ConfigImporter

A new component for the VeriNeC project

Geraldine Antener

Master Thesis in Informatics
University of Fribourg, Switzerland

Supervisors:
Prof. Ulrich Ultes-Nitsche
David Buchmann
Dominik Jungo

telecommunications, networks & security Research Group
Department of Computer Science

July 2005

Abstract

ConfigImporter is part of the VeriNeC project. VeriNeC aims to simplify the complex task of configuring a network. The core of VeriNeC is an abstract description of a network and its devices using XML. This abstract configuration can then be tested by means of simulation and translated to implementation specific configuration data. ConfigImporter imports the implementation specific configuration settings into the environment of the VeriNeC project. This paper shows how Linux (more precisely the distribution Fedora Red Hat) stores network configuration, and how these settings can be translated into VeriNeC's XML language.

Contents

Contents

1 Introduction

Network configuration is a complex task, particularly when the network is heterogeneous and different implementations of services are used. The telecommunications, networks & security Research Group of the University of Fribourg develops a tool to make this task easier: the VeriNeC (Verified Network Configuration) project. VeriNeC defines an abstract description of a network using XML. With this definition the abstract network can be simulated and translated to real configuration data.
For practical use, it should also be possible to do the opposite: import implementations specific configuration settings into the VeriNeC environment.
This master thesis shows how such an import process can be done for the services Ethernet card setup and iptables Firewall setup of the Linux distribution Fedora Red Hat.

1.1 Overview of this paper

This paper is part of my master thesis in Computer Sciences; it describes the research for and the implementation of ConfigImporter, a new component for VeriNeC. It is structured as follows:
In this chapter the thesis' objectives are declared which gives an impression of its process.
A second chapter gives a short introduction to the VeriNeC project and its relations to ConfigImporter.
The third chapter explains the theoretical background needed to understand how to import the configuration settings. The configuration files of Red Hat and the VeriNeC's network definition structure are analysed.
A further chapter shows how ConfigImporter is implemented.
The last chapter gives a conclusion from the master thesis and shows the limits of ConfigImporter.

1.2 Objectives of the project

The final goal of the thesis is to write a new component, a ConfigImporter, for the VeriNeC project (see Chapter 2). The ConfigImporter imports the configuration of a Linux machine, i.e. it analyses the existing configuration files of a computer and translates them into abstract configuration data.
This can be divided into the following objectives:

- Research on configuration files used in Linux systems. Find out which files contain the needed information, where they are stored in the file system and what syntax they have.
- Define the structure of these files and research on existing parsers for these formats. If there is no useful parser, write a custom parser.
- Implement ConfigImporter, a translator to transform the files into the VeriNeC XML format.

2 VeriNeC and ConfigImporter

VeriNeC [1] is a project of the telecommunications, networks & security Research Group of the Department of Computer Science at the University of Fribourg and is supported by the Swiss National Science Foundation.
VeriNeC aims to simplify the complex task of configuring a network. The core of VeriNeC is the abstract description of a network; its topology and the configuration of its components are defined using XML. This abstract configuration can then be tested by means of simulation, translated to implementation specific configuration data and distributed on machines. The VeriNeC system is in the progress of being developed. [2]
ConfigImporter is the name of the master thesis at hand, and the working name for a new component implemented for VeriNeC. ConfigImporter provides VeriNeC with the ability to import the network configurations of existing systems, that is to analyse its configuration settings and create the abstract configuration.

2.1 VeriNeC

In a heterogeneous network, with several operating systems, diverse implementations of services and different ways to set their properties, network configuration can be complex. With VeriNeC, a system administrator's life becomes easier. The project centralises his work and unifies the description of configurations.
The base of VeriNeC is its definition of an abstract network description. This is a level of abstraction and describes a service's functionality without regarding its implementation. E.g. a packet-filtering firewall can be reduced to lists of tests on different properties of IP packets.
These network descriptions are defined using XML according to the XML Schema described in the next section.
Once an abstract network definition is written, it can be distributed to any system for which a translator exists. Implementations can be replaced and reconfigured without the slightest effort. This can make the network configuration task more concise.
Before distributing the network settings, they can be tested and verified, this will augment the security of the system. [3]

VeriNeC consists of different modules: the simulator, the translator and the editor. Figure 1 shows the correlations.

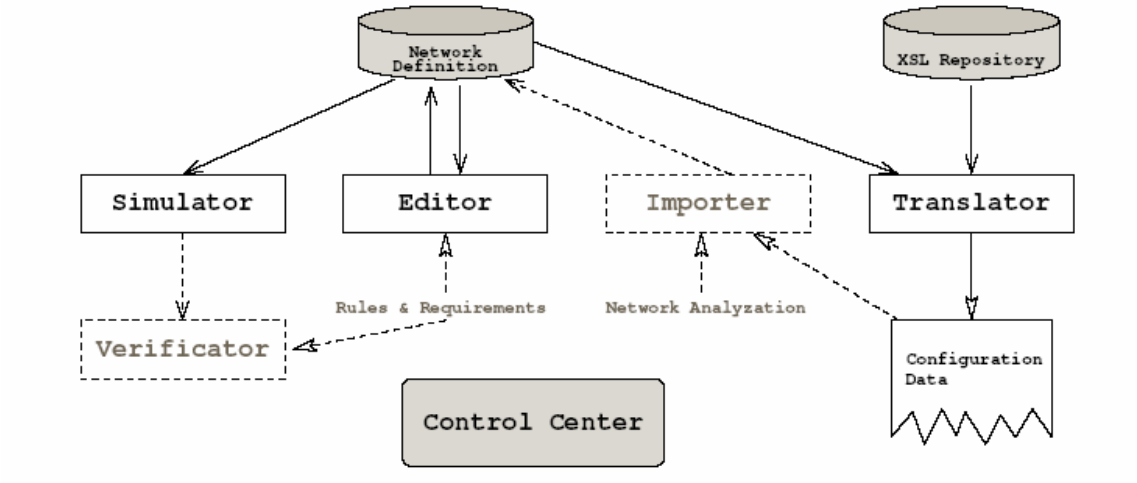


Figure 1: The modules of VeriNeC

The Simulator module imitates the behaviour of the network defined in XML. It can test a configuration and give feedback to the network designer.
The Translator module generates the implementation dependant configurations out of an abstract definition. In a second step, it distributes the generated configuration to the network devices.
A graphical user interface (GUI), the Editor, supports the user to create an abstract network. [2, 3] Each of the modules too has a GUI.
The new module Importer contains the Network Sniffer[1] to analyse the traffic on a network, and the ConfigImporter to analyse the concrete setups of a machine. Both create an abstract definition: Network Sniffer generates it from the recorded traffic, ConfigImporter form the analysed settings. The two can also complement one another: first sniff the network, and then import the found nodes.

The network definitions can be stored in and loaded from a repository.

