D2.1.4/D2.3.3
Proof of Concept Infrastructure / Implementation of Security Configuration and Policy Management

Project number:	257243
Project acronym:	TClouds
Project title:	Trustworthy Clouds - Privacy and Resilience for Internet-scale Critical Infrastructure
Start date of the project:	1st October, 2010
Duration:	36 months
Programme:	FP7 IP

Deliverable type:	Prototype
Deliverable reference number:	ICT-257243 / D2.1.4/D2.3.3 / 1.0
Activity and Work package contributing to deliverable:	Activity 2 / WP 2.1, WP 2.3
Due date:	April 2013 – M31
Actual submission date:	29th April, 2013

Responsible organisation:	IBM, SRX
Editor:	Sören Bleikertz, Norbert Schirmer
Dissemination level:	Public
Revision:	1.0

Abstract:	Accompainying document for prototypes Trustworthy OpenStack and TrustedInfrastructure Cloud
Keywords:	prototypes, trustworthy infrastructure

Editor

Sören Bleikertz, Norbert Schirmer (IBM, SRX)

Contributors

Roberto Sassu, Paolo Smiraglia, Gianluca Ramunno (POL)

Alexander Buerger, Norbert Schirmer (SRX)

Sören Bleikertz, Zoltan Nagy (IBM)

Imad M. Abbadi, Anbang Ruad (OXFD)

Johannes Behl, Klaus Stengel (TUBS)

Mihai Bucicoiu, Sven Bugiel, Hugo Ideler, Stefan Nürnberger (TUDA)

Disclaimer

This work was partially supported by the European Commission through the FP7-ICT program under project TClouds, number 257243.

Executive Summary

This report accompanies the prototype deliverables D2.1.4 and D2.3.3 which are about the implementation and integration of subsystems of Activity 2 and provides the documentation of these prototypes. It is a revision of the documentation that has been presented earlier in Part II of D2.4.2. This report documents the subsystems of both the prototype deliverable D2.1.4, for the single trusted cloud, as well as D2.3.3, for the security management. A combined report captures the integration that has been done on the implementation level, since the subsystems of WP2.1 and WP2.3 are now integrated into the prototypes *Trustworthy OpenStack* and *Trusted-Infrastructure Cloud*.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This report accompanies the prototype deliverables D2.1.4 and D2.3.3 which are about the implementation and integration of subsystems of Activity 2 and provides the documentation of these prototypes. It is a revision of the documentation that has been presented earlier in Part II of D2.4.2. This report documents the subsystems of both the prototype deliverable D2.1.4, for the single trusted cloud, as well as D2.3.3, for the security management. A combined report captures the integration that has been done on the implementation level, since the subsystems of WP2.1 and WP2.3 are now integrated into the prototypes *Trustworthy OpenStack* and *Trusted-Infrastructure Cloud*.

Both prototypes are hosted by partner Sirrix and in the remainder of the project the benchmark applications from Activity 3 are deployed on top of them. The following subsystems are associated with WP2.1:

- LogService §2.1
- Remote Attestation Service §2.2
- Cryptography-as-a-Service (Caas) §2.5
- Resource-efficient BFT (CheapBFT) §2.6
- Simple Key/Value Store (hsMemcached) §2.7

And the following ones belong to WP2.3:

- Ontology-based Reasoner §2.3
- Access Control as a Service §2.4
- Security Assurance of Virtualized Environments (SAVE) §2.8
- TrustedObjectsManager §3.1

Chapter 2

Trustworthy OpenStack Prototype

The Trustworthy OpenStack prototype code is available either as a tarball, which can be downloaded from https://jenkins.tclouds-project.eu/tarballs or as a package which URL is https://jenkins.tclouds-project.eu/packages. These files are generated by our Jenkins infrastructure, described in detail in Appendix A of D2.4.2 deliverable, whenever a patch submitted by a partner successfully passes the tests. In this case, Jenkins merges the submitted patch into the `tclouds-stable-folsom` branch (or the `tclouds-2.9.0` branch, for *python-novaclient*) of the corresponding component repository on the TClouds GIT server. It is also possible to set up an APT[1] repository in order to automatically install Trustworthy OpenStack packages and their dependencies in the Ubuntu 12.04 LTS distribution. The following instructions will refer to the latter installation method.

Before setting up the TClouds APT repository, it is necessary to configure Ubuntu 12.04 LTS to fetch OpenStack Folsom packages. By default, this distribution provides packages for the Essex version. In order to do that, please follow the guide from the OpenStack documentation at the URL http://docs.openstack.org/folsom/basic-install/content/basic-install_controller.html. Then, execute the following instructions to set up the TClouds APT repository.

Edit the $HOME/.ssh/config file and add the following lines:

```
Host jenkins.tclouds-project.eu
        Port 1028
```

Copy your SSH public key into the Jenkins server (credentials are stored in the TClouds SVN).

```
$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub tcloudsuser@jenkins.tclouds-project.eu
```

Create the file `/etc/apt/sources.list.d/tclouds.list` to include the TClouds packages repository in the APT sources:

```
deb ssh://jenkins@jenkins.tclouds-project.eu/home/jenkins/packages precise main
```

Create the file `/etc/apt/preferences.d/00-tclouds` with the following content to set the highest priority for packages contained in the TClouds APT repository:

```
Package: nova-*
Pin: version 2012.2.4+git*
Pin-Priority: 1001

Package: python-novaclient
Pin: version 2.9.0+git*
```

[1] https://help.ubuntu.com/12.04/serverguide/package-management.html

```
Pin-Priority: 1001

Package: horizon-*
Pin: version 2012.2.4+git*
Pin-Priority: 1001

Package: quantum-*
Pin: version 2012.2.4+git*
Pin-Priority: 1001
```

Refresh the APT configuration by executing:

```
# apt-get update
```

Then, you can install Trustworthy OpenStack normally by following steps described in the documentation of the OpenStack Web site at the URL http://docs.openstack.org/folsom/openstack-compute/install/apt/content. In order to use the Security Extensions of this prototype, you can refer to the documentation of specific subsystems in the remaining of this chapter.

Lastly, there are some remarks. First, after installing Nova or Quantum components it is necessary to edit a Libvirt configuration file, as explained in the documentation of Ontology-based Reasoner and, secondly, to use the TClouds theme in the Dashboard this additional command must be executed:

```
# apt-get install openstack-dashboard-tclouds-theme
```

2.1 LogService

The LogService is a Trustworthy OpenStack component providing secure logging capabilities. In this section will be shown the procedure to install the LogService on a Debian-like system. The instruction will be structured in three parts: the first one, will be on the setup of the the platforms that will host the services, the second, will be on the installation and configuration of the LogService sub-components and finally, the third will be focused on the Trustworthy OpenStack configuration. All the `.tar.gz` archives mentioned in the instructions will be available online in the official component page[2] and locally in the `tclouds-Y3-demo-<DATE>.tar.gz` archive.

2.1.1 Platform Setup

The old version of the LogService (presented in the Deliverable 2.4.2) was based on the library called `libsklog`. Such library is no longer maintained, due to the issues mainly related to the design of the code. For this reason a new library called `libseclog` has been developed. This new library implements the same functionality provided by the previous one including some new features as the *incremental verification* (necessary to mitigate the scalability issues) and the *driver-like management* of the secure logging schemes.

[2]TORSEC Group - Libseclog: http://security.polito.it/secure-logging/libseclog

Core library. Before proceeding with the installation of `libseclog` it's necessary to resolve several dependencies. All of them, except the Libumberlog library, could be installed through the package manager. To install the dependencies execute the commands listed below:

```
sudo apt-get update
sudo apt-get install make autoconf libtool libssl-dev uuid-dev libjansson-dev git pkg-config
git clone https://github.com/deirf/libumberlog.git libumberlog
cd libumberlog
mkdir m4
autoreconf -i
./configure
make
sudo make install
```

Listing 2.1: `libseclog` dependencies installation

When all the dependencies are installed it's possible to proceed with the installation of the `libseclog` library. To do that, follow the instructions listed below:

```
tar zxvf libseclog-0.1.0.tar.gz
cd libseclog
./autogen.sh
./configure --enable-ceelog --with-umberlog=/urs/local --enable-python
make
sudo make install
```

Listing 2.2: `libseclog` installation

Python bindings. At this point, the library a has been installed and the Python package containing the bindings has been generated to. To install it, run the following commands:

```
sudo apt-get update
sudo apt-get install python-pip
cd binding/python
sudo pip install pySeclog-0.1.0_<BUILD>.tar.gz
```

Listing 2.3: `libseclog` Python binding installation

2.1.2 LogService Subcomponents

The LogService is the composition of four components: LogCore, LogStorage, LogConsole and LogService Module. The LogCore is the trusted party involved in the initialisation and verification of a logging sessions. The LogStorage is the component that stores the log entries produced by Cloud Components and groups them by logging session. The LogConsole is a web based log management console merged in the OpenStack dashboard which can be used by an external entity (auditor or user) to verify and retrieve logs. Finally, the LogService Module is a software module that should be linked to the applications to interact withe the LogService.

LogService Module installation

This component is represented by a Python package called `logservice-client`. Such package provides the classes to write the client applicatio for both LogStorage and LogCore. In addition, it provides also Python logging handler implemented using the `libseclog` Python bindings. To install the package, run the following commands:

```
sudo apt-get update
sudo apt-get install python-pip
sudo pip install logservice-client-0.1.0.tar.gz
```

Listing 2.4: `logservice-client` Python package installation

LogCore installation

The LogCore requires, in addition to the `logservice-client` package, the installation of some other dependencies: Python Tornado, Celery, RabbitMQ and SQLAlchemy. The commands to install the dependencies and to install and run the service are listed below.

```
apt-get install rabbitmq-server
sudo /etc/init.d/rabbitmq-server start

sudo pip install tornado Celery SQLAlchemy
tar zxvf logcore-0.1.0.tar.gz
cd logcore
celeryd -l info -I tasks

python server.py --init-db (only the first time)
python server.py --host 0.0.0.0 --port 50001 --ls-host logstorage.example.com --ls-port 50002
```

Listing 2.5: LogCore installation and execution

The LogCore application is accessible only via HTTPS. In this way, the application will be able to perform client filtering based on the X509 certificate. The accepted clients, identified by the certificate subject, are specified in the file `static/authorized_clients` as depicted in the Listing 2.6.

```
/C=EU/ST=Italy/0=The Foo Company/OU=Foo Division/CN=Alice
/C=EU/ST=Italy/0=The Foo Company/OU=Foo Division/CN=Bob
/C=EU/ST=Italy/0=The Foo Company/OU=Foo Division/CN=Carol
...
```

Listing 2.6: `authorized_clients` file example

To retrieve more information about the usable parameters of the LogCore application, run the application with the option `--help`

```
python server.py --help

Usage:

  ./logcore [OPTIONS]

where OPTIONS must be one or more of these values

  -h  --host       Listening address          [0.0.0.0]
  -p  --port       Listening port             [50001]
  -c  --cert       Certificate file           [./static/cert.pem]
  -k  --key        Private key file           [./static/key.pem]
  -C  --CAcert     CA certificate file        [./static/cacert.pem]
  -l  --list       Authorized clients file    [./static/authorized_clients]

  --ls-host        LogStorage host            [logstorage]
  --ls-port        LogStorage port            [50002]

  --init-db        Initialize database
  -H  --help       Show this help
  -v  --version    Show version
```

Listing 2.7: LogCore help message

LogStorage installation

To install and run the LogStorage component it's necessary to run the commands depicted in the Listing 2.8. The same considerations done for the LogCore about the HTTPS settings must be applied to the LogStorage.

```
sudo apt-get install python-pip
sudo pip install tornado
tar zxvf logstorage-0.1.0.tar.gz
cd logstorage
python server.py --host 0.0.0.0 --port 50002
```

Listing 2.8: LogStorage installation and execution

The base version of the LogStorage stores locally the log entries using files as storage support. To enable the CheapBFT capabilities, it's necessary to run the application including the `--cheap-[host,port,path]` options. In the Listing 2.9 is depicted the help message of the LogStorage application.

```
python ./server.py --help

Usage:

  python ./server.py [OPTIONS]

where OPTIONS should be one or more of these values

  -h  --host      Listening address              [0.0.0.0]
  -p  --port      Listening port                 [50002]
  -c  --cert      Certificate file               [./static/cert.pem]
  -k  --key       Private key file               [./static/key.pem]
  -C  --CAcert    CA certificate file            [./static/cacert.pem]
  -l  --list      Whitelist file                 [./static/authorized_clients]

  --cheap-host    CheapBFT listenning address    [0.0.0.0]
  --cheap-port    CheapBFT listenning port       [58080]
  --cheap-path    CheapBFT API path              [/logservice]

  -H  --help      Shows this help
  -v  --version   Shows version
```

Listing 2.9: LogStorage application help

LogConsole installation

The LogConsole application is available in two modes: *standalone* and *integrated*. The former is a standalone application that will be not documented here, while the latter is the integration of the LogConsole code in the OpenStack dashboard application (Horizon). In any case, the LogConsole requires the installation of the package `logservice-client` (see "LogService Module installation" in Section 2.1.2)

2.1.3 Trustworthy OpenStack configuration

In this section will be illustrated the procedure to configure Trustworthy OpenStack in order to enable the secure logging facilities. In the following will be provided two sample configurations to enable the logging administration tab in the dashboard (Horizon) and to enable the secure logging in the OpenStack services (Nova).

