and it unmounts the NFS share afterwards (line 54). The full script is (because of its size) mentioned in appendix B.

```
#!/bin/bash

# SERVICE: set_network_interfaces
# USAGE: start/stop/restart

case "$1" in
start)
        echo \[set_network_interfaces\] starting service

        echo \[set_network_interfaces\] bringing eth0 down

        # bring interface eth0 down
        ifdown eth0

        echo \[set_network_interfaces\] adding new network settings for
            eth0

        # copy new interfaces file and overwrite old
        cp /etc/network/interfaces_new /etc/network/interfaces

        echo \[set_network_interfaces\] bringing eth0 up

        # bring interface eth0 up
        ifup eth0
;;
stop)
        echo \[set_network_interfaces\] bringing eth0 down

        # bring interface eth0 down
        ifdown eth0
;;
restart)
        echo \[set_network_interfaces\] bringing eth0 down

        # bring interface eth0 down
        ifdown eth0

        echo \[set_network_interfaces\] bringing eth0 up

        # bring interface eth0 up
        ifup eth0
;;

*)
echo SERVICE: set_network_interfaces
echo $"USAGE: $0 {start/stop/restart}"
exit 1

esac
exit 0
```

Figure 7: set_network_interfaces service (code)

```
#!/bin/bash

# SERVICE: set_iscsi_targets
# USAGE: start

case "$1" in
start)
        echo \[set_iscsi_targets\] starting service

        # extract MAC address (eth0) from ifconfig
        MAC=$(ifconfig eth0 | grep -i 'hwaddr' | awk '{print $5}' | sed 's
            /://g')

        # set counter
        COUNTER=0

        echo \[set_iscsi_targets\] searching SCSI storage devices

        # search SCSI storage devices
        for SCSI in $(ls /dev/sd* | grep 'sd[a-z]$')
        do
                DISK=$(echo $SCSI | sed 's/\/dev\///g')

                echo \[set_iscsi_targets\] setting iSCSI target $MAC:$DISK

                # add target to configuration file
                echo Target $MAC:$DISK >> /etc/iet/ietd.conf
                echo Lun $COUNTER Path=$SCSI,Type=fileio >> /etc/iet/ietd.
                    conf

                # increase counter
                COUNTER=$[ COUNTER+1 ]
        done
;;

*)
echo SERVICE: set_iscsi_targets
echo $"USAGE: $0 {start}"
exit 1

esac
exit 0
```

Figure 8: set_iscsi_targets service (code)

```
Target b8ac6f8b81bd:sda
Lun 0 Path=$SCSI,Type=fileio
```

Figure 9: iSCSI target block device configuration

```
IP ADDRESS: 192.168.10.15
VPN IP ADDRESS: 10.8.0.6

iSCSI TARGETS:
==============
b8ac6f8b81bd:sda
b8ac6f8b81bd:sdb

DISK INFORMATION:
=================
  *-disk
   description: ATA Disk
   product: ST3250824AS
   vendor: Seagate
   physical id: 0.0.0
   bus info: scsi@0:0.0.0
   logical name: /dev/sda
   version: 3.AD
   serial: 9ND0CZDL
   size: 232GiB (250GB)
   capabilities: partitioned partitioned:dos
   configuration: ansiversion=5 sectorsize=512 signature=8d4b79a1
  *-disk
   description: SCSI Disk
   physical id: 0.0.0
   bus info: scsi@6:0.0.0
   logical name: /dev/sdb
   size: 29GiB (31GB)
   capabilities: partitioned partitioned:dos
   configuration: sectorsize=512 signature=e2ed4f7e
```

Figure 10: Log file content

For the concept to work correctly with other services (provided by packages) these had to be started in a specific order. It is very important to set the network interfaces correctly from the beginning, because a lot of other services depend on the configuration. So the `set_network_interfaces` service has to be started first and therefore should start as service 14 in `/etc/rc2.d` as shown in figure 11. The next service limits the network connectivity (only allowing communication with the server) by loading the firewall rules with the `iptables-persistent` service (number 15). Then it starts the `openvpn` service that highly depends on the network configuration (number 16). For the `iscsitarget` service it is important to first set the iSCSI targets and then loads the service that distributes the targets across the network. Therefore the `set_iscsi_target` service is set to number 19 and the `iscsitarget` service is set to number 20. The `send_client_information` service depends on all the mentioned services and should be started later (number 30).

```
root@uburem:/# ls -l /etc/rc2.d
...
lrwxrwxrwx 1 root root  32 Jul  3 09:40 S14set_network_interfaces -> ../init.d/set_net
lrwxrwxrwx 1 root root  29 Jul  3 09:40 S15iptables-persistent -> ../init.d/iptables-p
lrwxrwxrwx 1 root root  17 Jul  3 09:40 S16openvpn -> ../init.d/openvpn
lrwxrwxrwx 1 root root  27 Jul  3 09:40 S19set_iscsi_targets -> ../init.d/set_iscsi_ta
lrwxrwxrwx 1 root root  21 Jul  3 09:40 S20iscsitarget -> ../init.d/iscsitarget
...
lrwxrwxrwx 1 root root  33 Jul  3 09:40 S30send_client_information -> ../init.d/send_c
...
lrwxrwxrwx 1 root root  18 Jul  3 09:40 S99rc.local -> ../init.d/rc.local
```

Figure 11: Auto-start order of services

4.2 Authoring tool

Remastering a live image can be a complicated process, because many things can be added and changed. This is a manual process consisting a lot of steps with specific commands, but not all of them are required for every situation. The necessary steps where discovered by research on the remastering of an Ubuntu live image. [15] It led to the following steps for the concept.

1. mount the input image (line 39 of appendix C)
2. extract the image contents (line 43)
3. extract the squash file system (line 47)
4. copy the network settings for network connectivity within the `chroot` environment (lines 51-52)
5. mount the device files for the `chroot` environment (line 59)
6. enter the `chroot` environment (line 66)
7. regenerate the desktop manifest file (lines 77-80)
8. compress the squash file system (line 84)
9. update the file system size file (line 87)
10. recalculate and update the MD5 checksum of the files (line 91)
11. create the new live image (line 94)

These steps were scripted into an authoring tool with a default live image (with all necessary packages, services and files) as the input image to automate and simplify the remastering process for the user. However, there was an issue with executing commands within the `chroot` environment. Entering the `chroot` environment paused the initial script and continued after manually exiting. For the remastering process to complete successfully, some additional steps were needed within the `chroot` environment. The process, system and pseudo-terminal file systems had to be mounted (lines 4-6 of figure 12), a couple of local variables had to be exported (lines 9-10) and some installation settings had to be configured (lines 13-15). At the end, temporary files had to be removed (lines 4, 7 10-12, 20-21 of figure 13) and the file systems had to be unmounted (lines 15-17). Researching the scripted execution in a `chroot` environment could resolve this issue, but was beyond the scope of this research. As a workaround, a start and an end script were created which are copied into the `chroot` environment (line 55-56) and need to be executed manually.

```
#!/bin/bash

# mount proc, sysfs and devpts in chroot
mount -t proc none /proc
mount -t sysfs none /sys
mount -t devpts none /dev/pts

# set locale variables
export HOME=/root
export LC_ALL=C

# set install settings
dbus-uuidgen > /var/lib/dbus/machine-id
dpkg-divert --local --rename --add /sbin/initctl
ln -s /bin/true /sbin/initctl

echo ""
echo Type ./end to end working in the chroot environment.
echo ""
```