[1] Network Sniffer is a Bachelor thesis within the VeriNeC project written by Patrick Aebischer [13]

2.2 Network Definition Schema

The abstract configuration of the network and its nodes in the VeriNeC project are defined using XML. The structure of these files is defined by the XML schema *network.xsd* and *node.xsd*[1].

A *network.xml* file is created for a network to describe the connections of its nodes' interface bindings. For each node the definitions of its services are stored in a *node.xml* file. The XML document describing a node (that is desktop computer or any other network element) has two elements: a hardware part and a services part.

Figure 2 shows the structure of the XML schema *node.xsd* (Not every element is described in detail.)

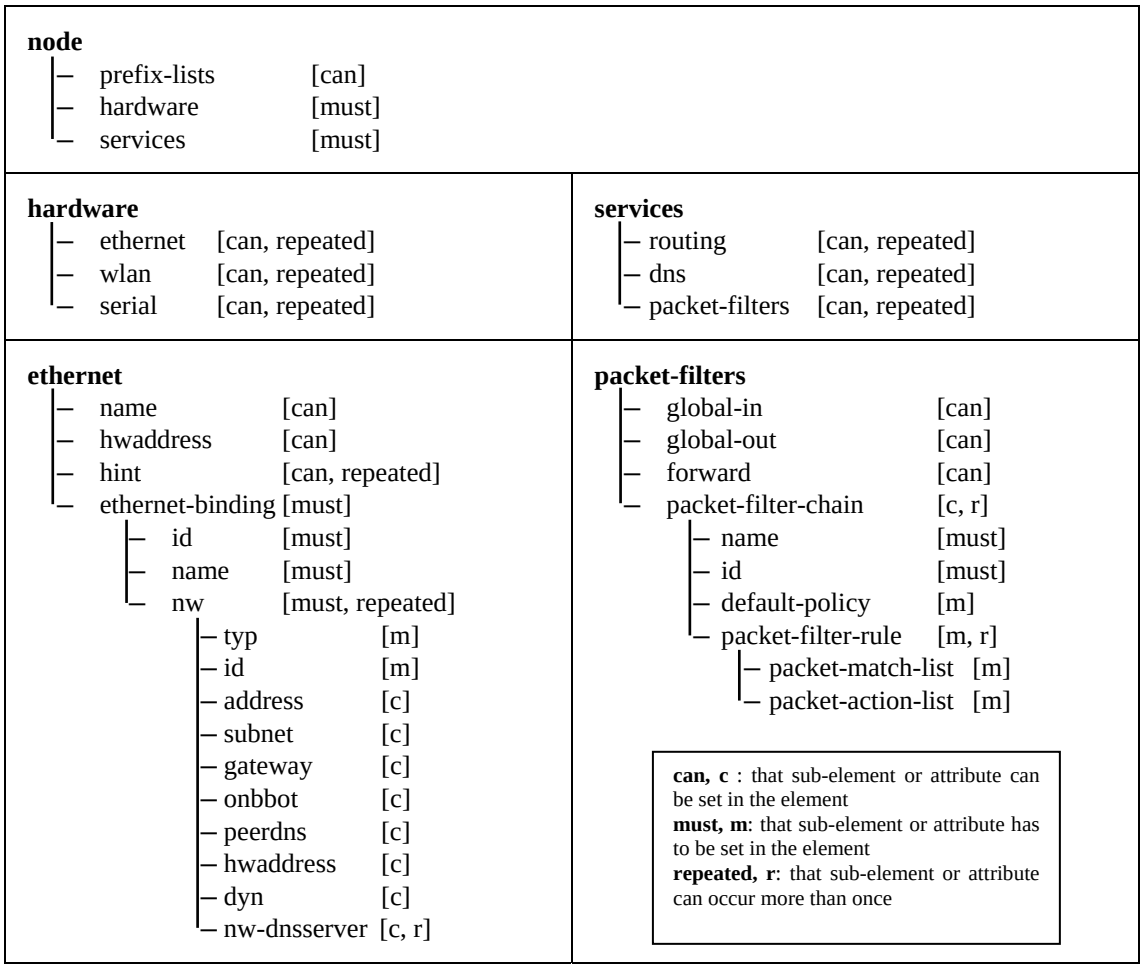


Figure 2: Elements defined in node.xsd with their attributes and sub-elements

[1] full namespace: http://diuf.unifr.ch/tns/projects/verinec/network, respectively .../node

The hardware element defines all network interfaces (Ethernet, WLAN or serial interfaces) of the system. Inside an interface element one or more bindings are defined (only one for Ethernet interfaces). These bindings can be connected in the *network.xml* file. Bindings can have several attributes as the name and the hardware address of the interface.
Also a <hint/> element can be added, to assign explicitly a hardware slot to the abstract interface. This can be important when the settings are distributed to a machine.

For each <ethernet-binging> element one or more network, <nw/> elements, are defined. A <nw/> element contains items like the IP-address and DNS server settings. If an interface has more than one <nw/> element, attributes as the IP-address can be set twice. So an Ethernet card for example can listen to more then one IP-address while not being physically connected more than once.

The services element currently supports the services routing, DNS and packet-filtering.
In the routing part route filters, static and dynamic routes can be set.
The <dns/> element defines several <Zone/> elements each defining a DNS Zone.
The <packet-filters/> element contains a list of packet filter chains, each chain defines packet filter rules.

2.3 ConfigImporter

The idea of ConfigImporter is to import the configurations of running systems into the VeriNeC environment. This can be of practical use: a system administrator can test the configurations with VeriNeC's Simulator module, or an established configuration can be distributed on another system with a different distribution or operating system via the Translator module of VeriNeC; ConfigImporter extends the possibilities of VeriNeC.

To import means to translate implementation dependent configuration files into XML data satisfying the VeriNeC's XML schema definition *node.xsd.*
In this thesis, I implemented the retrieval of the hostname, the Ethernet interfaces and packet-filter configurations for UNIX-like operating systems. Further I analysed how to import the DNS configuration settings, this is not yet implemented. Other configurations are currently ignored.

ConfigImporter is only tested for the Linux distribution Fedora Red Hat. Linux distributions, that is operating systems containing a Linux kernel plus open source software, can be divided in groups by the packet management system used. Fedora Red Hat uses the RPM (Red Hat Package Manager) which is widely used; distributions as Mandriva Linux, Novell Linux Desktop and SuSE are also of that type.

Other distributions are the distributions of the Debian family like Debian, Ubuntu and Knoppix and distributions of other types like Slakeware, Gentoo and Linux From Scratch.
ConfigImporter can be applied for systems using iptables[1] (to import the packet-filter configurations) and for distributions with the same interface configurations file structure as Fedora Red Hat (to import the Ethernet interfaces settings).

The ConfigImporter is integrated in the VeriNeC's editor module and can be started from its GUI.