Horizon

To enable the logging administration tab in the OpenStack dashboard it's necessary to install the Python package `logservice-clients` (see Section 2.1.2) on the server that hosts the application. Before running the application, go at the end of the file `local_settings.py` in `/etc/openstack-dashboard` and configure the application as presented in the example depicted in the Listing 2.10:

```
...

# Logging Tab settings

LOG_CLI_SSL_OPTS = {
    "certfile": "/path/to/file/logconsole.pem",
    "keyfile": "/path/to/file/logconsole.key",
    "ca_certs": "/path/to/file/cacert.pem"
}

LOG_CORE = {
    "host": "logcore.example.com",
    "port": 50001,
    "timeout": 60,
    "api_version": "0.1",
}
```

Listing 2.10: OpenStack Horizon `local_setting.py` file

Nova

To enable the secure logging features for the Nova services, edit the section `[secure_logging]` in the `nova.conf` file at `/etc/nova` as depicted in the Listing 2.11.

```
...

[secure_logging]
use_secure_log = True
logcore_address = logcore.example.com
logcore_port = 50001
logcore_cert = /path/to/file/logcore.pem
logstorage_address = logstorage.example.com
logstorage_port = 50002
cert = /path/to/file/novaservices.pem
key = /path/to/file/novaservices.key
cacert = /path/to/file/cacert.pem
freq = 100
size = 500
debug = True
secure_log_services = nova-scheduler,nova-api
```

Listing 2.11: OpenStack `nova.conf` file

2.2 Remote Attestation Service

This section provides the installation instructions for *OpenAttestation* and the documentation for the *RA Verifier* component. More details about the former component can be found in the `docs` folder of the official code repository (URL: https://github.com/OpenAttestation/OpenAttestation). The following instructions refer to Ubuntu 12.04 LTS distribution.

2.2.1 Operating Environment Setup

A typical installation of the *Remote Attestation Service* requires the setup of two hosts or virtual machines. One platform, called *Database Node*, is dedicated to run the Apache Cassandra database and the other, named *OpenAttestation Node*, to execute *OpenAttestation* and *RA Verifier*. The former platform should satisfy at least the following hardware requirements: CPU 2.0 Ghz, RAM 2 GB or more, Hard Disk 500 GB. Instead, for the latter platform, there are no particular needs.

Regarding the software requirements, it is necessary to configure in the *Database Node* a new APT repository by creating a file called `cassandra.list` in the `/etc/apt/sources.list.d/` directory:

```
deb http://www.apache.org/dist/cassandra/debian 11x main
deb-src http://www.apache.org/dist/cassandra/debian 11x main
```

Refresh the APT cache by executing:

```
# apt-get update
```

Then, install the following packages through APT:

- *Database Node*
  - `python2.7-dev` (python development files)
  - `python-pip` (utility to install the pycassa library)
  - `pycassa` python library (install it by executing `pip install pycassa`)
  - `openjdk-6-jdk` (OpenJDK JAVA development kit)
  - `gcc` and `make` (development tools)
  - `python-rpm` (tool to parse RPM packages)
  - `debmirror` (tool to download Ubuntu packages)
- *OpenAttestation Node*
  - `python2.7-dev` (python development files)
  - `python-pip` (utility to install the pycassa library)
  - `pycassa` python library (install it by executing `pip install pycassa`)
  - `python-matplotlib` (python library)
  - `python-networkx` (python library)
  - `mysql-client` (MYSQL client)
  - `mysql-server` (MYSQL server)

2.2.2 Prototype Build and Installation Instructions

Database Node

In order to use *RA Verifier* it is necessary to setup the database. First, install Apache Cassandra by executing the command:

```
# apt-get install cassandra
```

Extract the files from the tarball of the *ratools* software in the target directory (e.g. `/srv`):

```
$ tar zxf ratools-1.1.0.tar.gz -C /srv
```

Install the custom libraries for Apache Cassandra by calling the `install_cassandra_libs.sh` script and by providing the directory path of Cassandra libraries:

```
$ ./install_cassandra_libs.sh /usr/share/cassandra
```

Restart Apache Cassandra:

```
# service cassandra restart
```

Install the database schema by executing `cassandra-cli` in the `bin` directory of Apache Cassandra:

```
$ cassandra-cli -h localhost -B -f db/cassandra/schema/cassandra-schema-common.txt
$ cassandra-cli -h localhost -B -f db/cassandra/schema/cassandra-schema-deb.txt
$ cassandra-cli -h localhost -B -f db/cassandra/schema/cassandra-schema-rpm.txt
```

Create the `/etc/ra` directory, copy the configuration files in the `db/conf` directory of the *ratools* software to `/etc/ra` and remove the suffix `.sample`:

```
$ mkdir /etc/ra
$ cp db/conf/ra.conf.sample /etc/ra.conf
$ cp db/conf/pkgs_download_list.conf.sample /etc/ra/pkgs_download_list.conf
```

Edit the file `/etc/ra/ra.conf` and set the following BASH variables:

- `RABASEDIR` to the directory where the *ratools* tarball was extracted (e.g. `/srv/ratools-1.1.0`)
- `TARGETBASEDIR` to a temporary directory in a large partition (at least 100 GB) where packages will be downloaded (e.g. `/srv/ratools-1.1.0/Packages`).
- `CASSANDRAURL` to the <IP:port> of the Apache Cassandra database

Create the temporary directory specified in the `TARGETBASEDIR` variable:

```
$ mkdir /srv/ratools-1.1.0/Packages
```

Edit the file `/etc/ra/pkgs_download_list.conf` and configure the repositories of the Linux distributions from which the `update_pkgs.sh` script will download the packages and insert the information from extracted files into the database. The file has the following format:

```
<distro name> <distro version> <distro archs> <repo dir> <repo hostname> <subdir of \
    TARGETBASEDIR> <update type>
```

where the `distro archs` parameter may contain multiple architectures separated by comma. Allowed values will be introduced in Section 2.2.3.

OpenAttestation Node

Install the OpenAttestation DEB package with POL patches from the TClouds repository through the `apt-get` command:

```
# apt-get install oat-appraiser-base-oatapp
```

Then, install the *RA Verifier* component by following the instructions for the *Database Node*. For the configuration, it is necessary to set only the `RABASEDIR` variable in the configuration file `/etc/ra/ra.conf` to the directory where *ratools* was extracted.

Cloud Nodes

In order to perform the measurement of software executed in *Cloud Nodes* it is necessary to perform the following steps.

Extract the ratools package as done for the *Database Node* and copy the script `ima_load_policy.sh` located in the `ima_boot_scripts/ubuntu` directory of the *ratools* package into `/usr/share/initramfs-tools/scripts/init-bottom`:

```
# cp /srv/ratools-1.1.0/ima_boot_scripts/ubuntu/ima_load_policy.sh
/usr/share/initramfs-tools/scripts/init-bottom
```

Then, install the custom Linux kernel (with IMA enabled) from the TClouds repository:

```
# apt-get install linux-image-3.2.0-40-ima linux-headers-3.2.0-40-ima
```

Create the `/etc/ima` directory and put into it the IMA policy file named `ima-policy` which content should be:

```
measure func=BPRM_CHECK mask=MAY_EXEC
measure func=FILE_MMAP mask=MAY_EXEC
```

Finally, edit the file `/etc/fstab` and add the mount option `i_version` to the root filesystem.

2.2.3 Prototype Execution Instructions

Database Node

Insert data from packages of the configured Linux distribution into the database by executing the `update_pkgs.sh` script from the *ratools* software directory:

```
$ db/scripts/update_pkgs.sh
```

In addition, the script `update_pkgs.sh` allows to insert data from packages in a given directory by executing:

```
$ db/scripts/update_pkgs.sh -d <packages directory> -n <distro name> -q <distro
ver> -c <distro archs> -t <update type>
```

where allowed values for the options are:

- `<distro name>`: Fedora, Ubuntu
- `<distro ver>`: 17, 18, ... (for Fedora) or precise, quantal, ... (for Ubuntu)
- `<distro archs>`: i686, x86_64 (amd64 for Ubuntu), all (only for Ubuntu)
- `<update type>`: newpackage, updates, security (only for Ubuntu), testing

OpenAttestation Node

In order to configure OpenAttestation it is necessary to send commands from the command line interface e.g. by using the utility `curl`. Alternatively, we added some scripts, to ease the configuration task, that build requests to OpenAttestation by taking as input a small set of parameters. In the following, we use these scripts for the configuration.

Get the certificate of the Appraiser:

```
# oat_cert -h localhost
```

Register a *Cloud Node* (allowed values for Cloud Node OS are: Fedora<version>, precise):

```
# oat_host_demo -h localhost -t <Cloud Node hostname> -a <Cloud Node IP> -o
<Cloud Node OS>
```

Add a PCR to the whitelist for the added *Cloud Node*:

```
# oat_pcrwhitelist_demo -h localhost -t <Cloud Node hostname> -o <Cloud Node OS>
-n <PCR #> -v <PCR value>
```

Before using the *OpenAttestation Node* with OpenStack, it is recommended to try the *RA Verifier* component alone by testing the verification of an IMA measurements file. Obtain it by booting a *Cloud Node* with the updated configuration and executing:

```
# cat /sys/kernel/security/ima/ascii_runtime_measurements >
ima_measurements.txt
```

The file `ima_measurements.txt` should be similar to:

```
10 821...8fd ima bfc...5d2 boot_aggregate
10 1e1...0ae ima 481...be8 /proc/self/exe
10 232...182 ima 16b...ec4 ld-linux-x86-64.so.2
10 1d6...7b5 ima 5c4...d86 libc.so.6
10 f1c...f9f ima 231...b2f /bin/run-init
10 80e...712 ima ce7...d81 klibc-bhN-zLH5wUTKSCGch2ba2xqTtLE.so
10 19e...9b8 ima f98...b15 /sbin/init
10 ba6...3d6 ima 16b...ec4 ld-2.15.so
10 064...fd1 ima 317...251 libnih.so.1.0.0
10 0fc...36e ima 444...ae7 libnih-dbus.so.1.0.0
10 d66...f88 ima cde...037 libdbus-1.so.3.5.8
10 dfd...5db ima c5f...10d libpthread-2.15.so
10 3ca...96e ima 28e...f9c librt-2.15.so
10 d0f...6ed ima 5c4...d86 libc-2.15.so
10 0e5...955 ima 9df...f20 libnss_compat-2.15.so
10 a86...0a5 ima ba5...be3 libnsl-2.15.so
10 2b1...c0f ima 413...84e libnss_nis-2.15.so
10 1c6...fb2 ima c18...286 libnss_files-2.15.so
10 d8a...545 ima 5c8...27a /bin/hostname
10 dd1...def ima def...af1 /bin/sh
```

Then, copy this file to the *OpenAttestation Node* and verify the IMA measurements by executing the `ra_verifier.py` script from the *ratools* software directory:

```
$ verifier/ra_verifier.py -i ima_measurements.txt -q precise -t openattestation
-a IMA_STANDARD
```

It should return an output like:

```
ok:  501, unknown:  0, pkg_security:  0, pkg_not_security:  0
```

If the number of unknown measurements is greater than zero, it is possible that the digest of the file `ld.so.cache` is present in the measurement list due to the READ_IMPLIES_EXEC flag set in the process personality (see http://lwn.net/Articles/94068 for more details). If the above flag is set, the `ld.so.cache` file is measured by IMA even if the only protection flag set by the application that issued an mmap() system call is PROT_READ (the IMA policy created before should trigger the measurement of files executed or mmapped with protection flag PROT_EXEC set).

A workaround to temporarily address this issue is to disable services that have this behavior. At the moment discovered services are `grub-common` and `zfs-fuse`. To disable these services, execute the following command:

```
# update-rc.d <service> disable
```

Further, also the file `/etc/rc.local` will cause the detection of unknown digests because it is not included in any package. For this reason, since it is not mandatory as it is called by the `/etc/init.d/rc.local` script to execute users' provided commands, it should be moved to another location, like the user's home directory.

Finally, it is necessary to add to the database the files installed in the Cloud Node by *OpenAttestation* during the enrollment phase. To perform this task it is necessary to execute the following commands.

Create a temporary directory called `oat-rpm`:

```
$ mkdir oat-rpm
```

Copy the *OpenAttestation* RPM from the provided tarball to the created directory:

```
$ cp <tarball dir>/Trustworthy-OpenStack/POL/NIARL_OAT_Standalone-2.0-1.x86_64.rpm
oat-rpm
```

Insert content of the *OpenAttestation* RPM into the database:

```
$ db/scripts/update_pkgs.sh -d $(pwd)/oat-rpm -n Ubuntu -q precise -c amd64 -t
testing
```

Trustworthy OpenStack Management Node

Edit the configuration file `/etc/nova/nova.conf` and add the following lines at the end of the `[DEFAULT]` group so that *Nova Scheduler* filters the Cloud nodes that can run a VM depending on the integrity level specified by users:

```
scheduler_available_filters=nova.scheduler.filters.trusted_filter.TrustedFilter
scheduler_default_filters=TrustedFilter
```

Then, add these lines at the end of the same file to tell *Nova Scheduler* that it should contact the *OpenAttestation Node* at the specified address in order to obtain the current integrity level of a *Cloud Node*:

```
[trusted_computing]
server=<IP of OpenAttestation Node>
port=8443
api_url=/AttestationService/resources
```

2.3 Ontology-based Reasoner

2.3.1 Operating Environment Setup

In order to install Ontology-based Reasoner, it is necessary to install the package `openvswitch-datapath-dkms` by executing the command:

```
# apt-get install openvswitch-datapath-dkms
```

If the platform where this software is installed is also a *Cloud Node* for the remote attestation, it is necessary to rebuild the module for the custom kernel with IMA enabled. By assuming that this kernel was selected at boot time, the command to rebuild the module is:

```
# apt-get install --reinstall openvswitch-datapath-dkms
```

2.3.2 Prototype Build and Installation Instructions

Before installing the Ontology-based Reasoner subsystem, follow the documentation to install OpenStack Quantum until the step at the URL `http://docs.openstack.org/folsom/openstack-network/admin/content/install_quantum_server.html`. Instead of installing the standard Open vSwitch agent, as described in the next step, execute the command:

```
# apt-get install quantum-plugin-openvswitch-libvirt-agent
```

In order to use the Ontology-based Reasoner within Ubuntu 12.04 LTS, it is necessary to modify the configuration of the Open vSwitch init script, to clear the list of bridges stored in the persistent database. To perform this task it is sufficient to add following line in the file /etc/default/openvswitch-switch:

```
DELETE_BRIDGES=yes
```

If the Quantum L3 Agent is being used, it is necessary to properly configure network interfaces to allow virtual machines deployed with Trustworthy OpenStack to connect to Internet. To perform the configuration, it is sufficient to edit the file `/etc/network/interfaces` and to replace the name of the interface with public IP address with `br-ex`, the name of the bridge used by the L3 Agent.