Figure 12: Chroot start script (code)

```
#!/bin/bash

# remove no longer needed packages
apt-get autoremove

# remove local repository packages
apt-get clean

# unset install settings
rm /var/lib/dbus/machine-id
rm /sbin/initctl
dpkg-divert --rename --remove /sbin/initctl

# unmount proc, sysfs and devpts in chroot
umount /proc || umount -lf /proc
umount /sys || umount -lf /sysfs
umount /dev/pts || umount -lf /dev/pts

# remove history
rm -rf /tmp/* ~/.bash_history
history -c

echo ""
echo Type exit to exit the chroot environment.
echo ""
```

Figure 13: Chroot end script (code)

The first version of the authoring tool (0.1) only contains the basic steps. It was used for developing the default live image. The full version 0.1 script is (because of its size) mentioned in appendix C. The latest version of authoring tool (0.4) contains user input and specific configuration aspects of the live image and these are described next. The full version 0.4 script is (because of its even greater size) mentioned in appendix D.

Network settings The code related to network settings configures the static IP address, netmask and gateway on the `eth0` interface, according to the user input, to the `etc/network/interfaces_new` file (lines 173-176 of appendix D). This file is used to replace the `etc/network/interfaces` file at boot by the `set_network_interfaces` service. If the infrastructure doesn't requires a static configuration it will configure DHCP on the interface (line 179).

Firewall rules The code related to firewall rules configures rules, depending on to the user input. If the infrastructure doesn't require a static configuration, it will configure a firewall rule which allows DHCP traffic (line 182). The server IP address from the user input is configured in a way can only communicate with the server (line 186). The default VPN IP address of the server is configured in a way that it can communicate with the server across the VPN tunnel (line 199). All other traffic is blocked (line 213). All rules are added to `/etc/iptables/rules.v4` which are loaded at boot by the `iptables-persistent` service.

OpenVPN The code related to OpenVPN requires the server CA certificate (ca.crt) and key (ca.key), the index (index.txt) and serial to reside in the authoring tool's `openvpn/keys` folder. This is required because the authoring tool will create a client certificate and key for the VPN connection. It will copy the necessary `easy-rsa` files to the `openvpn` folder (lines 90-94) and generates the client certificate and key (line 101). Then the server CA certificate and the client certificate and key are copied into the `chroot` environment (lines 191-193). The server IP address from the user input replaces the SERVER space in the client configuration file (`/etc/openvpn/rabe_client.conf`) within the `chroot` environment (line 196). Eventually, the `openvpn` folder is cleaned by removing temporary files (lines 202-206, 209).

NFS The code related to NFS asks the user for an alternate NFS path (lines 115-116). The NFS path is then escaped to prevent path errors (line 122) and otherwise set to a default path (line 124). Later on, the NFS path in the `send_client_information` service file is replaced by the user input or default NFS path (line 217).

4.2.1 Dependencies

The authoring tool needs specific programs within the script and has to following packages dependencies.

easy-rsa For generating and signing the OpenVPN client certificate and key.
genisoimage For creating the new live image ISO file.
squashfs-tools For extracting and compressing the squash file system of the live image.

4.3 Server

The server implementation was initially not in the scope of this research, but for the proof of concept a minimum server side was needed. For the server implementation the Ubuntu Desktop 14.04 operating system was chosen. The server had to contain the following aspects:

1. provide the VPN service for the secure communication with the client
2. connect and manage iSCSI targets
3. provide a NFS share for the log file of the client
4. provide a simple web service to overview the log files (connected clients)

This resulted in a server environment as illustrated in figure 14.

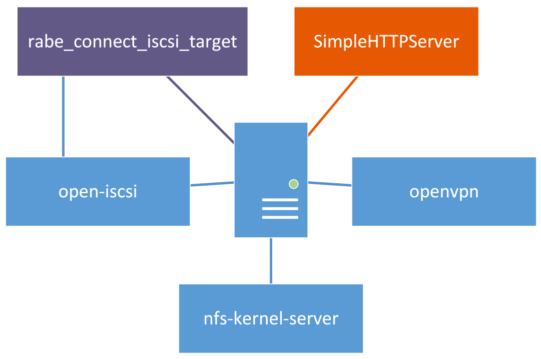


Figure 14: Server side implementation

4.3.1 Packages

For the aspects to work, three packages had to be installed and are described next.

nfs-kernel-server The `nfs-kernel-server` package contains the NFS server program. This package is used for providing the NFS share to the client. The NFS share is a folder on the server which is exported as a NFS share. The folder is configured in `/etc/exports`, so it can only be accessed through the VPN tunnel (secure connection).

open-iscsi The `open-iscsi` package is an open source iSCSI initiator program. It includes the `iscsiadm` program for discovering and connecting iSCSI targets. [16] The iSCSI initiator service needs to be enabled by uncommenting `node.startup = automatic` in `/etc/iscsi/iscsid.conf` to be loaded at boot. [**?**] For the concept, the (simple) management of iSCSI targets is provided by the `rabe_connect_iscsi_target` script as described in section 4.3.3.

openvpn The `openvpn` package is the OpenVPN program. It is an open source SSL VPN program for creating a routed IP tunnel (TUN) or ethernet tunnel (TAP). The package contains the programs for client and server implementations. [13]. This package is used for starting the server side of the routed VPN tunnel (TUN) to communicate securely with the client. The automatic start of a specific VPN connection needs to be enabled by setting `IAUTOSTART="<name-of-vpn-connection>"` in `/etc/default/openvpn` to be loaded at boot. [14] The server configuration file for the concept is shown in figure 15. This figure contains only enabled settings, except for the `;comp-lzo` setting which is important.

```
port 1194

proto tcp

dev tun

ca ca.crt
cert rabe_server.crt
key rabe_server.key

dh dh2048.pem

server 10.8.0.0 255.255.255.0

ifconfig-pool-persist ipp.txt

duplicate-cn

keepalive 10 120

cipher AES-256-CBC
keysize 256
auth SHA256

;comp-lzo

persist-key
persist-tun

status openvpn-status.log

log-append openvpn.log

verb 3
```

Figure 15: OpenVPN server configuration file

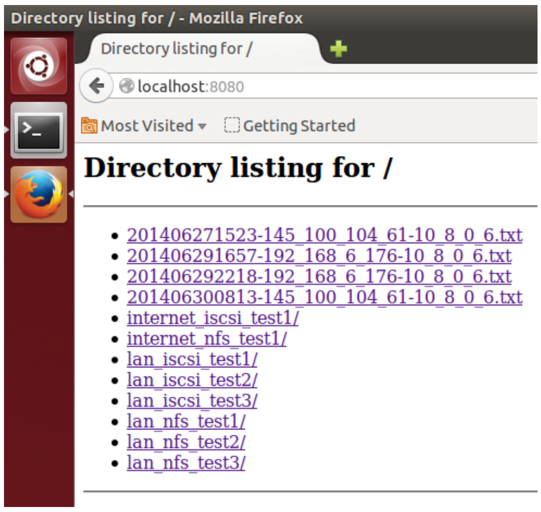
The settings in figure 15 are described next.