[1] ConfigImporter is only tested for the iptables version 1.2.9

3 Network Configuration in Linux Red Hat

To set up services at boot time and to maintain the configuration information over rebooting the system, the Linux distribution Red Hat provides configuration files for its services. Most of these configuration files are stored in the */etc/sysconfig/* directory. Besides files to configure the mouse and the keyboard, there are files to set up network services: [5]

- The information about network configuration is stored in the */etc/sysconfig/network* file.
- A configuration file for each network interface is in the */etc/sysconfig/network-scripts/* directory.
- Information to set up packet filtering services is located in the */etc/sysconfig/iptables* file.
- The file for the DNS named daemon is the */etc/sysconfig/named.conf* file[1]

Once a service is established, the information stored in the files can often also be extracted from the corresponding commands. This has the advantage to be distribution independent, but can have several disadvantages, such as that not all information is obtained.

3.1 Configuration of network interfaces

To add a machine to a network, its network interface has to be configured. This is done with the `ifconfig` command. This command applies to a hardware interface, assigns an address, sets other parameters, and enables or disables the interface. The command can also display the current settings of the interfaces. [4]
To bring up the interfaces the corresponding scripts call the `ifconfig` command with the parameters given in the configuration files. Changes of the parameters are also saved there. Under Red Hat Linux the configuration files for network interfaces are in the */etc/sysconfig/network-scripts/* directory. Each interface has its own file, a *ifcfg-<name>* file, where *name* is the name of the interface. These names are composed of the type of the interface followed by a number. Types are "eth" for Ethernet, "wlan" for WLAN or "lo" for the loopback interface and others.

[1] But zone, statistic, and cache files are in the */var/named/* directory

The Ethernet configuration files control the network interface cards (NIC). For example, the first Ethernet card is configured with the *ifcfg-eth0* file. Listing 1 shows such a file. If there is more than one NIC, there is a further file for each NIC. [5]

In these files parameters as the IP address, the subnet mask and if the corresponding interface has to be brought up at boot time are stored. Changes of these parameters can also directly be done in these files.

```
# ifcfg-eth0
DEVICE=eth0
BOOTPROTO=dhcp
HWADDR=00:06:5B:A9:EF:60
ONBOOT=yes
TYPE=Ethernet
```

Listing 1: The configurations file ifcfg-eth0

Special types of interface configuration files are alias files. Alias files can be used for example to assign more than one IP-address to one hardware network interface.
Interfaces having an alias file can be addressed by their or by the alias name and have two configuration files.
The alias files are named *ifcfg-<if-name>:<alias-value>*, for example ifcfg-eth0:0. [5]

The loopback interface is a special, not a real hardware, interface. It corresponds to a fictitious network. Any data sent through that interface is sent directly to the input queue of the very computer. The default values in this file shouldn't be changed. [4]
This interface has also its configuration file (*ifcfg-lo*) stored in the */etc/sysconfig/network-scripts/* directory. Although it is not an Ethernet interface it can be imported by ConfigImporter as an extra option. This is because *ifcfg-lo* it is an important file to set up a network and has the same structure as an Ethernet configuration file.

3.1.1 Structure of network interface configuration files

The network interface configuration files of Red Hat Linux are of a simple style: INI-style.
INI-style files consist of sections; a section has a section name written in square brackets, followed by option lines. In option lines a value is assigned to a variable. Additionally the files can have comment lines and blank lines. [6]
Listing 2 shows that structure and an example of an INI-style file.

Structure of INI-style files:

```
[section]
var = value
```

Example:

```
#List of all users

[Jill]
role = administrator
last_login = 2005-02-03

[Tim]
role = user
last_login = 2003-01-21
```

Listing 2: Structure of INI-style files

The network interface configuration files contain only one section and don’t have section names. They are therefore only lists of assignments of values to certain options. Options in these files are among others "DEVICE", "ONBOOT" and "IPADDR". The File */usr/share/doc/initscripts-<version-number>/sysconfig.txt* lists and explains all configurable parameters of interface configuration files.

See again Listing 1 as an example for an INI-style interface configuration file.

3.1.2 Get the information for a <ethernet/> element

To import the configuration of the network interfaces of a system into the VeriNeC environment, a <hardware/> XML element has to be defined. According to the XML Schema *node.xsd* that element needs different parameters for the different interface types.
The values needed to generate a <ethernet/> element for a NIC are found in the configuration file of that interface. Most of the information could also be extracted with the ifconfig command, but not all (it doesn't tell whether a static IP-address is used or if it is assigned with DHCP). Thus ConfigImporter uses the information found in the configuration files.

Each main Ethernet configuration file translates to one <ethernet/> element. And each <ethernet/> element contains exactly one <ethernet-binding/> element. If an interface has one or more alias files, the element has an additional <nw/> element for every alias file. Figure 3 shows what information is used for which part of the element.

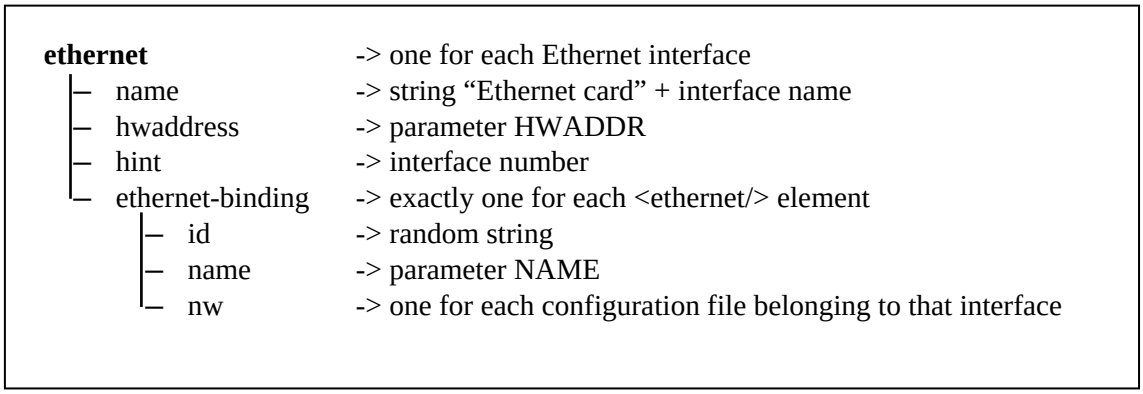


Figure 3: Information to generate a <ethernet/> element

The information to set the attributes of the <nw/> element are found in the configuration file. The values of the parameters can be used directly or with slight changes as values of the attributes.
Table 1 is a mapping of the attributes that can or must be set in a <nw/> element of an interface, and the corresponding names of the parameters in the configuration file.