Then, append the following lines to the configuration of `br-ex`:

```
pre-up ifconfig <public interface> up
pre-up /etc/init.d/openvswitch-switch start
pre-up ovs-vsctl add-br br-ex
pre-up ovs-vsctl add-port br-ex <public interface>
```

After that, disable the automatic start of Open vSwitch daemons, by executing:

```
# update-rc.d openvswitch-switch disable
```

Lastly, to fix a problem with Libvirt, edit the file /etc/libvirt/qemu.conf and set the following variable :

```
security_driver = "none"
```

2.3.3 Prototype Execution Instructions

In order to use Ontology-based Reasoner together with Trustworthy OpenStack, it is necessary to slightly modify some configuration files, in respect to what is written in the official OpenStack documentation.

Edit the file /etc/nova/nova.conf and set the following variable:

```
libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtTVDOpenVswitchVirtualPortDriver
```

Edit the files /etc/quantum/dhcp_agent.ini and /etc/quantum/l3_agent.ini (in the node where the DHCP and L3 agents are running) and set the following variable:

```
interface_driver = quantum.agent.linux.interface.OVSLibvirtInterfaceDriver
```

Edit the file /etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini and set the following variables:

```
tenant_network_type = gre
enable_tunneling = True
tunnel_id_ranges = 1:1000
local_ip = <your public IP>
local_interface = <network interface with IP set in the local_ip variable>
```

2.4 Access Control as a Service

2.4.1 Platform Setup

Ubuntu 12.10 is deployed as the base system on every node, installed and configured with default packages . The prototype relies on a complete deployment of OpenStack with ACaaS patches, and the Trusted Computing Infrastructure. A typical deployment includes deploying one node as the management nodes, and several nodes as the compute nodes.

ACaaS Setup

ACaaS prototype is implemented on OpenStack Folsom release. Its building and installation are as simple as applying ACaaS patch to Folsom source codes, and then following the general python software building procedure. As described above, two major components are modified: the OpenStack Nova Compute, and the python-novaclient.

1. OpenStack Nova

   - Fetching nova source code

     ```
     $apt-get source nova-compute
     ```

   - Applying ACaaS patches

     ```
     $cd nova-2012.1
     $patch -p1 < nova-acaas.patch
     ```

   - Build and install

     ```
     $python setup.py build
     $sudo python setup.py install
     ```

- Restart the services (For management node)

```
$sudo restart nova-api
$sudo restart nova-scheduler
```

(For compute node)

```
$sudo restart nova-compute
```

2. Python-novaclient

- Fetching python-novaclient source code

```
$apt-get source python-novaclient
```

- Applying ACaaS patches

```
$cd python-novaclient
$patch -p1 < novaclient-acaas.patch
```

- Build and install

```
$python setup.py build
$sudo python setup.py install
```

Trusted Computing Infrastructure Setup

The management node is deployed with the nova-scheduler, nova-api, central database, rabbitmq. A compute node is deployed with nova-compute, and optionally nova-volume and nova-network. These deployments and configurations follow the general OpenStack settings, which are well documented and can be found online, e.g. the "OpenStack Install and Deploy Manual" (http://docs.openstack.org/essex/openstack-compute/install/apt/content/index.html)

To enable ACaaS, the ACaaSScheduler should be specified as the computer scheduler driver, which can be achieved by modifying the /etc/nova/nova.conf to have the corresponding field set as follows:

```
compute_scheduler_driver = nova.scheduler.acaas_scheduler.ACaaSScheduler
```

Trusted Computing Infrastructure is deployed for enabling the Trusted-based scheduling, providing by ACaaS prototype. It is composed of the integrity measurement and reporting service on the compute nodes, and the remote attestation service on the management node.

1. Measurement Services

   The measurement services build the chain-of-trust on every compute node from its Core-Root-of-Trust-for-Measurement, a specific piece of code in it BIOS up to every software component running on top of it. The chain-of-trust is built in an iterative method during the bootstrapping procedure of the node, i.e. every component participating in the booting procedure measure the next component before loading and giving control to it. The term 'measure' in TCG terminology stands for taking the hash value of the target software component and store the value into the TPM.

   Consequently, every software component responsible for booting a platform should be modified to support the measurement services, namely the BIOS, bootloader, and the OS kernel. First of all, the integrity measurement service should be turned on in the BIOS to enable the first-step measurement for measuring the bootloader. Secondly, the trusted

BIOS should be installed to implement the trusted boot for measuring the kernel, e.g. the TrustedGrub (http://sourceforge .net/projects/trustedgrub/). Finally the Linux kernel on our base system should have its IMA component built and enabled, for implementing the measurement of every software component loaded on the platform. To implement this, the kernel of the base system (Ubuntu 12.04) should be re-configured and compiled with the IBM IMA configuration turned on. The kernel-arg entry in the grub configuration should also be added with the ”ima_tcb” argument.

2. Remote Attestation Service

   Remote attestation service is implemented by the OpenPTS system in our prototype. The deployment of OpenPTS includes the measurement reporting sub-system on the compute nodes, and attestation sub-system on the management node:

   (a) Setting up the compute nodes

   - Installing OpenPTS components

```
$sudo apt-get install trousers tpm-tools libtspi-dev libtspi1
$sudo dpkg -i openpts-0.2.6-2.x86_64.deb
```

   - Setting up TPM

```
$tpm_take_ownership -y -z
```

   - Configure ptsc
     Adjust the configuration le /etc/ptsc.conf, choosing/configuring the appropriate reference models (rm). The detailed rm configurations are specific to each platform and are out of scope of this documents. Detailed information can be found at
     http://sourceforge.jp/projects/openpts/. An exemplar is as follows:

```
irm.num=2
rm.model.1.pcr.4=grubpcr4hdd.uml
rm.model.1.pcr.5=grubpcr5 uml
rm.model.1.pcr8=grubpcr8.uml
rm.model.1.pcr.10=f12imapcr10.uml
```

   - Initialize Collector ptsc

```
$/usr/sbin/ptsc -i
```

   - Selftest the target platform

```
$/usr/sbin/ptsc -s
```

   - Startup tcsd and ptsc

```
$service trousers start
$service ptsc start
```

   - Set whether ptsc should run on startup

```
$chkconfig --add ptsc
```

   (b) Setting up the management nodes

   - Installing OpenPTS components

```
$sudo apt-get install trousers tpm-tools libtspi-dev libtspi1
$sudo dpkg -i openpts-0.2.4-1.x86_64.deb
```

- Setup SSH public key authentication between compute nodes and management node

```
$ssh-keygen
$ssh-copy-id ptsc@compute_node_N
```

- Enrollment with trust collector

```
$openpts -if compute_node_N
```

- Testing attestation with trust collector

```
$openpts -l ptsc -v compute_node_N
```

(c) Setting up the nova periodically attestation

To enable the periodically attestation supported by ACaaS Scheduler, ACaaS Scheduler Manager should be enabled, which can be achieved by modifying the /etc/nova/nova.conf to have the corresponding field set as follows:

```
scheduler_manager=nova.scheduler.manager_integrity.SchedulerManager
```

SchedulerManager is a modification to the OpenStach TrustedComputingPool. Instead of using the RESTful API, the SchedulerManager in ACaaS performs attestation directly. In our prototype this is achieved by invoking the OpenPTS facilities as described above. The following parameters can be specified in /etc/nova/nova.conf for controlling the openpts operation, with the default values given:

i. Enabling openpts When openpts is disabled, attestation in ACaaS is in demo mode: simply output text indicating the operation to invoke. Without a well-configured openpts infrastructure, enabling it will cause ACaaS to hang.

```
openpts_enabled = False
```

ii. Path of openpts command

```
openpts_bin = '/usr/bin/openpts'
```

iii. Path for storing aide db for openpts The aide db is used as the white-list, representing the expected trusted properties of a target node

```
openpts_aide_path = '/var/lib/aide/aide.gz'
```

iv. ssh username required by openpts

```
openpts_username ='ptsc'
```

v. ssh port number required by openpts

```
openpts_port = ''
```

vi. ssh key file required by openpts

```
openpts_key = ''
```

By modifying the configuration files, and if necessary, a minor portion of the scheduler manager source codes, other attestation facilities can easily be switched to.

2.4.2 Management Console

Requirement Management

Requirement management facilities provide functionalities for creating, removing and querying specifies user requirements for VM scheduling. The management interfaces are listed as follows:

- List all requirements

```
$nova-manage requirement list
```

- Create new requirement, with a string as the name of the requirement (e.g. location)

```
$nova-manage requirement create --requirement=NAME
```

- Remove the requirement with specific ID (e.g. 9)

```
$nova-manage requirement remove --id=ID
```

Trusted Requirement Management

Trusted Requirement management facilities provide functionalities for adding, removing, querying trusted properties (the while list database). These properties represent the expected configuration (or state) of a node. They can be trustworthy examined (attested to) by the trusted computing remote attestation as described above. Any violation of the attestation result to the white list (the expected configuration) indicates the untrusted state of the node, and will result in the node to be examined and re-initiated (in current prototype, the state-changed host with be removed with the white list property).

- List all white-lists

```
$nova-manage white-lists list
```

- Create new white-lists With a string as the name of the white-lists and the location for storing the white-list database as value for the name

```
$nova-manage white-lists create --white-list=white-list-file-path
```

- Remove the white-lists with specific WL_ID

```
$nova-manage white-lists remove --id=WL_ID
```

Security Properties Management

Security Properties management facilities provide functionalities for adding, removing, querying properties of a compute node. The management interfaces are listed as follows:

- Query a HOST for its security properties

```
$nova-manage host get_properties --host=HOST
```

- Add a security properties to a HOST Specified by NAME and VALUE. When NAME equals to "trusted-host", it represented the trusted properties and its VALUE should identifies one of the TP_IDs

```
$nova-manage host add_properties --host=HOST --properties="'NAME': 'VALUE',
..."
```

- Specify white-list White-list is specified to a host as a requirement to provide a same management interface. In current implementation, the name for the white-list requirement should be: whitelist.

```
$nova-manage host add_properties --host=HOST --properties="'whitelist':
'WL_ID'"
```

- Remove a security properties from a HOST, specified by NAME

```
$nova-manage host remove_properties --host=HOST --properties="['NAME', ...]"
```

Requirement-based VM Instantiation

- Initiating a VM with with general requirement matching The REQ_ID represents the requirement id for each requirement defined by the requirement management component, and the REQ_VALUE will be the expected value matching the specific requirement.

```
$nova boot --flavor XXX --image XXX --key_name XXX --security_group XXX
--req="REQ_ID:'REQ_VALUE'" IMAGE_NAME
```

- Staring VMs with exclude-user requirements A predefined alphabic REQ_ID is specified for these type of requirement matchings. The 'exclude-user' is represented by 'x' or 'X' as REQ_ID to specified the VM can only be launch on the compute host with NO other VMs belonging to the users identified by USER_IDs.

```
$nova boot --flavor XXX --image XXX --key_name XXX --security_group XXX
--req="'x':['USER_ID1', 'USER_ID2', ...]" IMAGE_NAME
```

- Staring VMs with trusted-hosts requirements A predefined REQ_ID is specified for these type of requirement matching. The 'trusted-host' is represented with 't' or 'T' as REQ_ID to specified the VM can only be launch on the compute host with a target attestation white-list (representing a set of trusted properties) identified by TP_ID.

```
$nova boot --flavor XXX --image XXX --key_name XXX --security_group XXX
--req="'w':TP_ID" IMAGE_NAME
```

2.5 Cryptography-as-a-Service (Caas)

2.5.1 Operating Environment Setup

Hardware Prerequisites

Cryptography-as-a-Service requires (in a minimal configuration) two platforms: a computing platform, on which the user's VMs are deployed, and verifying platform (short *Cloud Verifier*), which delegates management tasks between user and computing platform and which verifies the trustworthiness of the computing platform. The computing platform requires certain capabilities from the underlying platform:

Trusted Platform Module v1.2 A TCG TPM in version 1.2 must be present on the platform. Moreover, the module must be activated and owned. The owner and Storage Root Key (SRK) authentication values must be made available to the CaaS software.

Hardware Virtualization Support The CPU and chipset must support hardware virtualization, e.g., Intel VT-d or AMD-V. In the current version of CaaS only Intel VT-d is supported.

Download and Installation

CaaS builds on the Xen hypervisor in version 4.1.2, which can be retrieved from the Xen project webpage[3], the MiniOS (from the Xen 4.1 source tree), and Intel tboot[4] for a measured launch of the software stack. The corresponding patches (`libxl` and `libxc`) for Xen 4.1.2 and MiniOS to enable CaaS support and install a basic configuration/templates for a domain builder and crypto-service VM can currently only be retrieved upon request from partner TUDA[5].

CaaS can be easily integrated with OpenStack dashboard Horizon. Figure 2.1 depicts the components needed in order to run CaaS. For brevity, we will only focus on installing the CaaS modified components, the rest of them can be compiled and installed according to the default documentation.

Installation

For our prototype we use Ubuntu 12.04 as the base system, installed and configured with the default packages.