port The port to listen on (1194).
proto The protocol to use (tcp).
dev The type of tunnel to use (tun).
ca The root certificate (ca.crt).
cert The server certificate (rabe_server.crt).
key The server private key (rabe_server.key).
dh The Diffie Hellman parameters (dh2048.pem).
server The VPN IP range and subnet (10.8.0.0 255.255.255.0).
duplicate-cn Allow client certificates with the same 'common name' for multiple clients using the live image.
keepalive The keep alive ping settings to check the connection, repetition and maximum timeout in seconds (10 120).
cipher The cryptographic cipher to use (AES-256-CBC).
keysize The key-size to use, in bits (256).

auth The algorithm to use (SHA256).
;comp-lzo Disable compression across the connection.
persist-key Disable re-reading of key files.
persist-tun Disable closing and reopening of TUN/TAP devices.
status Enable outputting the status of current connections to a log file (openvpn-status.log).
log-append Enable append logging to a separate log file (openvpn.log).
verb Log level, 0 nothing and 9 maximum (3).

4.3.2 Web service

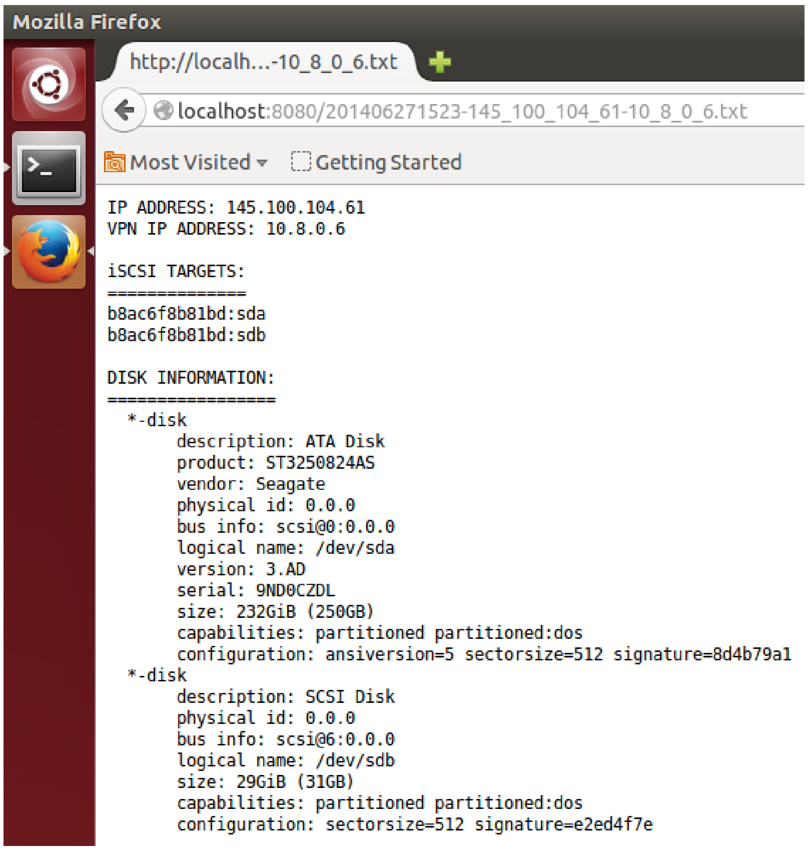
A web server was used to have an overview of all connected clients with their IP address, VPN IP address, iSCSI targets and disk (meta) information. The web server functionality was provided by the `SimpleHTTPServer` class in python. It allows for a folder to serve as the root folder of a web server and is started by using the `python -m SimpleHTTPServer <port-to-server-on>` command. The SimpleHTTPServer ran in the NFS root folder of the server, where all clients create their log file. To get the overview, one simply have to browse to the IP address and port with a web browser. The web browser shows all log files of connected clients (figure 16) and within each log file the information of the client (figure 17). This information is needed for connecting to and acquiring the storage device on that client.



Directory listing for /

- 201406271523-145_100_104_61-10_8_0_6.txt
- 201406291657-192_168_6_176-10_8_0_6.txt
- 201406292218-192_168_6_176-10_8_0_6.txt
- 201406300813-145_100_104_61-10_8_0_6.txt
- internet_iscsi_test1/
- internet_nfs_test1/
- lan_iscsi_test1/
- lan_iscsi_test2/
- lan_iscsi_test3/
- lan_nfs_test1/
- lan_nfs_test2/
- lan_nfs_test3/

Figure 16: Overview of connected clients (Firefox web browser)


Mozilla Firefox
http://localh...-10_8_0_6.txt
localhost:8080/201406271523-145_100_104_61-10_8_0_6.txt
Most Visited
Getting Started

```
IP ADDRESS: 145.100.104.61
VPN IP ADDRESS: 10.8.0.6

iSCSI TARGETS:
==============
b8ac6f8b81bd:sda
b8ac6f8b81bd:sdb

DISK INFORMATION:
=================
  *-disk
       description: ATA Disk
       product: ST3250824AS
       vendor: Seagate
       physical id: 0.0.0
       bus info: scsi@0:0.0.0
       logical name: /dev/sda
       version: 3.AD
       serial: 9ND0CZDL
       size: 232GiB (250GB)
       capabilities: partitioned partitioned:dos
       configuration: ansiversion=5 sectorsize=512 signature=8d4b79a1
  *-disk
       description: SCSI Disk
       physical id: 0.0.0
       bus info: scsi@6:0.0.0
       logical name: /dev/sdb
       size: 29GiB (31GB)
       capabilities: partitioned partitioned:dos
       configuration: sectorsize=512 signature=e2ed4f7e
```

Figure 17: Network, iSCSI and disk information of client (Firefox web browser)

4.3.3 Connect script

The `rabe_connect_iscsi_target` script asks the user for the VPN IP address of the client. The script uses the `iscsiadm` program to discover the iSCSI targets on the client and shows the targets to the user, filtering on the VPN IP address range (line 18). The filtering is applied because a discovery will return all IP addresses to which the iSCSI target can be connected. For this concept it needs to use a secure connection and thus needs to use the VPN IP address. The script asks the user to specify the iSCSI target by addressing name. The `iscsiadm` program is then used to connect to the iSCSI target (line 28). After that, it needs to query the message buffer of the kernel (`dmesg`) to determine the associated block device on the server and show it to the user (line 33).

```
#!/bin/bash

# set regular expression pattern for IP address
IPPATTERN="^10\.8\.0\.[1-2]?[0-9]?[0-9]{1}$"

echo ""

# set IP address of client
until [[ $IPADDRESS =~ $IPPATTERN ]]
do
        echo Enter the VPN IP address of the client \[10.8.0.###\]:
        read TARGETIP
done

# discover iSCSI target(s) on client
echo ""
echo Discovering iSCSI targets on client ...
iscsiadm -m discovery -t st -p $TARGETIP | grep '10.8.0'

# set iSCSI target
echo ""
echo Enter the name of the iSCSI target:
read TARGET

# connect to iSCSI target:
echo ""
echo Connecting to target $TARGET on $TARGETIP ...
iscsiadm -m node -p $TARGETIP -T $TARGET -l

# show target's storage device name on server
echo ""
echo Target $TARGET connected to:
tail dmesg | grep 'iSCSI'
```

Figure 18: rabe_connect_iscsi_target script (code)

4.4 Limitations

This proof of concept at this stage has three important limitations:

auto-mount not disabled The auto-mount function of storage devices is not disabled in the live image and can't guarantee the forensic soundness of the acquisition process. Therefore the concept can't be used in forensic cases yet!

listens only on interface eth0 The live image is configured to use the `eth0` interface and does not use other interfaces. It actually blocks traffic on all other interfaces and thus requires a connection on the `eth0` interface for the concept to work.

management network not blocked The live image blocks traffic on all other interfaces by using `iptables`. It doesn't take into account blocking a possible present management interface and thus leaves an opportunity for the owner to remotely connect to the computer.