Content in the <nw/> element	**Parameter in the configuration file**
Attribute "address"	IPADDR
Attribute "subnet"	NETMASK
Attribute "gateway"	GATEWAY
Attribute "onboot"	ONBOOT
Attribute "peerdns"	PEERDNS
Attribute "hardware"	HWADDR
Element <dyn/> Attribute "type" Attribute "retry" Attribute "timeout"	 BOOTPROTO MAXFAIL RETRYTIMEOUT
Element <nw-dnsserver/> Attribute "ip"	 DNS1 or DNS2
Attribute "type"	test of IPX- Parameters

Table 1: Mapping attributes of <nw/> elements to cofig file parameters

The <ethernet/> element can additionally contain the hardware address of the corresponding network interface card (NIC). The <ethernet-binding/> element has an name attribute to be set. These values are also found in the configuration files: the values of HWADDR and NAME (see Figure 3).

Now that all information is known, an <ethernet/> element can be generated. The <ethernet/> XML element given in Listing 3 is the result of transforming the configuration file ifcfg-eth0 of Listing 1.

```
<vn:ethernet name="Ehernet card eth0"
                hwaddress="00:06:5B:A9:EF:60">
    <vn:hint system="pc" slot="0" />
    <vn:ethernet-binding id="if1" name="eth0">
        <vn:nw id="i1" onboot="yes" type="ip">
            <vn:dyn type="dhcp" />
        </vn:nw>
    </vn:ethernet-binding>
</vn:ethernet>
```

Listing 3: The corresponding <ethernet/> element to the config file ifcfg-eth0

3.2 Configuration of packet filtering firewalls

Transmitting data on a network is done in packets. The commonly used protocol pair (in the Transport and Network layers[1]) is TCP/IPv4 (the Transmission Control Protocol and the Internet Protocol version 4). Packets in these protocols contain user data and a so called header which holds extra information how to handle the packet addressing (routing, get back user data etc.). [7]
This framework allows packet filtering, i.e. analysing the header of an incoming or outgoing packet and decide what to do with that kind of packet, whether to let pass or to drop it. This procedure can increase the security and availability of the system. [8]

3.2.1 The netfilter framework

Packet filtering in Linux is done with the netfilter framework. Netfilter is a kernel tool of the Linux kernel 2.4. Netfilter defines hooks, which are points in the way of a packet through the kernel. The number and position of the hooks are different for every protocol: In the IPv4 there are five hooks defined:

- PREROUTING [1]: for all incoming packets before routing.
- POSTROUTING [4]: for all outgoing packets after routing.
- FORWARD [3]: between routing, for packets not destined to and not coming from a local process.
- INPUT [2]: after routing before entering user-space
- OUTPUT [5]: for packets coming from a local process before routing.

The diagram from Linux Netfilter Hacking HOWTO shows this traversal (Figure 4). [9]

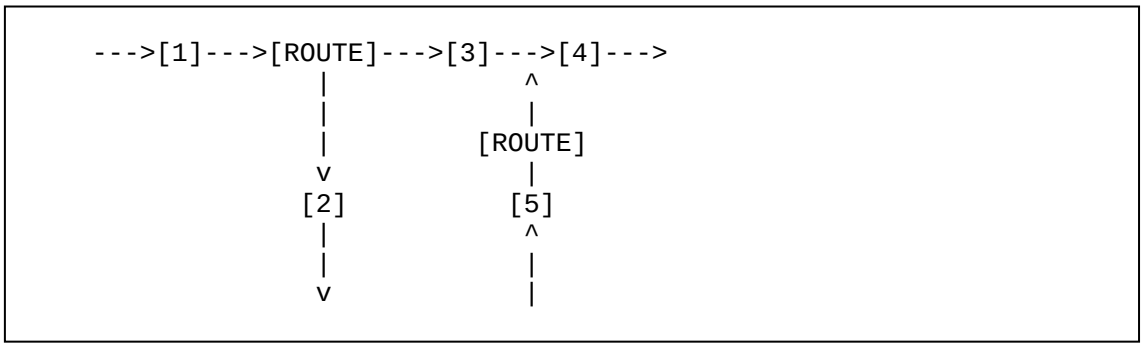


Figure 4: A packet traversing the netfilter system

[1] Layers 3 and 4 in the OSI model, see [14] for more information

3.2.2 The iptables command

The corresponding user-space program to configure the filter settings of the kernel is iptables. For every hook there is one (or more) chain defined. A chain defines the rules to check when a packet is on a certain point in its travel through the kernel.
Iptables, as the name suggests, consists of three tables: the *filter*, the *nat* and the *mangel* table. This tables group the chains into modules, which can by loaded separately.
The filter table contains the INPUT, the OUTPUT and the FORWARD chains. With the rules set in these chains, the user tries to control the access to local processes, and to protect the destination network. The rules only filter packets, i.e. they only decide whether to let pass a packet on the hook or to drop it, they don't alter the packets. [10]
The nat table has three built-in chains: PREROUTING, POSTROUTING and OUTPUT. The rules in the nat table are doing the network address translation. This can concern the source or the destination. Forms of NAT are masquerading, port forwarding, load sharing, and transparent proxying. [11]
In the mangle table are the chains PREROUTING and OUTPUT. These rules are changing special packet parameters.

The real filtering of packets in its pure meaning is done with the chains of the iptables filter table. The features of the nat and mangle tables are not supported by the XML schema *node.xsd* and ConfigImporter therefore only imports the filter table.

With the iptables command the user can control the settings for filtering network packets in the chains. Iptables allows creating or deleting rules for a chain, defining the match conditions for each new rule and also specifying what to do with a matching packet (i.e. setting the so called target). The whole possibilities of iptables can be found in its man page.
The settings of the tables can be stored in the */etc/sysconfig/iptables* file, the rules set there will be reapplied at boot time. In Red Hat this file contains primarily a list of the iptables command lines previously executed without the "iptables" string at the beginning of each line. [5]

3.2.3 The filter table

Packets enter the filter table when they are on one of the three hooks, INPUT, OUTPUT, or FORWARD (see The netfilter 3.2.1 above). They enter the corresponding chain in the table. Each chain is a list of rules. The rules say what to do (target) with a packet that has certain parameters set in its header. These two parts of a rule are called target and match list.
The packet gets tested sequentially on every rule of the chain until it matches with one. Then the defined target of the rule is performed.

Possible targets are the built-in targets ACCEPT and DROP, special targets as RETURN, LOG or REJECT or a user-defined chain.
User-defined chains are also lists of rules set by the user and are called by their name.
If a packet reaches the bottom of a user-defined chain without matching any rule, it returns to the calling chain and continues testing the rules. If it reaches the bottom of a built-in chain, it is the chains default policy which is executed. Default policies can only be one of the targets ACCEPT and DROP.
The match list contains tests whether certain bits in the header are set or not. They can specify the protocol of the packet, the bits of the source and destination address and lots of other header parameters. [12]

3.2.4 Structure of the iptables output

The */etc/sysconfig/iptables* file could be used to generate the abstract configuration. But the place and format of this file is distribution dependant, the importation would only work for the Red Hat distribution. Another possibility to get the needed information is with the iptables command itself. The command line "`iptables –Lvn`" lists all chains and rules in the filter table (the table name could also be given explicit, but filter is default).
Listing 4 shows how the output of this command could look like.