1. Install dependencies

```
 $ sudo apt-get install build-essential bcc bin86 gawk bridge-utils iproute
libcurl3 libcurl4-openssl-dev bzip2 module-init-tools transfig tgif
texinfo texlive-latex-base texlive-latex-recommended texlive-fonts-extra
texlive-fonts-recommended pciutils-dev git subversion mercurial make gcc
libc6-dev zlib1g-dev python python-dev python-twisted libncurses5-dev patch
libvncserver-dev libsdl-dev libjpeg-dev iasl libbz2-dev e2fslibs-dev git-core
uuid-dev ocaml ocaml-findlib libx11-dev bison flex xz-utils libyajl-dev gettext
exuberant-ctags libc6-dev-i386 python3 python3-crypto libtspi-dev debootstrap
tpm-tools trousers
```

[3] http://www.xen.org/
[4] http://sourceforge.net/projects/tboot/
[5] *mihai.bucicoiu@trust.cased.de*

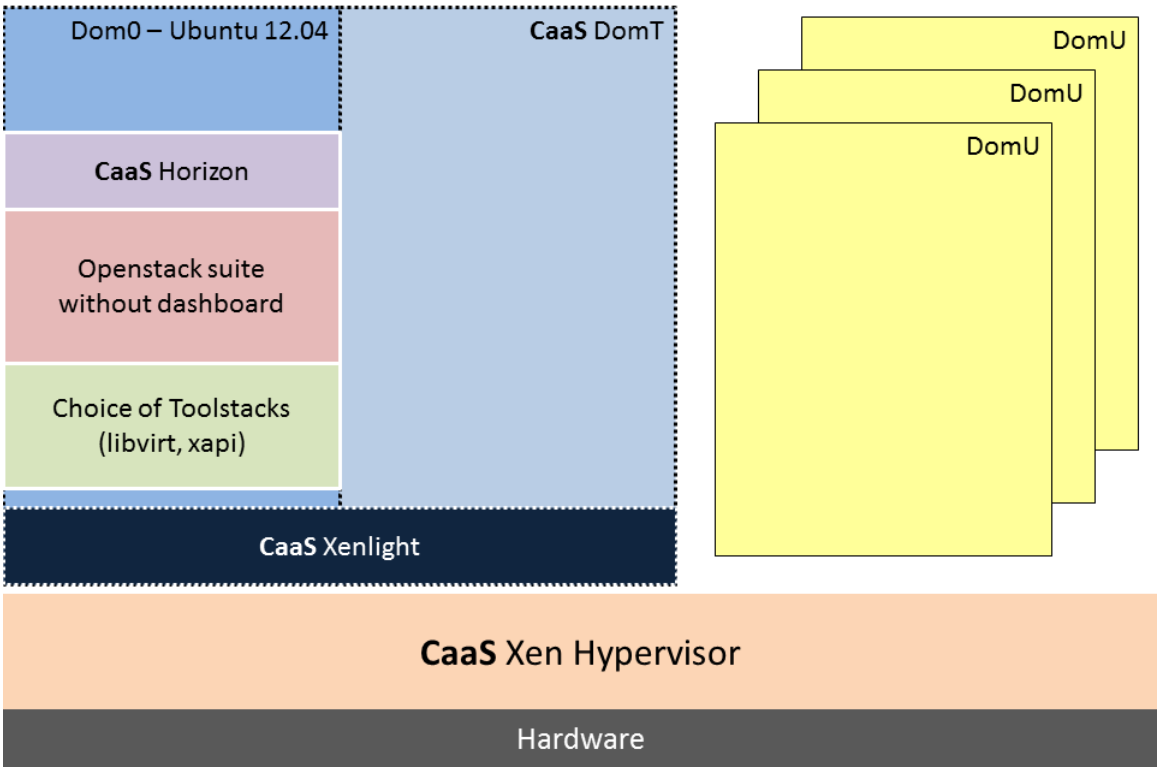

Figure 2.1: CaaS installation overview

2. Build and install CaaS-Xen kernel

```
 $make
$sudo make install
```

3. Build and install CaaS domT

```
 $cd stubdom
$make domt-stubdom
$sudo cp mini-os-x86_64-domt/mini-os /boot/domt
```

- Verify domT file

```
 $file /boot/domt
$/boot/domt:  ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically
linked, not stripped
```

4. Build and install CaaS domC

```
 $make
$make domc-stubdom
$sudo mkdir -p /usr/local/xen-domc
$sudo cp -p mini-os-x86_64-domc/mini-os.gz /usr/local/xen-domc/domc.gz
```

5. Set up some userspace helpers

```
 $cd tools/libxlc && make && sudo make install && sudo ldconfig
$cd tools/xenvfsd && make
$sudo cp xenvfsd /usr/local/bin/
$cd tools/xenkicker && make
$sudo cp xenkicker /usr/local/bin/
```

6. Compile and install CaaS OpenStack Horizon

```
 $sudo make install
```

Configuration

In order to make the system functional, some post-installation configuration need to be applied.

1. Fix a problem with compiled libs

```
 $echo /usr/lib64 | sudo tee -a /etc/ld.so.conf
$sudo ldconfig
```

2. Start Xen related daemons at bootime

```
 $sudo update-rc.d xencommons defaults
```

3. Start CaaS userspace tools

```
 $sudo vim /etc/rc.local
$LOGKICK=/var/log/xen/kicker
$OGVFSD=/var/log/xen/vfsd
$TMPVAL=/tmp/val
$dat=$(date)
$echo 1 > "$TMPVAL"
$echo "starting xenkicker $dat" > $LOGKICK
$tools/xenkicker/xenkicker >> $LOGKICK 2>&1
$echo "starting xenvfsd $dat" > $LOGVFSD
$tools/xenvfsd/xenvfsd /tmp/val >> $LOGVFSD 2>&1
```

4. Configure grub boot-loader

```
 $sudo rm -rf /boot/xen-4.1.gz /boot/xen-4.gz /boot/xen.gz /boot/xen-syms-4.1.2
$sudo update-grub
$sudo vim /etc/grub.d/20_ (add "module $rel_dirname/domt" after the line with
"multiboot")
$sudo mv /etc/grub.d/20_linux_xen /etc/grub.d/09_linux_xen
$echo "GRUB_CMDLINE_XEN=ño-real-mode dom0_mem=2048M loglvl=all guest_loglvl=all
console_timestamps iommu=verbose com1=115200,8n1,0x3f8,0 console=com1
caas_enforcing=1" | sudo tee -a /etc/default/grub
$sudo update-grub
```

5. Configure TBoot

   The configuration of the Intel Trusted boot follows the same steps explained in the software's documentation. One must be very careful when configuring the SINIT, this should be downloaded [6] for the architecture where CaaS is deployed. After this final step the grub booting menu should be similar to the following

```
 $kernel /boot/tboot.gz logging=serial,vga,memory vga_delay=0
$module /boot/xen.gz no-real-mode dom0_mem=2048M loglvl=all guest_loglvl=all
console_timestamps iommu=verbose com1=115200,8n1,0x3f8,0 console=com1
caas_enforcing=1
$module /boot/domt
$module /boot/vmlinuz-linux root=/dev/sda7 ro
$module /boot/initramfs-linux.img
$module /boot/pol/list.data
$module /boot/SINIT.BIN
$savedefault 5
```

One very important last step that needs to be accomplish before booting into CaaS Xen hypervisor, consist of configuring the TPM so that it can be used by domT. In order to do this, the user must have physical access to the running server, enter the BIOS, enable and clear the TPM. After enabling, boot into dom0, issue the following command and then reboot so that domT will be enabled.

```
$sudo tpm_takeownership -z -y
```

For the first boot, the default steps for booting a vanilla Xen with Intel tboot should be followed. During boot, the domain builder and management domain will automatically be launched. On first boot, the domain builder VM will automatically create a new TPM binding key, which is bound to the measurement of the platform and provided at a world-wide readable location. This key has to be provided to the user to enable encryption of VM images and secrets deployed in the cloud by the user.

2.5.2 Prototype Execution Instructions

After the system has booted and the configuration was completed, the workflow looks as follows:

- Cloud Verifier attests the Xen-based computing machine, and receives the public key of the certified binding key for this configuration. Figure 2.2

[6] http://software.intel.com/en-us/articles/intel-trusted-execution-technology/

- The user uploads his VM image to the cloud, and sends his user secret encrypted under the public TPM binding key to the CV.
- The computing platform asks the CV for the user secret in order to run the VM image.
- The CV sends the encrypted secret to the computing platform.
- The domain building code on the computing platform uses the TPM to decrypt the user secret, the VM image by using the secret, and starts building the domain.

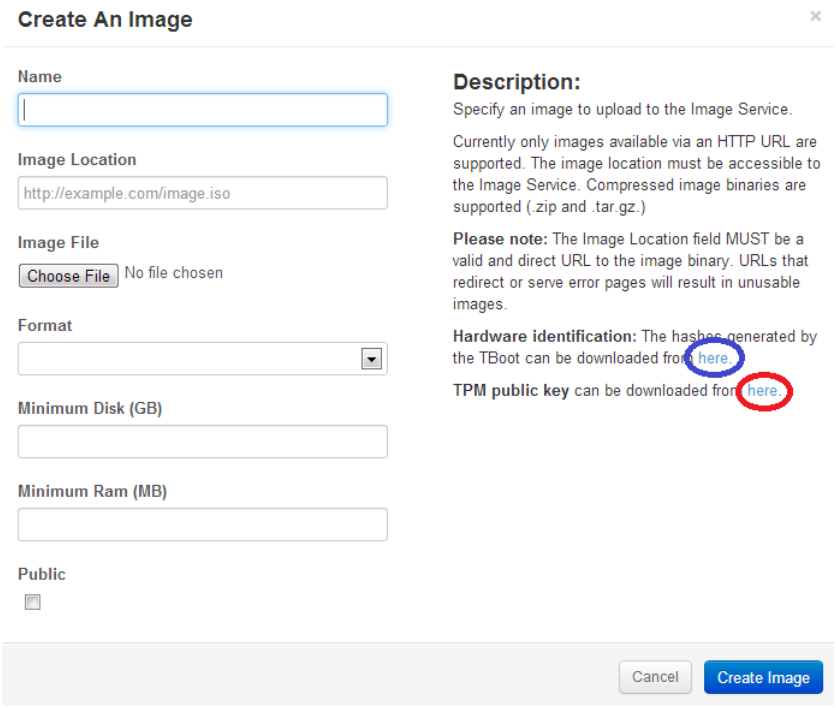

Figure 2.2: CaaS OpenStack dashboard integration

For a detailed description of the inner workings during user VM launch, we refer to deliverable D2.1.2 [ea12b].

2.6 Resource-efficient BFT (CheapBFT)

2.6.1 Operating Environment Setup

Hardware Prerequisites

A system setup of CheapBFT comprises at least four machines. Three server systems are required with the following specification:

- x86 (better x86-64) machines with identical or at least similar hardware specs; at least Intel Pentium 4 class processor
- 4 GB RAM
- PCI bus (necessary for the FPGA-cards, see below)

- 1 Gigabit/s or better IP-network connectivity
- Enterpoint "Raggedstone 1" FPGA card

In order to programm the FPGA cards, a Xilinx JTAG programmer with USB interface is necessary.

Further, one or two additional computer with similar specification as the server system (except the FPGA cards) are required for running the benchmark tools.

Software Environment

Following software should be installed on all test machines:

- Linux distribution with kernel 2.6.32 or better (e.g. Ubuntu 10.04), 32- or (preferably) 64-bits.
- `xz` data compression tools
- Eclipse 3.6 (Indigo) or better
- Java Runtime environment 6.0 or better
- Secure Shell Server (OpenSSH) with key-based authentication to enable scripted remote-logins
- `screen` terminal multiplexer
- Xilinx ISE WebPack 13.x for Linux
- Header files and build environment for the running Linux kernel

Installation

The required files are provided as a GNU tarball, packed with the xz data compression utility. In order to unpack it on a GNU/Linux system, use the following command: `xzcat cheapbft.tar.xz | tar xf -` All files are extracted into the subdirectory `cheapbft/`. The following subsections will describe the necessary steps to configure and translate the program code to get it into a working state. All specified paths are relative to the cheapbft directory, unless beginning with a forward slash (/), or mentioned otherwise.

FPGA Hardware The source code for the FPGA firmware and associated Linux kernel driver is located in the directory `cash/pci_base`. In order to build the firmware image, the Xilinx ISE tools must be configured properly and present in the search path of the Linux shell. This can be usually accomplished running the command `source settings32.sh` in the installation directory of the Xilinx suite.

Each of the three FPGA boards must be assigned an unique identifier so that they generate different signatures. This ID can be adjusted in the file `fpga/pci_base/source/adrdec.vhd` in the line starting with `constant SYS_ID`. The value is at the end of this line, enclosed in double quotes and must be a 16-bit hexadecimal number.

To compile the firmware image, change into the directory `fpga/pci_base` and simply call make. The shipped Makefile will create a `.bit` file suitable for the FPGA chip of the

"Raggedstone 1" board. This build process can take up to an hour, even on fast machines, as the logic is quite complex and requires expensive optimization processes to fit onto the FPGA.

When the build process has finished, the next step is to load the firmware onto the card. Make sure the USB programmer is connected to the Raggedstone board. There are two possible modes to store the firmware: The command `make load` will just reconfigure the FPGA and keep it until the host PC is turned off. The command `make i-really-want-to-flash` can be used to additionally save the image in persistent flash memory, so that it will survive when the machine is powered down. Just loading the firmware is the preferred mode when working on the FPGA code, as the flash memory has a limit on the number it can be erased. The host PC has to be rebooted before.

As each card requires an unique ID embedded in its firmware, the image has to be rebuilt and loaded for each of the three FPGA boards individually. After programming the FPGA, the PC hosting the card should be rebooted as soon as possible, otherwise the card won't be detected properly and may interfere with the operation of the machine in unpredictable ways.

Kernel Driver In order to use the cards from any Linux application software, it is necessary to install a kernel driver that can interface with the hardware registers on the FPGA board. The driver provides a device node called `/dev/counter` for the protocol implementation. To compile the driver, change into the directory `fpga/pci_base/kernel` and edit the Makefile so that the `KDIR` variable points to the directory containing the header files for your current kernel version.

Type make (using your regular user account) in order to build the driver. When the module `counter.ko` was built successfully, switch to the root account (e.g using `su` or `sudo`). Now the driver can be loaded using `make device`. This command will also create the necessary device node and allows everyone to use it. For security reasons it is advisable to change permissions for `/dev/counter`, so that access is restricted to the user account(s) that will run the application protocol later.

Userland applications The userland software is entirely written in the Java language (except for a few programs for testing purposes) and ships in form of Java archive (jar) files.

2.6.2 Prototype Execution Instructions

The benchmark suite is implemented as a set of bash scripts that connect to the participating machines via SSH and set up the required services. The progress can be monitored by switching between terminals in the `screen` software that is used to manage the parallel execution of aforementioned scripts. The required files are all stored in the directory `cheap/bin/`.

There is a script for each test scenario. For instance, `run_micro.bash` can be used to start micro benchmarks. However, the main functionality is implemented in `system.bash`, which is imported in every script for a scenario in order to provide consistent use.