5 Testing

5.1 Environments

For the testing part of this research, two different environments were used:

1. LAN
2. Internet

5.1.1 LAN

A corporate LAN was used for testing the RABE live image and the performance of NFS and iSCSI across a LAN. The RABE server was connected to the network and a random computer within the network was used as the RABE client. A simplified view of the LAN environment is illustrated in figure 19.

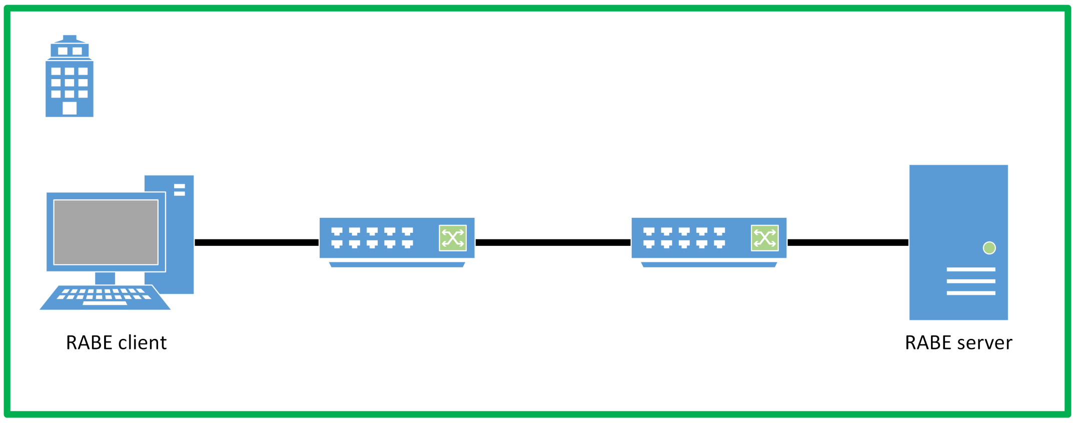


Figure 19: LAN test environment (simplified)

5.1.2 Internet

For testing the RABE live image and the performance of NFS and iSCSI across the internet, the corporate network and network of the University of Amsterdam (UvA) were used. The RABE server was connected to the corporate network with port forwarding configured for SSH (22) and OpenVPN (1194). The RABE client was my personal server at the UvA. It was configured with a private IP address which directly connects the client to the internet. A simplified view of internet environment is illustrated in figure 20.

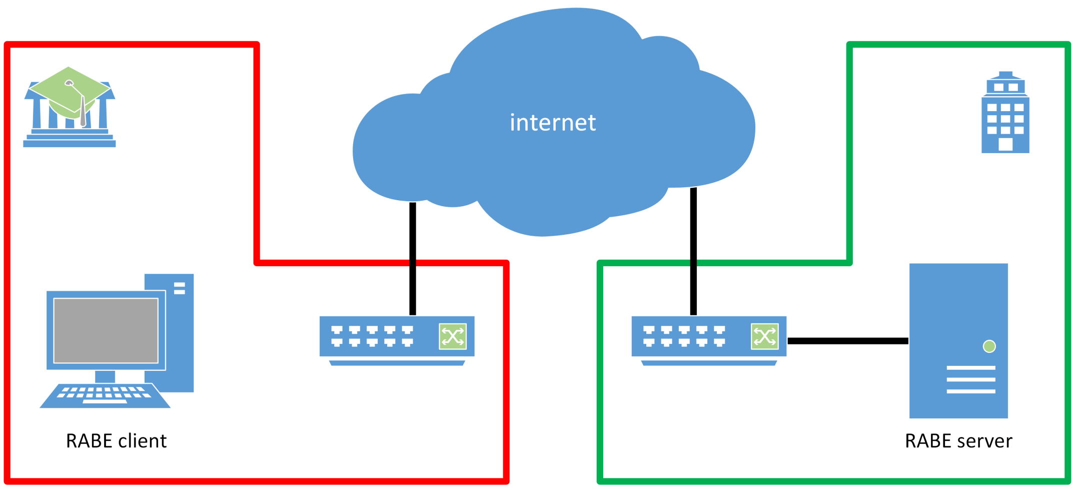


Figure 20: Internet test environment (simplified)

5.2 Tests

Both the NFS and iSCSI concept were tested. For each concept four grey box tests were defined which tested the proof of concept for both code and outcome. The performance of NFS and iSCSI was tested for outcome only.

5.2.1 NFS

NFS test 1 A full test of the live image within the LAN environment. The image is booted from an USB stick. The `eth0` interface is configured and firewall rules are loaded to restrict traffic to the client. An OpenVPN connection is started between the client and server to communicate securely. The client is connected to the NFS share on the server through the OpenVPN tunnel `tun0`. The clients sent its network and disk information to the server. The acquisition of the storage device is started with `ewfacquire`. The terminal output is logged with `script`.

NFS test 2 A full test of the live image within the LAN environment. The image is booted from an USB stick. The `eth0` interface is configured and firewall rules are loaded to restrict traffic to the client. The OpenVPN service is stopped on the server, so no VPN tunnel is available. The client is connected to the NFS share on the public IP address. The client sent its network and disk information to the server. The acquisition of the storage device is started with `ewfacquire`. The terminal output is logged with `script`.

NFS test 3 A full test of the live image within the LAN environment and the same as NFS test 1.

NFS test 4 A full test of the live image, but in this case within the internet environment.

5.2.2 iSCSI

iSCSI test 1 A full test of the live image within the LAN environment. The image is booted from an USB stick. The `eth0` interface is configured and firewall rules are loaded to restrict traffic to the client. An OpenVPN connection is started between the client and server to communicate securely. The storage devices are distributed across the secure connection as iSCSI targets. The client sent its network and disk information to the server. The server is connected to the iSCSI target through the OpenVPN tunnel `tun0`. The acquisition of the storage device is started with `ewfacquire`. The terminal output is logged with `script`.

iSCSI test 2 A full test of the live image within the LAN environment. The image is booted from an USB stick. The `eth0` interface is configured and firewall rules are loaded to restrict traffic to the client. The OpenVPN service is stopped on the server, so no VPN tunnel is available. The storage devices are distributed across the secure connection as iSCSI targets. The clients sent its network and disk information to the server. The server is connected to the iSCSI target on the public IP address. The acquisition of the storage device is started with `ewfacquire`. The terminal output is logged with `script`.

iSCSI test 3 A full test of the live image within the LAN environment and the same as iSCSI test 1.

iSCSI test 4 A full test of the live image, but in this case within the internet environment.