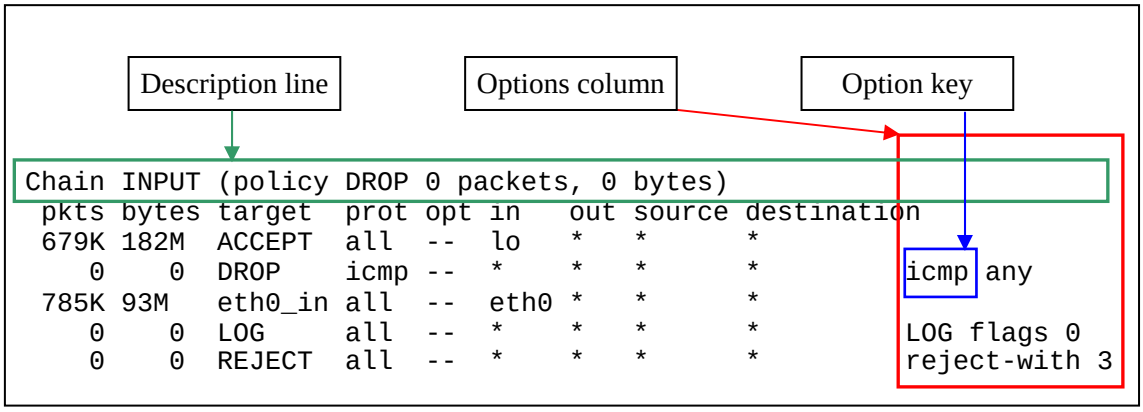
```
[verinec@iiufpc33 sysconfig]$ sudo /sbin/iptables –Lvn

Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target  prot opt in   out source    destination
 679K 182M  ACCEPT  all  --  lo   *   0.0.0.0/0 0.0.0.0/0
    0    0  DROP   !icmp --  *    *   0.0.0.0/0 0.0.0.0/0     state INVALID
 785K 93M   eth0_in all  --  eth0 *   0.0.0.0/0 0.0.0.0/0
    0    0  LOG     all  --  *    *   0.0.0.0/0 0.0.0.0/0     LOG flags 0
    0    0  REJECT  all  --  *    *   0.0.0.0/0 0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target  prot opt in   out source    destination
    0     0 DROP   !icmp --  *    *   0.0.0.0/0 0.0.0.0/0     state INVALID
    0     0 LOG     all  --  *    *   0.0.0.0/0 0.0.0.0/0
    0     0 REJECT  all  --  *    *   0.0.0.0/0 0.0.0.0/0

Chain OUTPUT (policy DROP 1 packets, 60 bytes)
 pkts bytes target   prot opt in  out source    destination
 679K  182M ACCEPT   all  --  *   lo  0.0.0.0/0 0.0.0.0/0
    0     0 DROP    !icmp --  *   *   0.0.0.0/0 0.0.0.0/0     state INVALID
    0     0 LOG      all  --  *   *   0.0.0.0/0 0.0.0.0/0     LOG flags 0
    0     0 REJECT   all  --  *   *   0.0.0.0/0 0.0.0.0/0

Chain eth0_in (1 references)
 pkts bytes target   prot opt in  out source    destination
 785K   93M ACCEPT   all  --  *   *   0.0.0.0/0 0.0.0.0/0
```

Listing 4: The filter table. Output of the iptables –Lvn command.

The syntax of the iptables command output is not specified and there is (therefore) no useful parser for that format. This makes an analysis of the structure of this string output necessary.
The syntax of the iptables command output is as follows: (The full result of this analysis is printed in the Appendix.)

The string output is organised as a list of blocks beginning with the string "chain". Each chain block consists of a head line, a description line and zero, one ore more rule lines. The head line contains the name of the chain, and information written in parentheses. Inside theses brackets there can be the default policy or the number of references to this chain.

The rest, i.e. the description line and the rule lines are organised as a table, see Listing 5. The description line words are the labels for the columns, the line contains always these nine words:
`pkts bytes target prot opt in out source destination.`
The table can contain one more column, an unlabeled options column.
The rule lines contain values for each column. The minimum of words is eight: the target column can be empty, the others always contains a word. In the additional options column, there can be more than one word.
An options cell begins with an option key followed by one or more option value words. Option keys relevant for ConfigImporter are "DSCP", "ECN", "MAC", "icmp", "tcp", "udp", "reject-with" and "LOG".



Listing 5: The chain blocks are organised as a table.

The first two columns pkts and bytes are running statistics and thus not relevant to the project. The third column contains the rule's target. Prot, opt, in, out, source and destination describe the match list, they specify the set of packets the rule has to match. The options column can contain additional settings to the target or to parts of the match list

As the syntax of the iptables command output is not specified, the above definition is the result of an intensive research on the iptables command. This was done with the help of the man page of iptables which describes in detail what can be set in the filter chains: every possible command changing the configuration of the filter chains was executed. The settings were then shown with the iptables listing command, and its characteristics were analysed. This was done for the iptables version 1.2.9. It is thinkable, that for other versions the output may not be the same, and ConfigImporter therefore will not work correctly.

3.2.5 Get the information for a <packet-filters/> element

To import the iptables settings into the VeriNeC environment, a <packet-filters/> element has to be added to the <services/> element.
The <packet-filters/> consists of a <packet-filter-chain/> element for each chain block. A chain block is only imported if it contains at least one rule line. The global-in, global-out and forward attributes are set to the corresponding chain[1] reference if such a chain exists.

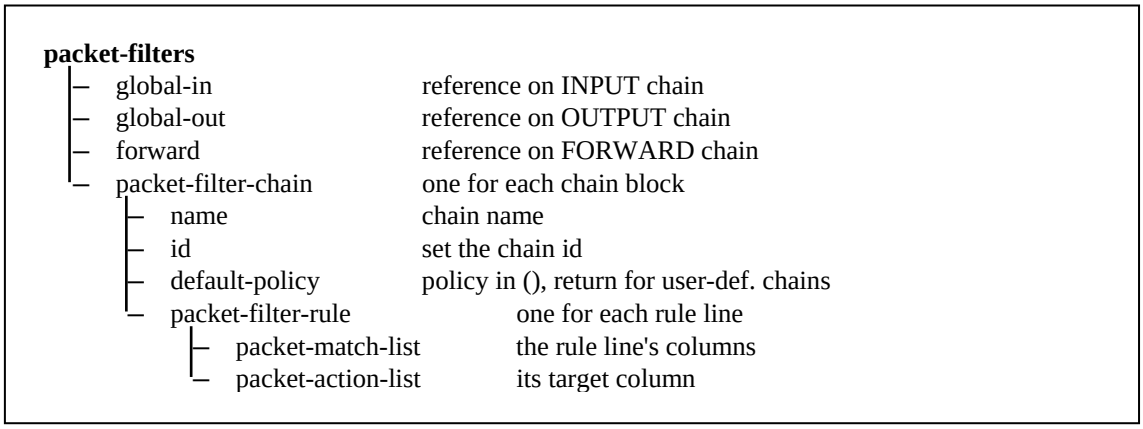


Figure 5: Information to generate a <packet-filters/> element

The <packet-filter-chain/> element's name attribute is set to the chain name; its id attribute is a unique string for referencing the chain.
The <default-policy/> element is set to an accept action if the default policy of the chain (found in parentheses) is ACCEPT and to a drop action if it is DROP. User-defined chains in iptables can't have a default policy, the <default-policy/> element for such chains is always set to a return action.
Further the <packet-filter-chain/> element has a <packet-filter-rule/> element for each rule line. A <packet-filter-rule/> element has always a <packet-match-list/> and a <packet-action-list/> element.