The behavior of the running system is configured via three configuration files which are stored in `cheap/bin/config`: `system.config` is the main configuration file, which contains, for example, settings concerning the used consensus protocol. `hosts.config` contains a list of all replica endpoints and `logging.properties` can be used to configure the logging output generated during a run. Instead of editing these configuration files directly, the scenario scripts (or `system.bash`) should be used in conjunction with configuration templates (also contained in `cheap/bin/config`) to set up the system. For instance,

`system.config.cheap` is a template for a configuration with CheapBFT as consensus protocol. The same holds for `system.config.cheap.soft`, with the exception that a software module is employed for the message verification instead of an FPGA.

Before a test run can be started, the addresses of the hosts executing servers and clients have to be configured. The addresses are usually stored in files named according to the pattern `hosts.config.*`. These files contain two variables: `ENDPOINT_SERVERS` and `HOSTS_CLIENT`. The former is a list of machines equipped with the FPGA boards together with a base port. The specified machines act as servers and server sockets are opened with port numbers starting from the given base port as required. (For that purpose, at least five ports starting from the base port should be available.) The latter of the two variables is a list of host addresses only. Here, the clients for the test run are executed. Please ensure that your current user account can login to all specified server and client hosts via SSH, without being asked for a password. This can be accomplished by either deploying host-based authentication or (probably easier) just supplying proper SSH keys for the user.

After the host addresses have been provided, the test configuration has to be set up. For example, the command `./system.bash setup_prot cheap` sets `system.config.cheap` as protocol configuration, hence selects CheapBFT as consensus protocol. `./system.bash setup_hosts <yourhosts>` initializes the server and client hosts according to the file `hosts.config.<yourhosts>`. With `./system.bash setup_logging con` logging output is redirected to the console.

Next, a screen environment can be started by running `./system.bash screen` This should start the `screen` program with a basic configuration from which the individual client and server programs will be launched.

The actual benchmark can be started, for example, using `./run_micro.bash start` This will connect to the previously configured host machines and run a minimal performance benchmark. Statistics are printed in regular intervals to the screens.

All available commands and their options can be obtained by `./run_micro.bash help` (or `./system.bash help`).

2.7 Simple Key/Value Store (hsMemcached)

2.7.1 Introduction

This file describes the installation procedure for the hsMemcached prototype. hsMemcached is a reconfigureable and secure implementation of the Memcached protocol designed to run on top of a thin operating system layer in a virtualized environment, currently supporting the Xen hypervisor.

2.7.2 System Requirements

- 64-bit x86 machine with 64-bit Linux, version >= 2.6.32, glibc >= 2.7, 1 GB RAM, about 5 GB disk space, Internet connection
- C/C++ compiler from the GNU Compiler Collection (GCC), version >= 4.4

  The version shipped with recent Linux distributions is usually fine, older versions are known to miscompile gmplib.

- GNU Autoconf, version 2.59
- GHC Haskell compiler version 6.12, generic 64-bit Linux binaries

  HaLVM is still based on an older version of the GHC compiler. Newer version introduced incompatible changes to some of the language extensions used across the software stack.

  Download: http://www.haskell.org/ghc/download_ghc_6_12_3
- GNU Multiple Precision Arithmetic Library (gmplib), version 4.3.2

  The GHC 6.12 compiler expects the 4.x version of the library interface but most Linux distributions nowadays only ship with 5.x versions.

  Download: http://ftpmirror.gnu.org/gmp/gmp-4.3.2.tar.bz2
- Xen Hypervisor version 4.0.3 or better for building/executing VM image

  Earlier versions have a bug in the virtual memory manager that will cause the hypervisor to panic with certain workloads.

  Download: http://www.xen.org/download/index_4.0.3.html

2.7.3 Installation

Prerequisites

In order to build the system, it is necessary to ensure that proper versions of all the build requirements are available. The following subsections describe additional steps that may have to be taken on current Linux systems to make these available.

GMP Library The first thing needed is probably the GMP library, as many current Linux distributions no longer ship with the required older version. We suggest to query the your Linux distribution's package database whether any of the available packages provides the file `/usr/lib/libgmp.so.3`. If so, it is recommended just to install that package. Otherwise the following paragraph provides instructions for setting up the library from source code. After downloading the GMP package, unpack and install it to a temporary directory:

```
$ tar xfj gmp-4.3.2.tar.bz2
$ pushd gmp-4.3.2
$ ./configure --prefix=/tmp/ghc-6
$ make && make check && make install
$ popd
```

Now the library has to be added to the search list of the dynamic linker, which can be done by setting the following environment variable:

```
$ LD_LIBRARY_PATH=/tmp/ghc-6/lib
$ export LD_LIBRARY_PATH
```

Xen header files If the machine used to build doesn't already have a recent Xen version installed as part of the Linux distribution, it is at least necessary to partially compile the Xen source archive, so that HaLVM has access to the header files for the hypervisor interface. Just unpacking the Xen source code is not sufficient, as the exact contents of the header files depend on information about the machine and a set of configurable options. The Xen build process has its own set of prerequisites, which are described in the `README` file included with the Xen source archive. If these are met, we just do a local build of the hypervisor, which will create the necessary configuration and header files in the process.

```
$ tar xfz xen-4.0.3.tar.gz
$ pushd xen-4.0.3
$ :>.config
$ make tools
$ popd
```

Unpacking the source

The source code of the prototype is contained in the file `sources.tar.bz2` and can be uncompressed using the following command line:

```
$ tar xfj sources.tar.bz2
```

This will create a directory `sources` with three subdirectories with the required parts to build the service: HaLVM, HaNS and hsMemcached. The installation of each of these required components is described in the next sections.

HaLVM – Xen/Haskell Library OS

The interface to the Xen Hypervisor is provided by the HaLVM layer. The build process essentially downloads the source code of the GHC Haskell compiler from the Internet and builds a customized version and some libraries that provide access to Xen's low level driver interface. In case HaLVM can use the Xen version shipped with the Linux distribution on the build machine (i.e. the manual preparation of Xen was skipped), the `--with-xen-tree` option must be omitted from the configure command. We also add the resulting compiler wrappers (`halvm-*`) in the search path so that other haskell packages can use them.

```
$ pushd sources/HaLVM
$ autoconf2.59
$ ./configure --with-xen-tree="$PWD/../../xen-4.0.3"
$ make
$ PATH="$PWD/dist/bin:$PATH"
$ popd
```

HaNS – Network stack

In this step we will build the TCP/IP stack for HaLVM. This subsystem is based on the cabal, a package manager for Haskell. Thus, the cabal utility will take care of resolving all external dependencies and download any missing libraries from the Internet. The package will be installed into HaLVM's `dist/` directory.

```
$ pushd sources/HaNS
$ halvm-cabal install -f halvm
$ popd
```

hsMemcached – Configurable memcache service

Now that all operating system components are available, the final step is about configuring and installing the hsMemcached demo application. The service is also a cabal package, so it can be built almost the same way as HaNS from the previous subsection. The reconfigurable aspects of the memcache service can be found in `HsMemcached/Main.hs`. Changing the configuration parameters is done by editing the corresponding `*Config` and `*Advice` settings in that file.

```
$ pushd sources/hsMemcached
$ halvm-cabal install -f halvm
$ popd
```

The compiled binary will be installed into `HaLVM/dist/bin/memcached`. This is not a regular Linux executable, but a Kernel that can be booted as a Xen paravirtual instance, containing the hsMemcached application.

2.8 Security Assurance of Virtualized Environments *(SAVE)*

The SAVE prototype can either operate with access to the compute nodes and their hypervisor management interface directly, or using a modified version of OpenStack that collects hypervisor configurations of the compute nodes on SAVE's behalf. In the following we describe the latter version, although the former one can be used simply by adding the hostnames of the compute nodes (and appropriate credentials) to the probe configuration of § 2.8.3.

2.8.1 Operating Environment Setup

In order to enable the SAVE discovery using OpenStack, OpenStack needs to be modified. The environment has to fulfill the following requirements:

- OpenStack's cactus release patched with the TClouds API patch.
- Access to the OpenStack management interface.

After patching OpenStack, it can collect hypervisor configuration data on behalf of SAVE. To leverage the full capabilities of SAVE, the OpenStack cloud infrastructure should have a size and complexity which are exceed the capabilities for a manual assessment by an administrator.

2.8.2 Prototype Build and Installation Instructions

- Unzip the cactus release of OpenStack
- patch -p1 < openstack-save.patch
- start up and operate OpenStack as normally

2.8.3 Prototype Execution Instructions

SAVE relies on using probes to communicate with target systems. Such a probe must be configured either for OpenStack, or hypervisor types. An example configuration for OpenStack:

```
<?xml version="1.0" encoding="UTF-8"?>
<DiscoverConfig>
  <Host hostname="openstack1" address="127.0.0.1" enabled="true">
    <Credential type="API" username="openstack-admin" password="secretpassword" port="8775"/>
  </Host>
</DiscoverConfig>
```

Upon starting up SAVE, one can select the project to execute. This should be a directory containing the above XML in a ”discovery.xml” file. Once that is loaded, executing the probe and showing the discovery result in a visual way can be done from the GUI.

Chapter 3

TrustedInfrastructure Cloud Prototype

This chapter describes the TrustedInfrastructure Cloud prototype based on the subsystems TrustedObjects Manager, TrustedServer, Trusted Management Channel and the Confidentiality Proxy for S3. We focus on the description of the TrustedObjects Manager, as this provides the interface to the other subsystems.

3.1 TrustedObjectsManager setup

Essentially the TrustedObjectsManager (TOM) is a webserver that provides the necessary management and administrative functions via a web interface.
The managment console holds the complete set of configuration-files for all TrustedDesktops and TrustedServers, which are attached to the TOM. These appliances establish a permanent secure tunnel to the management console and get their dedicated initial configuration, as well as configuration changes via this tunnel automatically (TrustedChannel).
The mentioned appliances derive the settings (firewall-, router-, security-, user-settings, etc.) which have to be applied, from the centrally downloaded configuration and apply them autonomously.

This chapter describes the steps to be done in order to attach a TrustedServer and a TrustedDesktop to the TOM in order to start, stop install and remove a compartment (an virtual machine (VM) associated with a trusted virtual domain (TVD)) and to use its provided services.

3.1.1 Using the management console

The web interface of a newly installed TOM is reachable via the https-adress 192.168.1.11 from any client's webbrowser within the same network, showing the login screen as in Figure 3.1. The TOM's IP-address can be changed later on. After logging in with the predefined credentials the screen appears as shown in Figure 3.2.

3.1.2 Creating a company

At first a "company"-object has to be created, to store the essential structures like VPNs, (local) networks, users, TVDs, compartments and their relationships within a company or a project. In Figure 3.3, the company "TCLOUDS" is created via a right-click on "Management Console" and selecting "New".

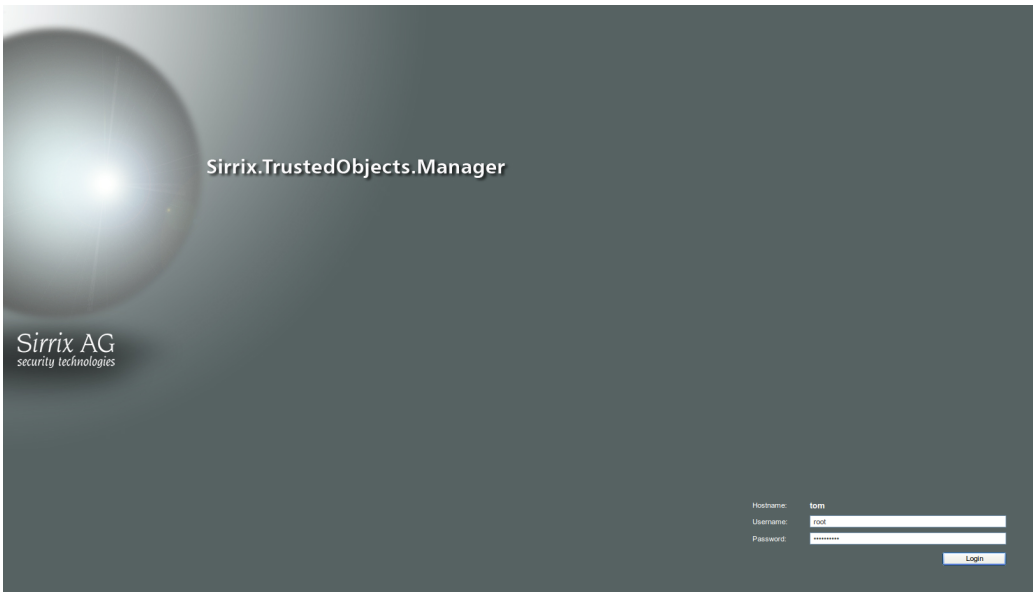

Figure 3.1: The TrustedObjectsManager Login screen

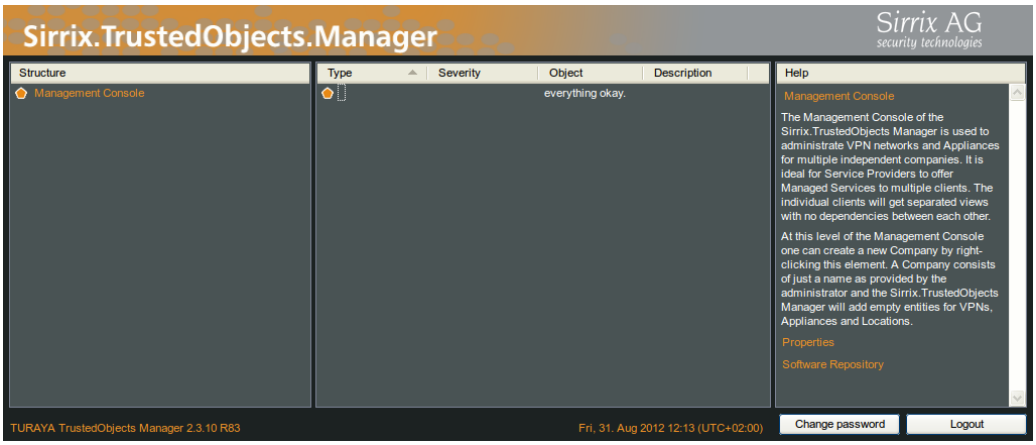

Figure 3.2: The TrustedObjectsManager overview screen after login

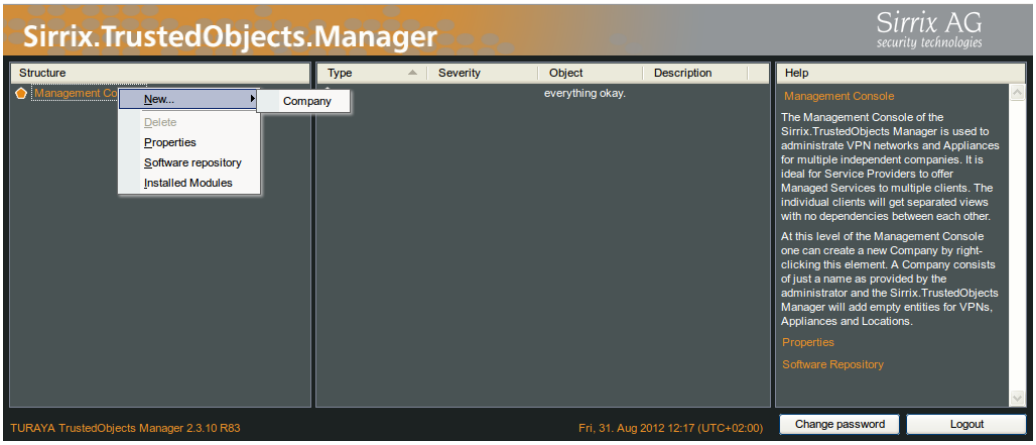

Figure 3.3: Creating a “Company”