5.3 Results

In all of the tests the RABE live image worked correctly. The interface was configured and firewall rules were loaded. The (secure) connections where made and the image provided the server with the network and disk information of the client. The `ewfacquire` application provided the end results for the acquisition process. An example as such is shown in figure 21.

```
Written: 9.3 GiB(10000000188 bytes) in 17 minute(s) and 0 second(s) with 9.3MiB/s
(9803921 bytes/second).
MD5 hash calculated over data:d1bac32b46721780b314f170058e6db5
ewfacquire: SUCCESS
```

Figure 21: ewfacquire end result

The results of two of the first three NFS tests show an average speed of 9,3 MiB/s when remotely acquiring within a LAN environment (NFS test 1 and 3). NFS test 2 shows an average speed of 10 MiB/s which is slightly faster (by 0,7 MiB/s). This is due the fact that the VPN connection is disabled. A VPN connection requires more data in the network packet, leaving less space for actual (acquisition) data and resulting in a slower copying speed. These results are shown in table 1.

test	**time**	**speed MiB/s**	**MD5 hash**	**size GiB**	**size in bytes**
NFS 1	00:17:00	9,3	d1bac32b46721780b314f170058e6db5	9,3	10000000000
NFS 2	00:15:38	10	d1bac32b46721780b314f170058e6db5	9,3	10000000000
NFS 3	00:17:04	9,3	d1bac32b46721780b314f170058e6db5	9,3	10000000000

Table 1: NFS tests (LAN environment)

The results of two of the first three iSCSI tests show the same trend as the NFS tests, except with a slightly better performance in speed. iSCSI test 1 and 3 show an average speed of 10 MiB/s. iSCSI test 2 shows an average speed of 11 MiB/s which is also slightly higher (by 1 MiB/s). This is due the same cause as NFS test 2. These results are shown in table 2.

test	**time**	**speed MiB/s**	**MD5 hash**	**size GiB**	**size in bytes**
iSCSI 1	00:15:30	10	d1bac32b46721780b314f170058e6db5	9,3	10000000000
iSCSI 2	00:14:15	11	d1bac32b46721780b314f170058e6db5	9,3	10000000000
iSCSI 3	00:15:30	10	d1bac32b46721780b314f170058e6db5	9,3	10000000000

Table 2: iSCSI tests (LAN environment)

The overall maximum speed of 10-11 MiB/s is caused by the corporate network's speed limitation.

The results of the tests within the internet environment show an average speed of 1.1 MiB/s. A test without the OpenVPN connection was not conducted, due to the security aspect (sending data unencrypted across the internet) and time issues (a completion time of more than 2 hours). The different MD5 hashes of the data are the result of booting the original operating system of the client, making changes to the disk. The MD5 hashes from the LAN and internet tests are different, due to the use of different client computers. These results are shown in table 3.

test	**time**	**speed MiB/s**	**MD5 hash**	**size GiB**	**size in bytes**
NFS 4	02:13:39	1,1	0c27b2131c240fa88ceeab132ca326d0	9,3	10000000000
iSCSI 4	02:22:06	1,1	d1b749285de3e6ec69537fb1212b4dd0	9,3	10000000000

Table 3: NFS and iSCSI tests (internet environment)

The overall maximum speed of 1.1 MiB/s is caused by the maximum download speed of corporate network's internet connection.

6 Conclusion

6.1 General

The main research question was:

Can a bootable Linux CD / PXE be build for the remote acquisition of multiple computers and how does it perform compared to the traditional method?

The assumption/hypothesis for this research was:
The remote acquisition of multiple computers (in general) is slower then the traditional method and across the internet it is slower then across a LAN. However, if the acquisition is performed remotely without being on location, it can be done parallel to other activities. This could make it a time efficient solution for partial and sparse acquisition in the future.

This research proved the hypothesis to be correct, only with some side notes. The remote acquisition of multiple computers can be performed across a LAN or the internet. An optical disk, USB stick or PXE can be used to boot the a Linux environment. A set of firewall rules and a VPN connection provide the necessary secure connection, especially when performing the acquisition across the internet. All these aspects are assembled in the Remote Acquisition Boot Environment (RABE) based on the iSCSI concept. The competitive concept is NFS which is already used as a triage tool. Testing and comparing both concepts, showed a slightly faster performance for the iSCSI concept (by 0,7 MiB/s). With that in mind continuing the development of RABE on the iSCSI concept is the best choice.

As assumed in the hypothesis, the acquisition within a LAN showed a faster performance (by 8,2 MiB/s). However, there are some side notes. The tests within a LAN and across the internet had their limitations. The 10 MiB/s for the LAN and 1.1 MiB/s for the internet, where technical limitations. The LAN limitation was caused by the corporate network's maximum speed and the internet limitation was caused by the corporate network's maximum download speed of the internet connection. Consumer internet connections can reach 200 Mbps. Data centre and university connections can reach 10 Gbps and beyond. So in theory it can perform faster. Eventually it is the weakest link that determines the connection speed. Not every law enforcement agency has a high speed internet connection and therefore these theoretical speeds won't be matched, still needing the faster traditional method.

In general, the remote acquisition of a computer is slower than the traditional acquisition method of disassembling the storage devices. Instead of going to the scene, one can just provide the live image to the client side and start the remote acquisition from ones central location and return to other activities. Although it will not (yet) succeed the completion time of the traditional method, this concept can be a time efficient solution for the partial acquisition of a computer. The partial acquisition is a time efficient solution when one only needs a specific partition or folder on a storage device. It can also be a solution for triage with sparse acquisition. In large IT infrastructures there is often a need to determine which computer has to be seized. Acquiring all computers simply can't be an option. Sparse acquisition with copy-on-read will support the triage process to justify the seized or not seized computers. The previewed data is red and acquired, therefore one can always justify the decision for taking or leaving a computer. In this case the acquisition time is much shorter and highly efficient.

6.2 Achievements

created a proof of concept This research created a proof of concept for the remote acquisition of multiple computers based on iSCSI, called the Remote Acquisition Boot Environment (RABE). The concept consists of a default RABE live image for the client, an authoring tool for configuring RABE live images and a basic configuration for the server.

created a default live image This research created the default RABE live image. The image consists of all necessary packages and three newly created services for an automated process of remotely acquiring multiple computers. Some unnecessary packages have been purged to reduce the image size. It can be booted from an optical disc, USB stick or a PXE. After booting the client with the live image, it loads the needed firewall rules and starts a VPN connection to communicate securely with the server. It then distributes the client's storage devices across the network using iSCSI.

created an authoring tool This research created an authoring tool to configure RABE live images. This tool provided a simple user interface for configuring the live image with specific settings for the network, VPN connection and NFS share. Besides this specific purpose for RABE, it also includes the code for the automated remastering of an Ubuntu live image which can be used by others.

provided a partial conclusion on NFS vs. iSCSI This research provided test results for the use of NFS and iSCSI in a concept for the remote acquisition of multiple computers. The tests showed a slightly faster performance in favour of iSCSI.

created an open framework for future research This research created an open platform for future research. The proof of concept provides the key aspects to perform a real acquisition and a starting point for future research. It will be published on GitHub as the 'rabe-framework' project.