[1] global-out corresponds to the built-in INPUT chain, global-out to OUTPUT and forward to FORWARD

A <packet-match-list/> element consists of further elements depending on the values in the match list relevant columns (prot, opt, in, out, source, destination and options). The following list shows when and how these elements are set:

- match-in-interface, match-out-interface:
    - If the in respectively the out cell is not "*", that value is imported.
    - The interface given in that cell must exist: a reference on an element in hardware must be possible otherwise it can't be imported.
    - Iptables supports also the matching of interface types (e.g. match all Ethernet interfaces) this is not possible for VeriNeC
- match-mac:
    - If a match mac extension option is set, that value is imported.
- match-ipv4:
    - The source und destination cell values are imported (if they are not set to "0.0.0.0/0").
    - The opt column is analysed, the attribute fragment is set to "all", "first" or "subseq" when the value is "--", "!f" respectively "-f".
    - The match extensions dscp and ecn are searched for ip relevant settings.
- match-tcp:
    - If prot is "tcp" such an element is created.
    - Additional settings in the element are made if there are tcp of ecn extensions set.
- match-udp:
    - If prot is "udp" such an element is created.
    - Additional settings in the element are made if there are udp extensions set.
- match-icmp:
    - If prot is "icmp" such an element is created.
    - Additional settings in the element are made if there are icmp extensions set.

Unknown protocol names, other than "tcp", "udp" and "icmp", can not be imported.
Only a few values of the options column are paid regard to (described above). The man page of iptables describes other so called match extensions as the length, the owner, the state and the arrival time of a packet. These settings are not imported.

The <packet-action–list/> element is set to the value in the target column. Accepted values are ACCEPT, DROP, LOG, REJECT, RETURN, and names of user-defined chains. Other values are not imported, and a noop action is set. A noop action is also set if no target is printed in the target column.

The <packet-filters/> element to the filter table introduced in Listing 4 is printed in the Appendix; a reduced version of the INPUT chain is given in Listing 6.

```
<vn:packet-filter-chain name="INPUT" id="chain1">
        <vn:default-policy>
                <vn:drop-action />
        </vn:default-policy>
        <vn:packet-filter-rule>
                <vn:packet-match-list />
                <vn:packet-action-list />
        </vn:packet-filter-rule>
        <vn:packet-filter-rule />
        <vn:packet-filter-rule />
        <vn:packet-filter-rule />
        <vn:packet-filter-rule />
</vn:packet-filter-chain>
```

Listing 6: An imported INPUT chain.

While implementing ConfigImporter and the research on iptables output the XML schema node.xsd had to be changed slightly. As the abstraction for packet filtering is derived from ipchains (the antecessor of iptables) the functionalities of iptables had to be enabled.

- A forward attribute can be set in a <packet-filters> element to allow incoming packets to directly enter the FORWARD chain.
- It must be possible to set a return action as default policy.
- The negation of items of the match list must be feasible.

Iptables is special in the way interfaces can be matched: each rule can define an incoming and / or outgoing interface. In other implementations of packet filtering services a chain is linked to one interface and only packets coming from (or going to) that interface are filtered with that chain. Instead of the <match-interface/> element now a <match-in-interface/> and <match-out-interface/> can be set to each rule.

3.3 The hostname attribute

Another thing to set in the resulting XML file is the hostname. The hostname is an attribute directly set to the <node/> element.
In Linux, the hostname can be set and displayed with the hostname command. The output of this command is used to import the hostname when importing the settings of the local machine.
If the configuration of a remote computer is imported, the hostname is already given by the user to connect to the system.
In the case of importing configurations stored in files, the hostname is set to "test" or left at the name given by the user.

3.4 Configuration of Domain Name System

A network needs a tool to resolve human readable hostnames to numerical IP addresses. This is done with the Domain Name Service (DNS). Basically DNS is a list mapping hostnames to IP addresses. As there are far too many hosts to be centrally administrated, the list is organised hierarchically and distributed.
The hierarchy is given by the different parts of the address separated by a dot. Such a part is called a "zone". Each zone needs to specify its namespace, i.e. to control the naming of its sub-domains. So each network holds a part of the DNS list.

The default software used to set up DNS for a network under Red Hat is BIND (Berkeley Internet Name Domain). The name resolution service is done by the named daemon. Named is configured with the named.conf file in the /etc directory. Zone, statistic and cache files used by BIND are in the /var/named directory. [5]

3.4.1 The named.conf file

The configuration file for BIND is the named.conf file. This file specifies the role of the host for each zone. The settings are made in a list of statements. Each statement starts with a key word, the statement's type. It is followed by a statement name and optionally by a statement class. The statement's options are listed in braces.
There are a number of different statement types:

- With the `include` statement, additional files can be included into named.conf. This is used to break up large files or to hide certain data.
- The `options` statement assigns values to global options. Options can be the directory path and other things. If an option is not listed here, its default value is used. The options can be locally overridden by zones.
- The `acl` statement is an access control list which matches addresses to a name. By using the name of the statement this list can be called as an address match list.
- The basic statements in the named.conf file are the `zone` statements. Each zone statement defines one zone the host is authoritative for. In these statements, the location of its zone file and the local value of certain options are set. Listing 7 shows how a zone statement is structured and gives a typical example.

```
Structure

zone <zone-name> <zone-class> {
     <zone-options>;
     [<zone-options>; ...]
};

Example

zone "localhost" IN {
       type master;
       file "localhost.zone";
       allow-update { none; };
};
```

Listing 7: The structure of a zone file

Other settings to named can be made with further statements as the `controls`, the `logging`, the `server` and the `key` statement. [4]

3.4.2 The zone files

To each zone defined in named.conf a zone file is assigned. A zone file contains two types of information: parser commands and resource records. The parser commands are tasks for named to execute as to assign a local value to the $ORIGIN directive which can then be used in the zone file.