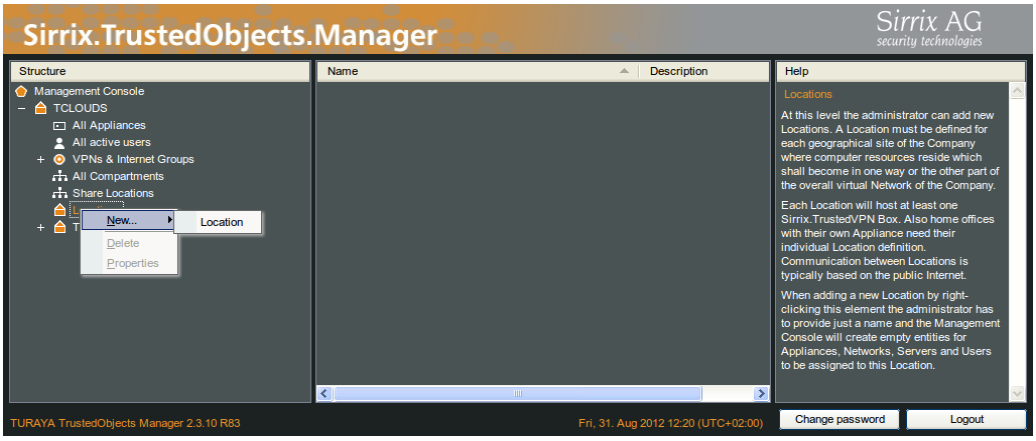

Figure 3.4: Creating a “Location”

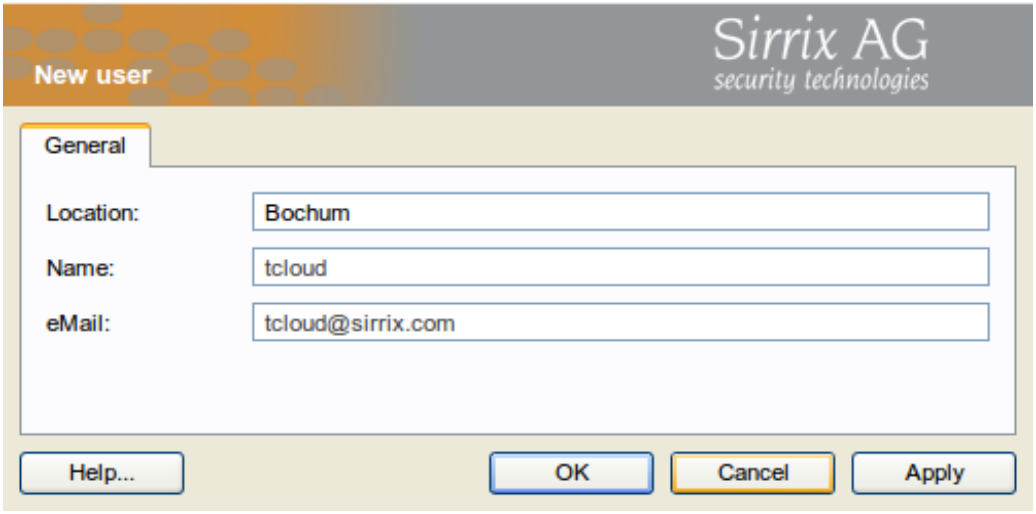

Figure 3.5: Adding a user, step 1

3.1.3 Creating a location

Within this newly created company one has to create a new location in order to logically sort different company branches. Here the location “Bochum” is created by expanding the sub-tree menu, selecting “Locations” and a right-click choosing “New...” and “Location”.

3.1.4 Adding users

Choose the newly created location “Bochum” and open the sub-tree by a left-click. The appearing “Users”-entry allows to add users to the infrastructure via a right-click selecting “New”, “User”. The form has to be filled out with a real-name and an existing email-address (Figure 3.5). After clicking “Apply”, three additional tabs appear, whereat the “Login Name”, a “Password” which has to be confirmed, and a “Hold-back time” has to be entered. The “Expiry Date” is calculated based on the value entered in the “Hold-back time” field (Figure 3.6).
Only those users entered in the “Users” sub-tree are granted access to the whole infrastructure.

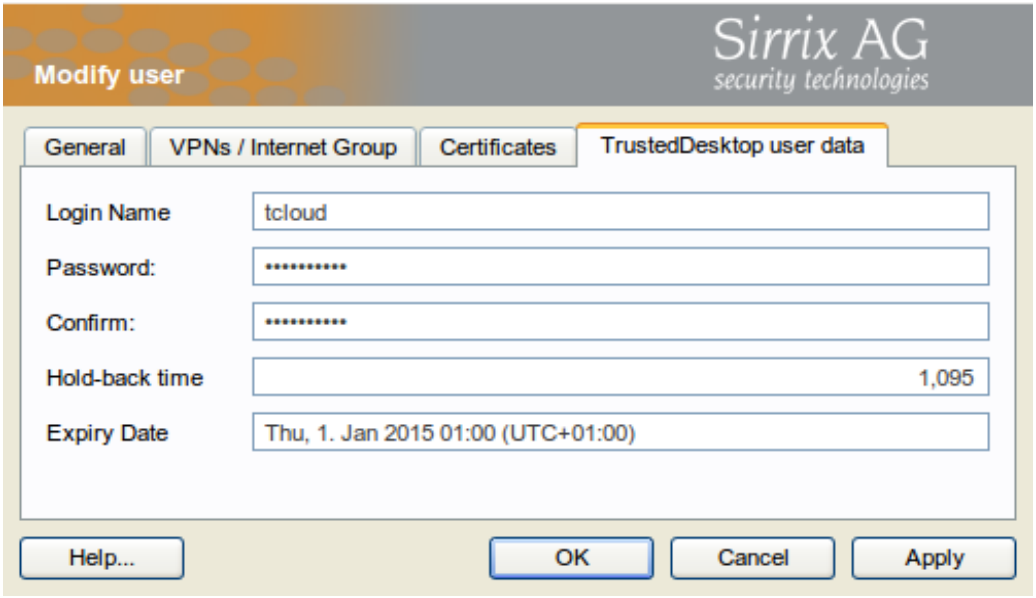

Figure 3.6: Adding a user, step 2

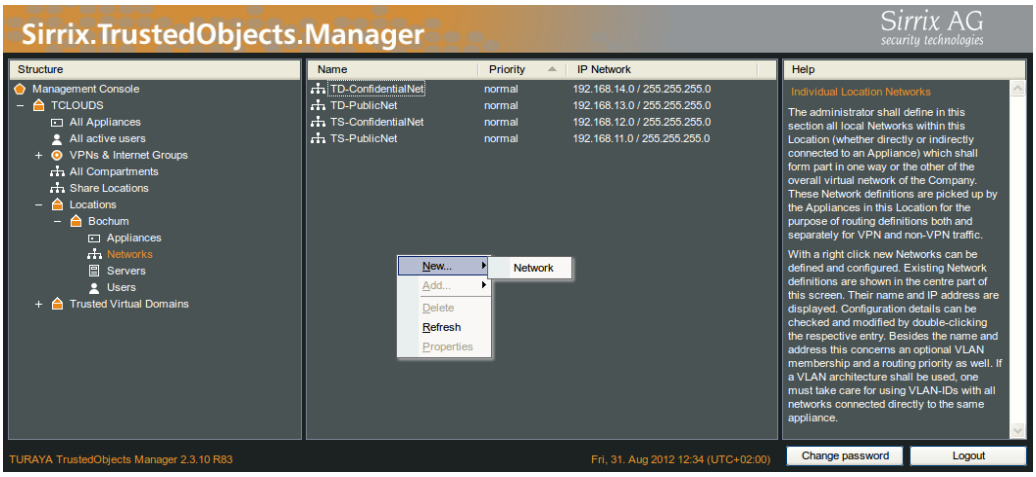

Figure 3.7: Adding networks

3.1.5 Network configuration

Within the newly created location, the local networks have to be defined, choosing “Locations”, “Bochum” and a right-click on ”Networks (Figure 3.7). Here, the 4 networks “TD-ConfidentialNet”, “TD-PublicNet”, “TS-ConfidentialNet” and “TS-PublicNet” are created with different IP-ranges. These networks i.e. IP-adress-ranges, will be attached to the virtual machines’ network interfaces running on the TrustedServer (TS) and on the TrustedDesktop (TD). Furthermore the “Confidential”-networks, as well as the “Public”-networks will reside within the “Confidential”- or “Public”-VPN respectively, which themselves will be part of the “Confidential” or “Public”-TrustedVirtualDomain (TVD). The assignment of networks to VPNs and TVDs ensures a separation of information-flow on the network level.

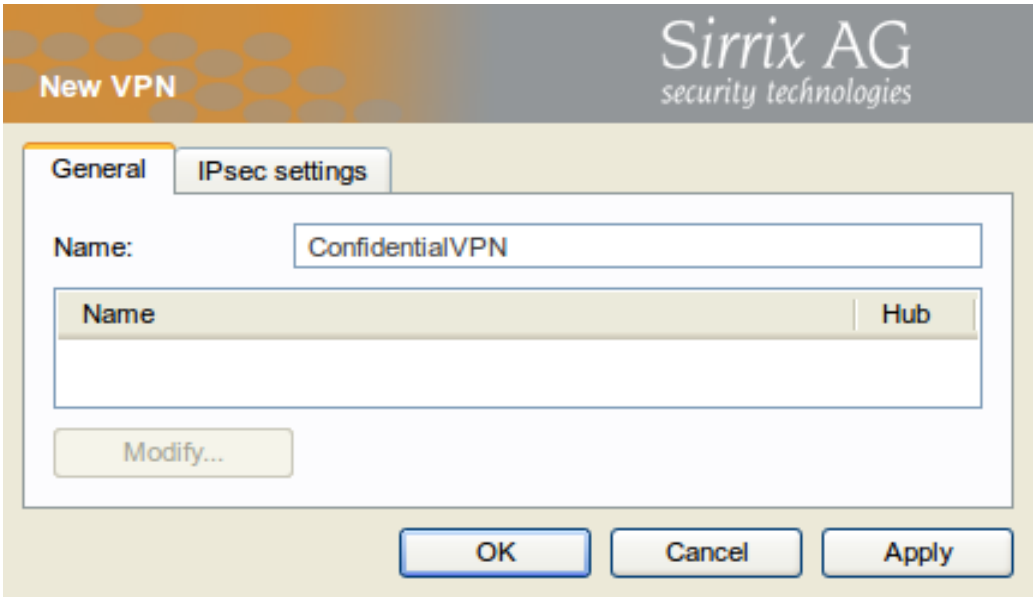

Figure 3.8: Adding a VPN

3.1.6 Creating and configuring VPNs

Logical VPNs are created by choosing "VPNs & Internet Groups", "New", "VPN". The appearing dialog (Figure 3.8) requires in this step just a name, here "ConfidentialVPN". The assignment of networks to VPNs (see: Section 3.1.5), takes place in a later step (see: Section 3.1.11).

3.1.7 Attaching appliances

Adding appliances like TrustedServer or TrustedDesktop to the company takes place within the sub-entry "Appliances" of a "Location". Right-clicking "Appliance" and choosing "New", "Appliance" opens a window like in Figure 3.9. The appliance has to be named, and the "Serial number" has to be entered. Here the TrustedServer is created. This 5x5 alphanumeric number has to gathered from the console of a newly installed TrustedDesktop or TrustedServer. The "Interface" field is greyed out and set statically to "eth0 - external network interface". This cannot be changed currently. If the appliance to attach is reachable via a static IP-address the "Static" radio-button has to be chosen and the appropriate fields "IP / Mask", "Gateway" and "DNS 1 / DNS 2" have to be filled. In case, the appliance gets its IP-address from a DHCP-server, the "Dynamically using DHCP"-option has to be chosen.
After clicking the "Apply" button, the "Download configurations / software update"-button will become active. By clicking this, a dialog opens, where the "configurations"-file can be downloaded. This file has to be stored on the root-folder of an USB-drive.
Attaching the USB-drive to the newly installed TrustedServer or TrustedDesktop, the configuration will be found and applied to this dedicated machine. Not till then, the appliance is able to connect to the TrustedObjectsManager, getting additional configuration settings after that the integration of the appliance to the infrastructure will be finished.