6.3 Future research

This report offers an open framework for future research as mentioned in section 6.2. During this research a couple of aspects were discovered that could help to improve the concept on various subjects.

6.3.1 Live image

The RABE live image in this research is a concept and far from final. It can be improved by the following subjects:

auto-mounting devices The used Ubuntu Desktop 14.04 image has an aggressive manner of automatically mounting storage devices. Despite all known methods of disabling the auto-mount function, the live image doesn't have auto-mount disabled. A lot of the known methods have been tried, but none with success. Additional research could solve this matter.

other network interfaces The live image is configured to only use the `eth0` interface and blocks traffic on all other interfaces. On computers with multiple network interfaces one doesn't no which network interface is used and in such situations the live images should still work. Additional research could solve this matter and improve the live image.

remove GUI / reduce size The live image still contains a GUI, because it uses a standard Ubuntu Desktop 14.04 image. Most of the GUI applications have been purged after basic research. The GUI isn't needed for the concept and can be removed, but removing the wrong packages can leave the image inoperable. Additional research could give new insides in fully removing the Ubuntu Desktop GUI and this would also decrease the image size considerably.

6.3.2 Authoring tool

The RABE authoring tool in this research is a straight forward bash script, but for the remastering process the `chroot` environment is needed. The bash script can't hop into the `chroot` environment to execute code and therefore pauses the initial script until manually exiting the environment. Additional research could solve this matter and improve the authoring tool.

6.3.3 Performance testing

The RABE concept has been tested on NFS and iSCSI for performance, but also on speed for LAN and internet connections. As mentioned in section 6.1 these tests had some side notes, because they were limited by the network or internet connection speed. More testing on different networks and internet connections (cable, xDSL and fibreglass) could give new insides in the feasibility of using remote acquisition as a successor of the traditional acquisition method.

6.3.4 Forensics

From a forensic perspective the RABE concept can also be improved.

auto-mounting devices As mentioned in section 6.3.1, this could increase the forensic soundness of the live image considerably and could make it usable in forensic cases.

block management network The live image blocks traffic by using `iptables`, but doesn't take into account blocking a possible present management interface which leaves the opportunity for the owner to remotely connect to the computer. Additional research into methods to block such interfaces could increase the forensic soundness of the live image.

reduce memory footprint / include memory acquisition The live image resides fully in memory and overwrites memory space, depending on the available amount. In digital forensics the acquisition of the memory has become a very useful and important subject. A subject that has to be taken into account before acquiring the storage devices in a computer. Additional research into the feasibility of reducing the memory footprint of the live image, could provide the image with a lesser footprint and the acquisition of the memory. This would make RABE a complete acquisition environment for digital forensics.

other tools The image has been stripped to contain only the necessary packages for the concept. As a live environment for digital forensics other tools could be added. This would make RABE a more complete acquisition environment for digital forensics.

preview/triage mode The image was tested for the full acquisition of storage devices. However, it can also be a solution for partial and sparse acquisition or triage. Additional research could give better inside in these subjects and integrate those in RABE. This includes integrating the research of Eric van den Haak about sparse acquisition with copy-on-read.

References

[1] Owen OConnor,
Deploying forensic tools via PXE,
http://www.sciencedirect.com/science/article/pii/S1742287604000581,
2004,
article.

[2] Martin B. Koopmans and Joshua I. James,
Automated network triage,
http://www.sciencedirect.com/science/article/pii/S1742287613000273,
2013,
paper.

[3] Wikipedia (community),
List CD,
http://en.wikipedia.org/wiki/Live_CD,
2014,
website.

[4] Wikipedia (community),
List of live CDs,
http://en.wikipedia.org/wiki/List_of_live_CDs,
2014,
website.

[5] Christopher Negus,
Live Linux CDs: Building and Customizing Bootables,
Pearson Education Inc.,
ISBN 0132432749,
2007,
book.

[6] Wikipedia (community),
Preboot Execution Environment,
http://en.wikipedia.org/wiki/Preboot_Execution_Environment,
2014,
website.

[7] Wikipedia (community),
Network File System,
http://en.wikipedia.org/wiki/Network_File_System,
2014,
website.

[8] Ubuntu Official Documentation,
Network File System (NFS),
https://help.ubuntu.com/14.04/serverguide/network-file-system.html,
2014,
website.

[9] Wikipedia (community),
Network File System,
http://en.wikipedia.org/wiki/Network_File_System,
2014,
website.

[10] Ubuntu Official Documentation,
iSCSI Initiator,
`https://help.ubuntu.com/14.04/serverguide/iscsi-initiator.html`,
2014,
website.

[11] SourceForge,
iSCSI Enterprise Target,
`http://iscsitarget.sourceforge.net/`,
2010,
website.

[12] Linhost.info,
Configure Ubuntu to serve as an iSCSI target,
`http://linhost.info/2012/05/configure-ubuntu-to-serve-as-an-iscsi-target/`,
2012,
website.

[13] OpenVPN Technlogies,
OpenVPN Community Software,
`https://openvpn.net/index.php/open-source/overview.html`,
2014,
website.

[14] Ubuntu Community Help Wiki,
VPN Server,
`https://help.ubuntu.com/community/VPNServer/`,
2014,
website.

[15] Ubuntu Community Help Wiki,
Live CD Customization,
`https://help.ubuntu.com/community/LiveCDCustomization`,
2014,
website.

[16] Open-iSCSI,
Open-iSCSI Project,
`http://www.open-iscsi.org/`,
2005,
website.

List of Figures

List of Tables

Appendices

A Purged packages (list)

account-plugin-aim*
account-plugin-jabber*
account-plugin-salut*
account-plugin-yahoo*
activity-log-manager*
aisleriot
brasero*
checkbox-gui
cheese*
cheese-common*
deja-dup*
empathy*
eog
evince*
evolution-data-server-online-accounts
example-content
firefox*
gcr
gnome-bluetooth*
gnome-contacts
gnome-font-viewer
gnome-mahjongg
gnome-mines
gnome-orca
gnome-power-manager
gnome-sudoku
gucharmap*
ibus
im-config
indicator-bluetooth*v landscape-client-ui-install
language-selector-gnome
libcheese-gtk23*
libreoffice*
mcp-account-manager-uoa*
nautilus-sendto-empathy
network-manager
network-manager*
printer-driver-foo2zjs-common*
remmina*
seahorse*
shotwell*
simple-scan*
software-properties-gtk*
system-config-printer-gnome
thunderbird*
totem*
transmission-gtk
ubiquity
unity-control-center-signon*
unity-lens-photos*

unity-scope-gdrive
unity-webapps*
update-manager*
vino
webaccounts-extension-common*
webbrowser-app*
xdiagnose
xterm
xul-ext-ubufox
xul-ext-webaccounts*
yelp*