The resource records are forming the database for DNS. There are different types of resource records. The first resource record to appear in the file is the SOA record (exactly one per zone). SOA stands for "Start Of Authority". This record holds the name of the zone, the email address of the administrator and timeout values. See Listing 12 for the form of a SOA record (@ stands for the $ORIGIN directive).

```
@      IN      SOA    <primary-name-server> <hostmaster-email> (
                      <serial-number>
                      <time-to-refresh>
                      <time-to-retry>
                      <time-to-expire>
                      <minimum-TTL> )
```

Listing 8: The structure of a SOA record

There are further resource record types which are structured similarly. The following listing shows this form.

```
<match>      IN      <record name>      <target>
```

Listing 9: The structure of resource records

Address (A) records assign an IP-Address to a hostname. Name Server (NS) records define the servers that are responsible for the zone. To map an alias name to a real host names CNAME records can be used. TXT records add text to a host. MX records define the destination of emails to a certain namespace. MX records have an additional priority value to show which mail server to prefer to another. There are further resource record types which are less frequently used and can't be imported to the VeriNeC project.

Each zone has also its reverse name resolution zone file to translate IP addresses in hostnames. The records found in these files are created automatically by VeriNeC and therefore need not to be imported.

3.4.3 Get the information for a <dns/> element

To import the DNS settings into the VeriNeC environment for each zone defined in the named.conf file a <zone/> element has to be added to the <dns/> element (not for the reverse name resolution zone files).
Figure 6 maps the attributes and sub-elements of a <zone/> element to the parameters found in the named.conf and the zone files.

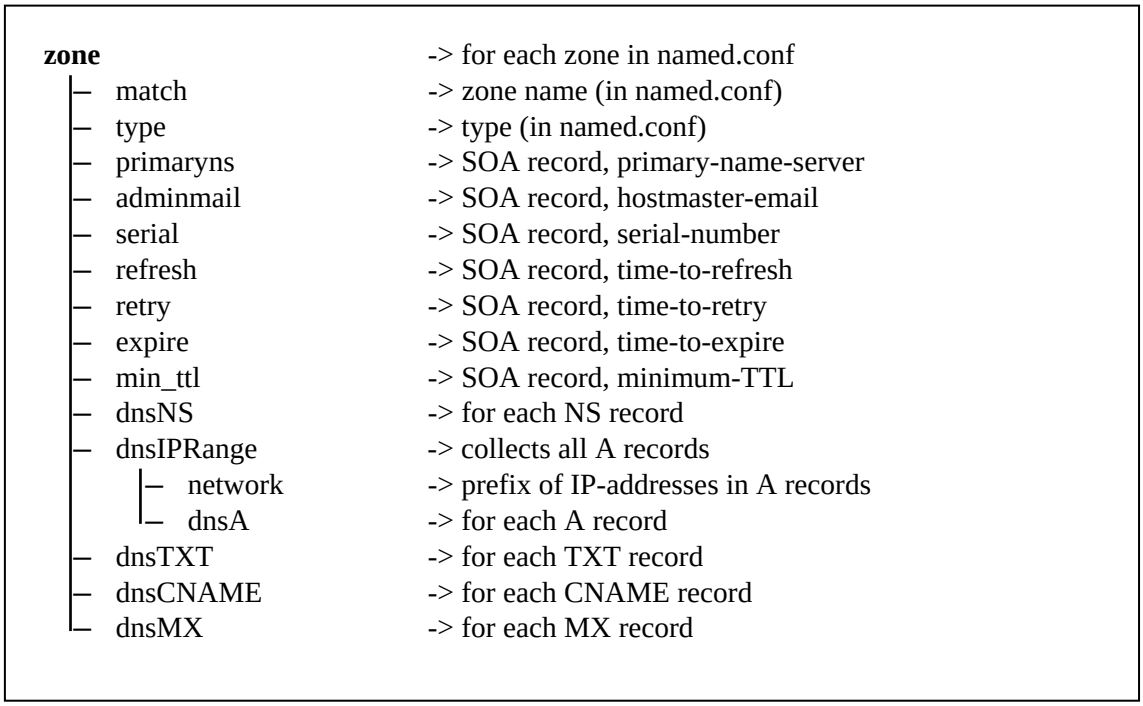


Figure 6: Mapping of <zone/> element attributes to file parameters

The match attribute has to be set to the name of the zone, the type attribute to the type value of the zone statement in the named.conf file.
The values for the attributes primaryns, adminmail, serial, refresh, retry, expire and min_ttl can be found in the SOA record of the zone's file.

The elements <dnsNS/>, <dnsTXT/>, <dnsCNAME/> and <dnsMX/> are created if a corresponding resource record is given in the zone file, its attributes are set to the <match> and <target> parameter of that resource respectively. A <dnsIPRange/> element is created for each group of A records with the same prefix. The network attribute is set to that prefix value and a <dnsA/> element is created for every A record of the group.

This is only a general survey of how to import DNS settings into the VeriNeC environment. A detailed analysis of the structure of the files needs to be done before implementing a parser and an importer for that service.

4 Java Implementation of ConfigImporter

ConfigImporter is implemented in Java (J2SE v1.4.2 SDK[1]) using the Eclipse Platform 3.0. In VeriNeC's GUI the user starts ConfigImporter: a new instance of the *ImporterDialog* class is created. This opens a window to communicate with the user and starts the two importation parts of Ethernet configurations files and iptables settings.

4.1 Importer, ImporterDialog, ImporterEnvironment

The interface between VeriNeC and its new component ConfigImporter is the class *ImporterDialog*. A new instance of that class is created when a user chooses to import the configuration of a computer in the VeriNeC's GUI.
ImporterDialog implements JFrame[2], a window opens when it is instanced. (See chapter 5, ConfigImporter User Guide, for the appearance of the window.)
The user can choose different options in the window; these are stored in a new *ImporterEnvironment* object, a member variable of the *ImporterDialog* object.

Only one object of *ImporterEnvironment* is instantiated during one run of ConfigImporter. That object holds all needed information. That is the settings made by the user about what to import and information about how to connect to the system, information of the calling object about how to store the created XML data, and other data generated during the importation of the systems network configuration.
Further *ImporterEnvironment* creates a new *XMLConfiguration* object which holds all information about the VeriNeC <node/> XML element to be created. *XMLConfiguration* provides methods to add sub-elements and attributes to the resulting <node/> element. These methods are used on different points during the importation.
In the end of the importation a method of *XMLConfiguration* can be called to put all the stored items together and create the resulting <node/> element.
These methods are realised using the Java API JDOM (org.jdom).

[1] Java 2 Platform, Standard Edition, v 1.4.2, Software Development Kit
[2] javax.swing.JFrame

When the user has made his decisions and has pressed the "Import" button, the data is stored in the *ImporterEnvironment* and *Importer* is started.
Importer creates an *ImportConfigFiles* and an *ImportIptables* object to import the configuration data of network interfaces and iptables depending on the user settings.
These two classes save their analysis in the *ImporterEnvironment* object. In the end *Importer* calls the method of *XMLConfiguration* described above to generate the resulting XML file with the data gathered during the importation run. The file is then saved for debugging reason. Then the VeriNeC item ConfigImporter is started for is adapted, i.e. its <ethernet/> and <packet-filters/> elements are exchanged with the newly created elements.