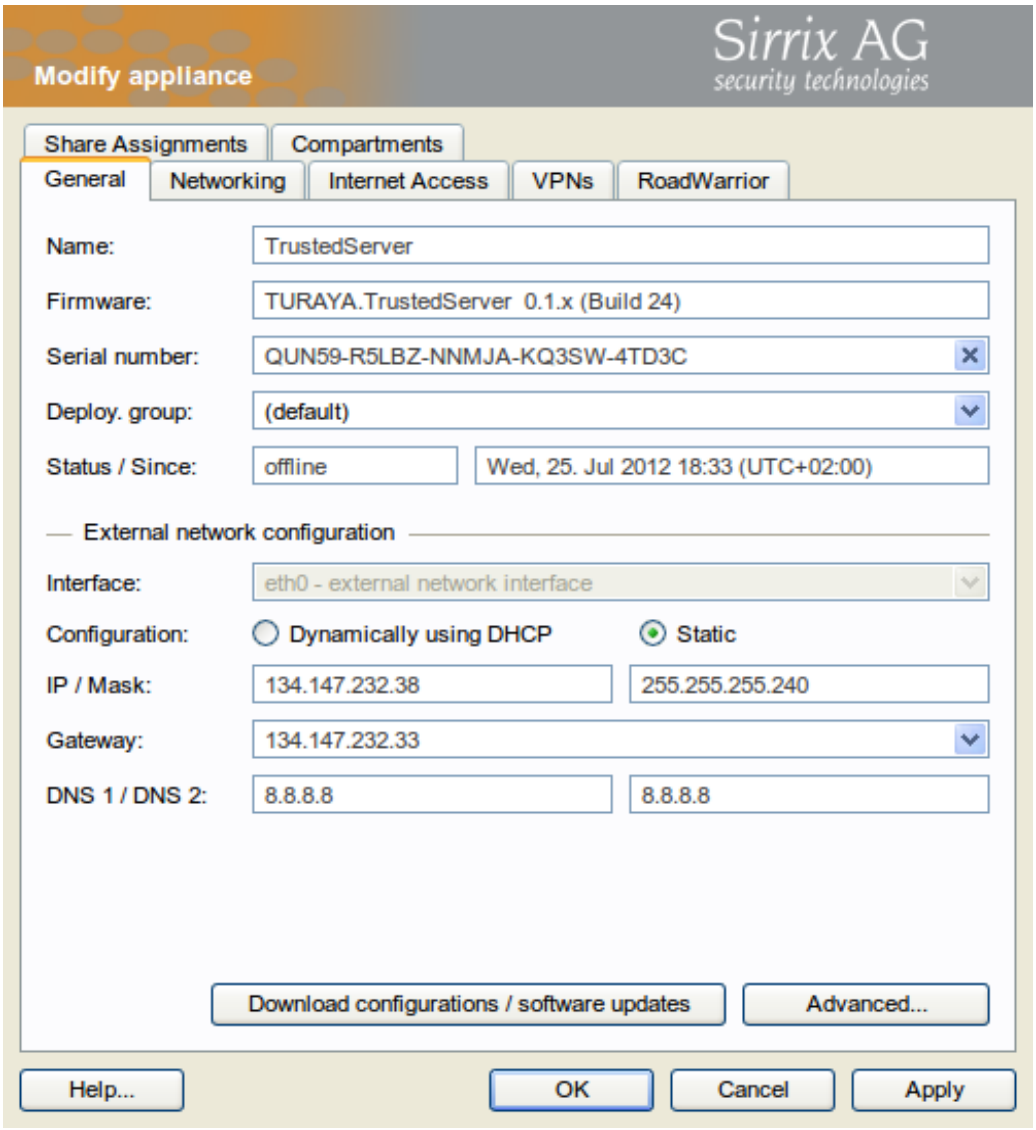

Figure 3.9: Add a new appliance to the company

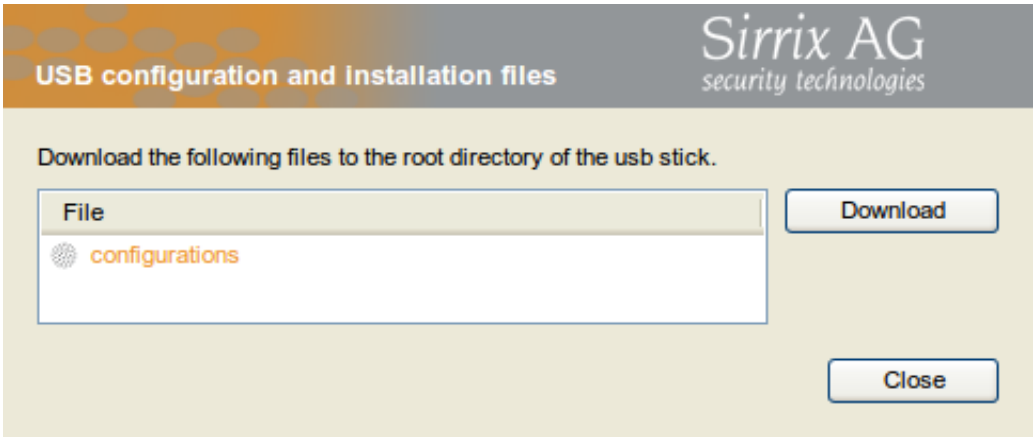

Figure 3.10: Dialog to download the configuration for the specific appliance

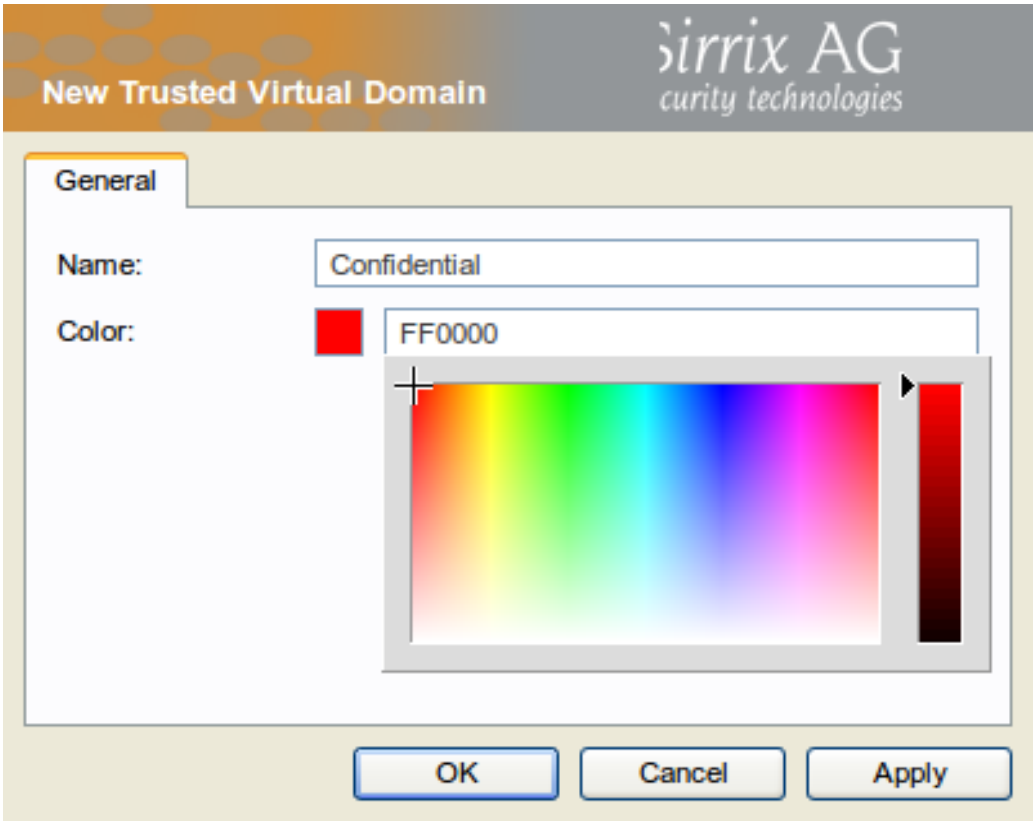

Figure 3.11: Dialog to create a new TVD

3.1.8 Creating TrustedVirtualDomains

To create a new TVD, "TrustedVirtualDomains" can be chosen from the context menu, right-clicking and choosing "New", "TrustedVirtualDomain". The TVD has to be named and a color has to be chosen from the color palette (Figure 3.11). The chosen color will appear as an visual border around the running compartment on TrustedDesktop. As shown in the Figure 3.11, in this scenario the TVDs "Confidential" and "Public" are created.

3.1.9 Adding compartments to TVDs

In order to add a virtual machine disk image to a TVD, one chooses for example "TrustedVirtualDomains", "Confidential", "Compartments" and right-clicks on "New". The new compartment screen (Figure 3.12) asks for the name, and a description of the compartment. The "Availability" checkbox has to be checked, in order to allow clients to download and use the compartment. Furthermore the virtual disc image template has to uploaded. If there are already files registered at the underlying database, one of these can be selected. Otherwise a new virtual disc image can be uploaded via the "Manage"-button. Press "Add local file", if the *.file is already on the TOM's harddisk or "Upload" in order to upload a *.vdi-file from remote.
An explaining comment of the image's content is optional. The "Last change" field in the "New compartment" window is automatically filled with the derived virtual disc image's timestamp.

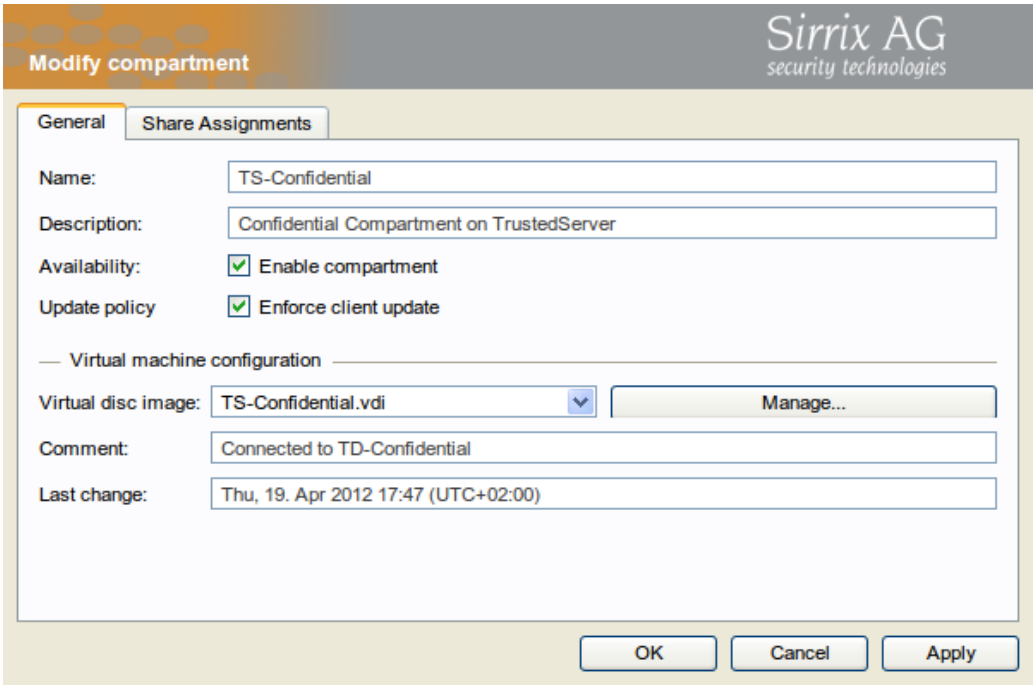

Figure 3.12: Adding a new compartment

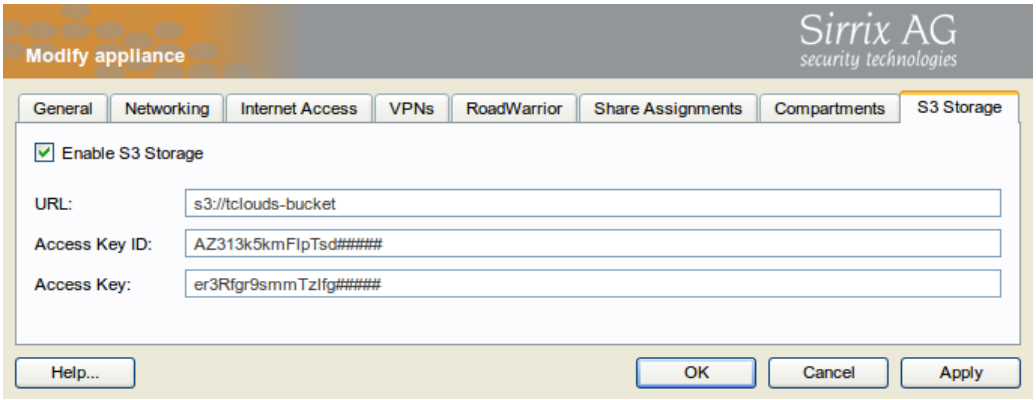

Figure 3.13: Attaching external S3-storage to TrustedServer

3.1.10 Adding external cloud storage to TrustedServer

It is possible, though not necessary to add external cloud storage to the TrustedServer. This storage can be accessed from within the compartments running on the TrustedServer natively, or can be provided to affiliated TrustedDesktops via a service (smb/sshfs/nfs). In all cases, files written to the attached cloud-storage (Amazon S3 storage as an example in figure 3.13) are transparently encrypted with a TVD-specific key, so the files can only be accessed from a TVD, they were initially written from.

From the ”Appliance” - ”Properties” submenu, the user chooses the S3 Storage tab and enters the URL to the S3-bucket, created via ”Amazon AWS Console” beforehand. The URL-pattern follows the form `s3://<BUCKET NAME>`. The backend login and password for accessing S3 are not the user id and password used to login into Amazon Webpage, but the AWS access key id and AWS secret access key provided by the users’ AWS ”MyAccount/Access Identifiers”. Clicking ”OK” results in an mount of the S3-bucket on TrustedServer.