B send_client_information (code)

```
#!/bin/bash

# SERVICE: send_client_information
# USAGE: start

case "$1" in
start)
        echo \[send_client_information\] starting service

        echo \[send_client_information\] mounting NFS share

        # mount the NFS share
        mkdir /mnt/nfs
        mount 10.8.0.1:<NFSPATH> /mnt/nfs

        echo \[send_client_information\] appending network information \(ifconfig\)

        # extract IP address (eth0) from ifconfig
        IP=$(ifconfig eth0 | grep -i 'inet addr' | awk '{print $2}' | sed 's/addr://g')

        # extract IP address (tun0) from ifconfig
        VPNIP=$(ifconfig tun0 | grep -i 'inet addr' | awk '{print $2}' | sed 's/addr://g')

        # date and time (of start/creation)
        DATE=$(date +%Y%m%d%H%M)

        # file name of log file
        LOGNAME=$(echo "$DATE"-"$IP"-"$VPNIP" | sed 's/\./_/g')
        LOGFILE=/mnt/nfs/"$LOGNAME".txt

        # add/append network information to log file
        echo IP ADDRESS: $IP > $LOGFILE
        echo VPN IP ADDRESS: $VPNIP >> $LOGFILE

        echo \[send_client_information\] appending iSCSI targets \(\/etc\/iet\/ietd.conf\)

        # append iSCSI targets to log file
        echo "" >> $LOGFILE
        echo "iSCSI TARGETS:" >> $LOGFILE
        echo "==============" >> $LOGFILE
        cat /etc/iet/ietd.conf | grep '^Target ' | sed 's/Target //g' >> $LOGFILE

        echo \[send_client_information\] appending disk information \(lshw\)

        # append disk information to log file
        echo "" >> $LOGFILE
        echo "DISK INFORMATION:" >> $LOGFILE
        echo "====================" >> $LOGFILE
        lshw -c disk >> $LOGFILE

        echo \[send_client_information\] unmounting NFS share

        # unmount the NFS share
        umount /mnt/nfs
;;

*)
echo SERVICE: send_client_information
echo $"USAGE: $0 {start}"
exit 1

esac
exit 0
```

C Authoring tool 0.1 (code)

```
#!/bin/bash

# check arguments
if [[ $1 == "--help" ]];
then
        echo ""
        echo Remote Acquisition Boot Environment \(RABE\) authoring tool
        echo ============================================================
        echo ""
        echo This tool provides an interface for remastering the RABE iso file with network, iptables and OpenVPN settings.
        echo ""
        echo Syntax: ./rabe_authoring_tool \[iso-file\] \[iso-destination\] \[pxe-destination\]
        echo ""
        echo " [iso-file] the source RABE iso file"
        echo " [iso-destination] the destination path including file name for the iso file"
        echo " [pxe-destination] the destination path for the pxe extraction of the iso file"
        echo ""
        exit
else
        if [[ -z $1 ]];
        then
                echo ""
                echo RABE authoring tool requires a RABE iso file as an argument! Exiting!
                echo ""
                exit
        else
                ISOSRC=$1
        fi
        if [[ -z $2 ]];
        then
                ISODST="../rabe_$(date +%Y%m%d%H%M).iso"
        else
                ISODST="../$2"
        fi
fi

# mount iso
mkdir mntiso
mount -o loop $ISOSRC mntiso

# extract iso contents
mkdir livecd
rsync --exclude=/casper/filesystem.squahshfs -a mntiso/ livecd

# extract squashfs
mkdir edit
unsquashfs mntiso/casper/filesystem.squashfs
```

```
mv squashfs-root/* edit

# copy network settings for chroot
cp /etc/resolv.conf edit/etc/
cp /etc/hosts edit/etc/

# copy chroot scripts
cp scripts/start edit/
cp scripts/end edit/

# mount dev for chroot
mount --bind /dev/ edit/dev

echo ""
echo Type ./start to start working in the chroot environment.
echo ""

# enter chroot
chroot edit

# umount dev for chroot
umount edit/dev || umount -lf edit/dev

# remove chroot scripts
rm edit/start
rm edit/end

# regenerate manifest
chmod +w livecd/casper/filesystem.manifest
chroot edit dpkg-query -W --showformat='${Package} ${Version}\n' > livecd/casper/filesystem.manifest
cp livecd/casper/filesystem.manifest livecd/casper/filesystem.manifest-desktop
sed -i '/ubiquity/d' livecd/casper/filesystem.manifest-desktop
sed -i '/casper/d' livecd/casper/filesystem.manifest-desktop

# compress filesystem
rm livecd/casper/filesystem.squashfs
mksquashfs edit livecd/casper/filesystem.squashfs

# update filesystem.size file
printf $(du -sx --block-size=1 edit | cut -f1) > livecd/casper/filesystem.size

# update md5sum.txt file
cd livecd/
find -type f -print0 | sudo xargs -0 md5sum | grep -v isolinux/boot.cat | tee md5sum.txt

# create new iso
mkisofs -D -r -V "$IMAGE_NAME" -cache-inodes -J -l -b isolinux/isolinux.bin -c isolinux/boot.cat -no-emul-boot -boot-load-size 4 -boot-info-table -o $ISODST .

```

```
# go to parent directory
cd ..

# unmount iso
umount mntiso

# remove directories
rm -R mntiso
rm -R livecd
rm -R edit
rm -R squashfs-root

# extract new iso for PXE
echo ""
echo Extract files for PXE \[y/n\]:
read PXE

if [[ $PXE == y* ]];
then
        if [[ -z $3 ]];
        then
                PXEDST="pxe/"
                mkdir $PXEDST
        else
                PXEDST=$3
        fi

        # mount new iso
        mkdir mntnewiso
        mount -o loop $(echo $ISODST | sed 's/..\///g') mntnewiso

        # copy files for pxe
        cp -R mntnewiso/* $PXEDST

        # unmount iso
        umount mntnewiso

        # remove directory
        rm -R mntnewiso
else
        exit
fi
```