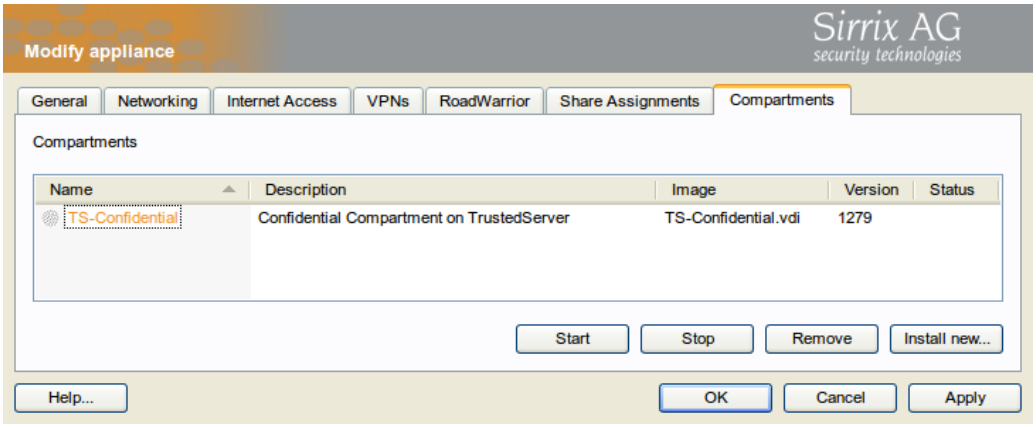

Figure 3.14: Installing a registered compartment to TrustedServer

3.1.11 Connecting everything together

The available compartments, uploaded to the TOM in Section 3.1.9 can now be installed to the TrustedServer by choosing the "Properties" of the appliance (Figure 3.14). Switching to the "Compartments"-tab allows the user to choose "Install new..." and select the desired compartment. The remote installation of the compartment starts immediately after a click on "OK" in case the TrustedServer is online.
Now, the corresponding network, defined in Section 3.1.5 has to be attached to the installed compartment. For that purpose, the registered appliance has to be modified via "Properties". The networking-tab, lists all installed compartments of the chosen appliance. Choosing one of them ("TS-ConfidentialNet" in Figure 3.15) leads to the selection of an "Interface", which is an entry like "tun1280 - internal compartment interface". All upcoming fields and checkboxes are automatically set to the values already defined during the step of creating networks. See Section 3.1.5. Pressing "Set" and "OK" applies all changes made to this tab. The installed compartment on the TrustedServer will now get a network connection after startup.
In order to separate the information-flow between different TVDs on different machines the VPNs are now set up finally. Dragging the TrustedServer-appliance from the list to the previously created "ConfidentialVPN"-entry in the "VPNs & Internet Groups" opens a window like in Figure 3.16. Checking "This appliance connects to all other appliances" is mandatory. Furthermore the "Policy" for the "Share"-networks have to be choosen, which is "IP forwarding (bidirectional)" in this case. Clicking "OK" closes the dialog and the setup is complete.

The procedure described in this section has to be repeated for the "Public"-TVD and the containing VPNs, networks and compartments.

The compartments can now be started, stopped or removed remotely by clicking the appropriate button in the "Compartments"-tab of an appliance (see Figure 3.14).

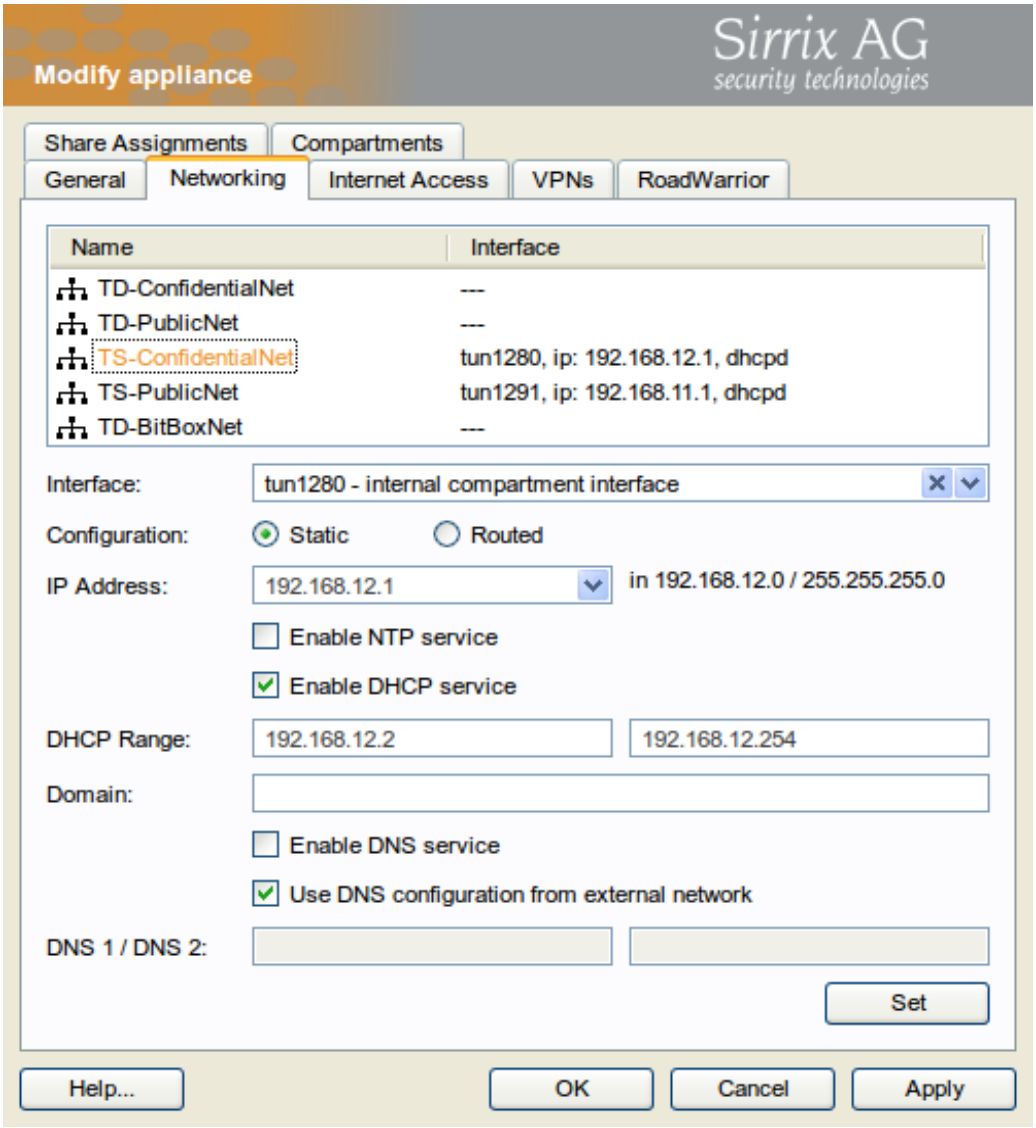

Figure 3.15: Attaching networks to compartments installed on TrustedServer

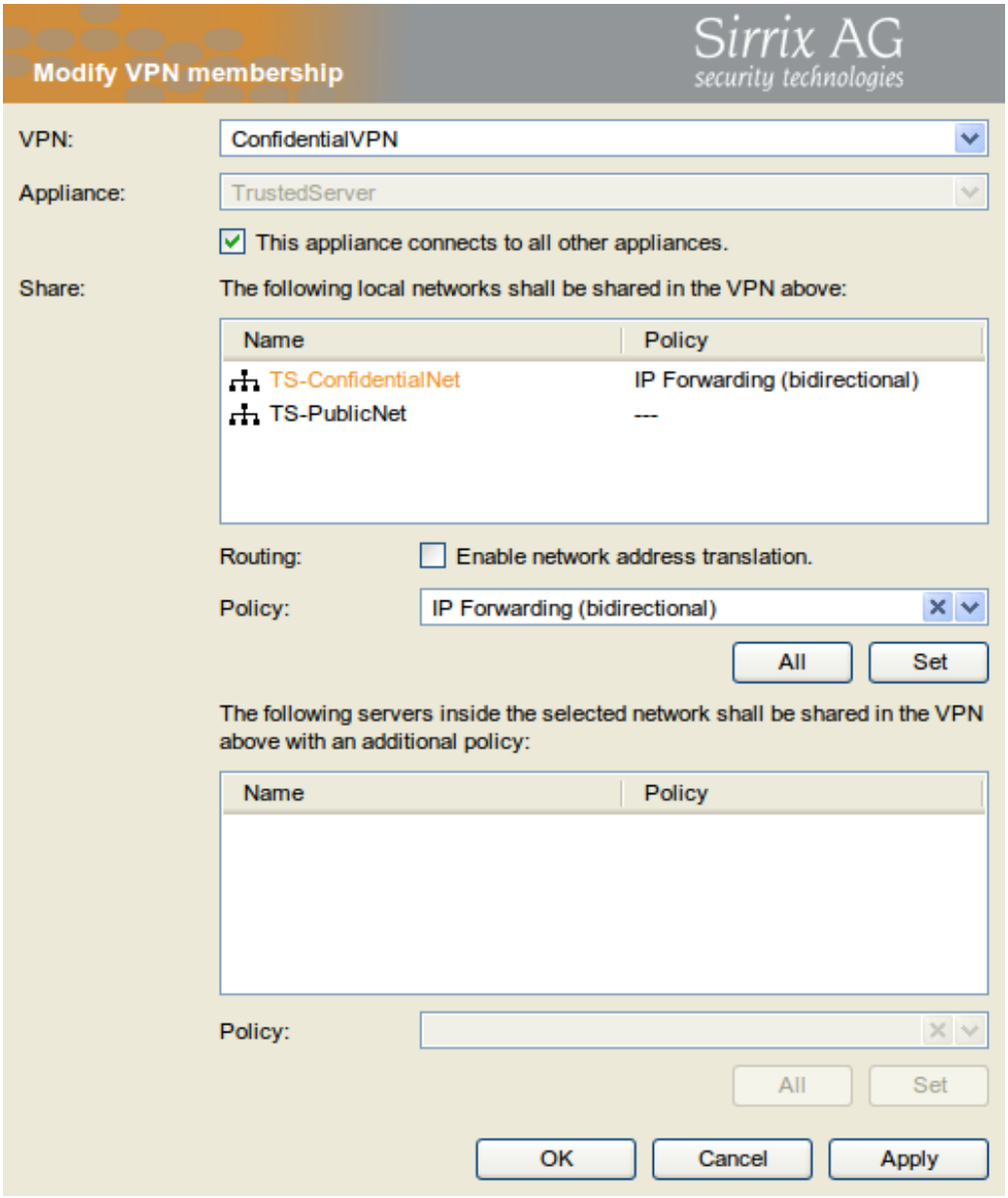

Figure 3.16: Editing VPN membership of TrustedServer

Bibliography

[AN12] Marco Abitabile and Marco Nalin. D3.3.3 - Validation Protocol and Schedule for the Smart Power Grid and Home Health Use Cases. Technical report, FSR, September 2012. TClouds deliverable.

[BACF08] Alysson N. Bessani, Eduardo P. Alchieri, Miguel Correia, and Joni S. Fraga. DepSpace: a Byzantine fault-tolerant coordination service. In *Proc. of the 3rd ACM European Systems Conference – EuroSys'08*, pages 163–176, April 2008.

[BGJ+05] Anthony Bussani, John Linwood Griffin, Bernhard Jansen, Klaus Julisch, Genter Karjoth, Hiroshi Maruyama, Megumi Nakamura, Ronald Perez, Matthias Schunter, Axel Tanner, and et al. Trusted virtual domains: Secure foundations for business and it services. *Science*, 23792, 2005.

[ea11a] Alysson Bessani et al. D2.2.3 Proof-of-concept of Middleware for Adaptive Resilience. Technical report, FFCUL et al., September 2011. TClouds deliverable.

[ea11b] Christian Cachin et al. D2.3.1 - Requirements, Analysis, and Design of Security Management. Technical report, IBM et al., October 2011. TClouds deliverable.

[ea11c] Emanuele Cesena et al. D2.4.1 - Clouds Prototype Architecture, Quality Assurance Guidelines, Test Methodology and Draft API. Technical report, Politecnico di Torino et al., September 2011. TClouds deliverable.

[ea11d] Marcelo Pasin et al. D2.2.1 - Preliminary Architecture of Middleware for Adaptive Resilience. Technical report, FFCUL et al., October 2011. TClouds deliverable.

[ea12a] Alysson Bessani et al. D2.2.2 - Preliminary Specification of Services and Protocols of Middleware for Adaptive Resilience. Technical report, FFCUL et al., September 2012. TClouds deliverable.

[ea12b] Norbert Schirmer et al. D2.1.2 - Preliminary Description of Mechanisms and Components for Single Trusted Clouds. Technical report, SRX et al., September 2012. TClouds deliverable.

[ea12c] Soẗen Bleikertz et al. D2.3.2 - Components and Architecture of Security Configuration and Privacy Management. Technical report, IBM et al., September 2012. TClouds deliverable.

[Gel85] David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

[GHSS11] Ruediger Glott, Elmar Husmann, Matthias Schunter, and Ahmad-Reza Sadeghi. D1.1.1 - Draft Scenario and Requirements Report. Technical report, UMM et al., April 2011. TClouds deliverable.

[GVM00] Garth A. Gibson and Rodney Van Meter. Network attached storage architecture. *Communications of the ACM*, 43(11):37–45, November 2000.

[KS12] Anil Kumar and Jerry St.Clair. A Unit Testing Framework for C. Retrieved from: http://cunit.sourceforge.net/, September 2012.

[Lev12] Peter Levart. FUSE-J: A Java binding for FUSE. Retrieved from: http://sourceforge.net/projects/fuse-j/, 2012.

[Mil12] Stewart Miles. A lightweight library to simplify and generalize the process of writing unit tests for C applications. Retrieved from: http://code.google.com/p/cmockery/, September 2012.

[MT09] Di Ma and Gene Tsudik. A new approach to secure logging. *Trans. Storage*, 5:2:1–2:21, March 2009.

[NF12] Gergely Nagy and Zoltán Fried. CEE-enhanced syslog() API. Retrieved form: https://github.com/deirf/libumberlog, September 2012.

[OvT12] Open vSwitch Team. Open vSwitch. Retrieved form: http://openvswitch.org/, September 2012.

[SK99] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur.*, 2:159–176, May 1999.

[SV12] Paulo Santos and Paulo Viegas. D3.2.3 - Smart Lighting System Draft Prototype. Technical report, EFACEC ENG, September 2012. TClouds deliverable.