D Authoring tool 0.4 (code)

```
#!/bin/bash

# check arguments
if [[ $1 == "--help" ]];
then
        echo ""
        echo Remote Acquisition Boot Environment \(RABE\) authoring tool
        echo ==========================================================
        echo ""
        echo This tool provides an interface for remastering the RABE iso file with network, iptables and OpenVPN settings.
        echo ""
        echo Syntax: ./rabe_authoring_tool \[iso-file\] \[iso-destination\] \[pxe-destination\]
        echo ""
        echo " [iso-file] the source RABE iso file"
        echo " [iso-destination] the destination path including file name for the iso file"
        echo " [pxe-destination] the destination path for the pxe extraction of the iso file"
        echo ""
        exit
else
        if [[ -z $1 ]];
        then
                echo ""
                echo RABE authoring tool requires a RABE iso file as an argument! Exiting!
                echo ""
                exit
        else
                ISOSRC=$1
        fi
        if [[ -z $2 ]];
        then
                ISODST="../rabe_$(date +%Y%m%d%H%M).iso"
        else
                ISODST="../$2"
        fi
fi

# set regular expression pattern for IP address
IPPATTERN="^[1-2]?[0-9]?[0-9]{1}\.[1-2]?[0-9]?[0-9]{1}\.[1-2]?[0-9]?[0-9]{1}\.[1-2]?[0-9]?[0-
$"

echo ""
echo \[START USER INPUT\]

# set network configuration
echo ""
echo Set static network configuration \[y/n\]:
```

```
read NETWORK

if [[ $NETWORK == y* ]];
then
        # set static IP address
        echo ""
        until [[ $IPADDRESS =~ $IPPATTERN ]]
        do
                echo Enter the IP address of the client \[###.###.###.###\]:
                read IPADDRESS
        done

        # set netmask
        echo ""
        until [[ $NETMASK =~ $IPPATTERN ]]
        do
                echo Enter the netmask of the network \[###.###.###.###\]:
                read NETMASK
        done

        # set gateway
        echo ""
        until [[ $GATEWAY =~ $IPPATTERN ]]
        do
                echo Enter the gateway of the network \[###.###.###.###\]:
                read GATEWAY
        done
fi

# set IP address of remote server
echo ""
until [[ $SERVER =~ $IPPATTERN ]]
do
        echo Enter the IP address of the remote server \[###.###.###.###\]:
        read SERVER
done

echo ""
echo Are the OpenVPN server certificate \(ca.crt\), key \(ca.key\), index \(index.txt\) and serial in the openvpn/keys folder \[y/n\]:
read OPENVPN

if [[ $OPENVPN == y* ]];
then
        # copy temporary easy-rsa files
        cp /usr/share/easy-rsa/build-key openvpn/
        cp /usr/share/easy-rsa/openssl-1.0.0.cnf openvpn/
        cp /usr/share/easy-rsa/pkitool openvpn/
        cp /usr/share/easy-rsa/vars openvpn/
        cp /usr/share/easy-rsa/whichopensslcnf openvpn/

        # set source file
        cd openvpn
        source vars

        # generate client certificate and key
        ./build-key rabe_client

        # go to parent directory
        cd ..
else
        echo ""
        echo RABE authoring tool requires OpenVPN server ca.crt, ca.key, index.txt and serial files in openvpn/keys folder! Exiting!
        echo ""
        exit
fi

# set NFS share path
echo ""
echo Set NFS share path \[y/n\]:
read NFS

if [[ $NFS == y* ]];
then
        echo ""
        echo Enter the NFS share path \[\/\<path\/to\/nfs\/share\>\/\]:
        read NFSPATH
        NFSPATH=$($NFSPATH | sed 's/\//\\\//g')
else
        NFSPATH="\/mnt\/nfsroot\/"
fi

echo ""
echo \[END USER INPUT\]
echo ""

# mount iso
mkdir mntiso
mount -o loop $ISOSRC mntiso

# extract iso contents
mkdir livecd
rsync --exclude=/casper/filesystem.squahshfs -a mntiso/ livecd

# extract squashfs
mkdir edit
unsquashfs mntiso/casper/filesystem.squashfs
mv squashfs-root/* edit

# copy network settings for chroot
cp /etc/resolv.conf edit/etc/
cp /etc/hosts edit/etc/

# copy chroot scripts
cp scripts/start edit/
cp scripts/end edit/
```

```

# mount dev for chroot
mount --bind /dev/ edit/dev

echo ""
echo Type ./start to start working in the chroot environment.
echo ""

# enter chroot
chroot edit

# umount dev for chroot
umount edit/dev || umount -lf edit/dev

# remove chroot scripts
rm edit/start
rm edit/end

# append network settings and related firewall rule
if [[ $NETWORK == y* ]];
then
# append network settings to configuration file (static IP)
        echo iface eth0 inet static >> edit/etc/network/interfaces_new
        echo address $IPADDRESS >> edit/etc/network/interfaces_new
        echo netmask $NETMASK >> edit/etc/network/interfaces_new
        echo gateway $GATEWAY >> edit/etc/network/interfaces_new
else
        # append network settings to configuration file (DHCP)
        echo iface eth0 inet dhcp >> edit/etc/network/interfaces_new

        # append firewall rule to configuration file (allow DHCP)
        echo -A INPUT -i eth0 -p udp --dport 67:68 --sport 67:68 -m state --state NEW,ESTABLISHED -j ACCEPT >> edit/etc/iptables/rules.v4
fi

# append firewall rules to configuration file (allow all remote server)
echo -A INPUT -i eth0 -s $SERVER -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT >> edit/etc/iptables/rules.v4

if [[ $OPENVPN == y* ]];
then
        # copy/move OpenVPN client certificate and key files
        cp openvpn/keys/ca.crt edit/etc/openvpn
        mv openvpn/keys/rabe_client.crt edit/etc/openvpn
        mv openvpn/keys/rabe_client.key edit/etc/openvpn

        # replace in OpenVPN client configuration file
        sed -i "s/<SERVER>/$SERVER/g" edit/etc/openvpn/rabe_client.conf

        # append firewall rules to configuration file (allow all OpenVPN server)
        echo -A INPUT -i tun0 -s 10.8.0.1 -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT >> edit/etc/iptables/rules.v4

        # remove temporary easy-rsa files
        rm openvpn/build-key
        rm openvpn/openssl-1.0.0.cnf
        rm openvpn/pkitool
        rm openvpn/vars
        rm openvpn/whichopensslcnf

        # remove temporary signing request file
        rm openvpn/keys/rabe_client.csr
fi

# append firewall rules to configuration file (block all other)
echo -A INPUT -j DROP >> edit/etc/iptables/rules.v4
echo COMMIT >> edit/etc/iptables/rules.v4

# replace in send_client_information service
sed -i "s/<NFSPATH>/$NFSPATH/g" edit/etc/init.d/send_client_information

# regenerate manifest
chmod +w livecd/casper/filesystem.manifest
chroot edit dpkg-query -W --showformat='${Package} ${Version}\n' > livecd/casper/filesystem.manifest
cp livecd/casper/filesystem.manifest livecd/casper/filesystem.manifest-desktop
sed -i '/ubiquity/d' livecd/casper/filesystem.manifest-desktop
sed -i '/casper/d' livecd/casper/filesystem.manifest-desktop

# compress filesystem
rm livecd/casper/filesystem.squashfs
mksquashfs edit livecd/casper/filesystem.squashfs

# update filesystem.size file
printf $(du -sx --block-size=1 edit | cut -f1) > livecd/casper/filesystem.size

# update md5sum.txt file
cd livecd/
find -type f -print0 | sudo xargs -0 md5sum | grep -v isolinux/boot.cat | tee md5sum.txt

# create new iso
mkisofs -D -r -V "$IMAGE_NAME" -cache-inodes -J -l -b isolinux/isolinux.bin -c isolinux/boot.cat -no-emul-boot -boot-load-size 4 -boot-info-table -o $ISODST .

# go to parent directory
cd ..

# unmount iso
umount mntiso

# remove directories
rm -R mntiso
rm -R livecd
rm -R edit
rm -R squashfs-root

# extract new iso for PXE
echo ""
echo Extract files for PXE \[y/n\]:
read PXE

if [[ $PXE == y* ]];
then
        if [[ -z $3 ]];
        then
                PXEDST="pxe/"
                mkdir $PXEDST
        else
                PXEDST=$3
        fi

        # mount new iso
        mkdir mntnewiso
        mount -o loop $(echo $ISODST | sed 's/..\///g') mntnewiso

        # copy files for pxe
        cp -R mntnewiso/* $PXEDST

        # unmount iso
        umount mntnewiso

        # remove directory
        rm -R mntnewiso
else
        exit
fi
```