4.2 ImportConfigFiles

A new instance of *ImportConfigFiles* is made by the *Importer* object. *ImportConfigFiles* analyses the user settings given in the *ImporterEnvironment* object, i.e. it looks up if it has to import Ethernet interface configurations or not, and whether to import the local settings, to remotely connect to another system or to import configurations given in files on the local file system.
The analysis results in creating a temporary copy of the directory containing the network interface configuration files (normaly the */etc/sysconfig/network-scripts* directory).

ImportConfigFiles searches in this directory for all files starting with "ifcfg-eth" (and "ifcfg-lo"). For each file found, a new *ConfigFile* object is produced. *ConfigFile* provides a method to create a VeriNeC <nw/> XML element with the data of the configuration file passed (see the next section, ConfigFile).
ImportConfigFiles creates a VeriNeC <ethernet/> XML element for all main (non-alias) files and adds the <nw/> element given by the corresponding *ConfigFile* object.
If an interface has one or more alias files, *ImportConfigFiles* adds that alias files' <nw/> element to the interface's <ethernet/> element.
The <ethernet/> elements are added to the *ImporterEnvironment* object by using the intended method of *XMLConfiguration*.

4.2.1 ConfigFile

ConfigFile represents an interface configuration file. It gets the name of the file to analyse, fetches the string content of the file and creates an *IniEditor* object of the file.

As the configuration files are of a simple form, there are existing parsers for them. One is IniEditor[1]; this parser was chosen because it also works for one-sectioned files.
IniEditor is a library to analyse ini-style files. The application provides methods to get the value of a certain option or to return all options defined in the file. IniEditor is a product of ubique.ch and is under a Berkeley Software Distribution license[2].

[1] IniEditor is Copyright © 2003-2005, Nik Haldimann, http://ubique.ch/code/inieditor/
[2] The license text is printed in the Appendix

With the help of these methods, ConfigImporter parses the file and generates a VeriNeC <nw/> XML element: For every possible attribute and sub-element the XML file can define *ConfigFile* tests whether the corresponding option (see chapter 3.1.2, Get the information for a <ethernet/> element) is set in the configuration file. If yes, it uses a JDOM method to create an attribute, set its value and add it to the <nw/> element.

ConfigImporter provides methods to access the parameters set in the configurations file and the created <nw/> element.

4.3 ImportIptables

As described in chapter 3.2, the source of the importation of packet filters configuration is the output of the iptables command `iptables -Lvn`.
The implementation is realised with the class *ImportIptables*. After analysing the *ImporterEnvironment* object, this class has to set up a connection with the concerning machine and execute the command on that system, or get the data form a file. The output is stored in a String variable of the class and given to the *Iptable* class to create a new object.
As there is no existing parser for the format of that information, an implementation of a parser is needed. This is realised with the *Parser* and *Lexer* classes of ConfigImporter (see next section, Parser for details).
The parser generates a parse tree; this is then traversed by *Iptable*. That class contains a class for every possible sub-element of the VeriNeC <packet-filter-chain/> XML element.
The constructors of these classes get a parameter, i.e. an instance of the corresponding *Parser* class and decide whether to make an element. Each class has, as interface to its calling object, a public variable holding the created element and a public boolean variable saying whether an element has been created or not.
ImportIptables gets a list of the <packet-filter-chain/> elements created by *Iptable* and adds them to a new VeriNeC <packet-filters/> XML element. Some additional attributes are set, and the element is added to the XML file using the method of *XMLConfiguration*. Further, the <prefix-list> elements also created during the importation of iptables are added to the XML file.

4.3.1 Parser

The output of the command needs to be semantically tested, and a parse tree, that is a hierarchical structure of the relevant items, has to be created. That is usually done by a so called parser. Before the parser can do its job, a lexer has to divide the string into lexical units.

The *Lexer* class makes this lexical analysis of an input. It divides the string into tokens using *StringTokenizer*[1]. As separators, the following characters are defined: blank, return, new line, tabulator, the brackets "(" and ")", the colon "," and the exclamation mark "!". For each token, a new instance of the class *Node* is generated, which can store the token's word, its type, the line number and the position of its occurrence in the string.

The list of tokens is then handed over to the *Parser* class. *Parser* analyses the tokens and decides whether they can occur at that place. For example, the first token in the output has to be the String "chain" (as blanks and new lines in the beginning of the string are eliminated by the *Lexer*).
Besides testing, the parser has to generate the parse tree. The *Parse* tree's nodes are extended *Node* objects, which can store the node's type and the type-specific items. A node of the type "Rule" for example can store eight further Nodes object, one for each column.
The *Parser* does not test if the string is an accurate iptables configuration, it can not test, if the items are allowed to occur in iptables. It is supposed to be satisfactory. But it tests whether the output has the expected structure, to guarantee a correct importation of the settings.

[1] java.util.StringTokenizer, see its JavaDoc for details

5 ConfigImporter User Guide

Once VeriNeC is started and its GUI appears, the user has to create a new node or to load a saved network. With a right click on a node in the GUI the node's content menu appears (see Figure 7). To start ConfigImporter, the user clicks on the "Import" menu item.

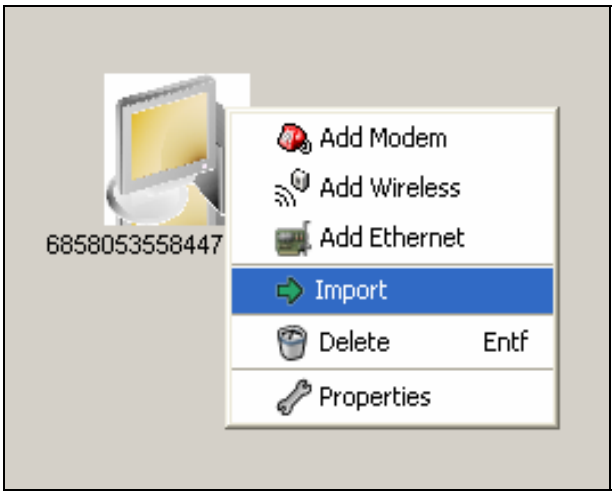


Figure 7: The node's context menu

The configurations imported with ConfigImporter are in the end added to that node, changing its XML <node/> definition.
After clicking on "Import" a new Import Dialog window appears (see Figure 8). This window has three parts: a choice field, a settings field and a buttons filed.

The choice field

In the choice field the user decides which computer's network configurations to import. He has to select one of the three options:

1. Import the configurations of this computer
2. Import the configurations of a remote computer
3. Import the configurations stored in files

He can import the settings of the machine he is working on by choosing the first item. To import the settings of another machine on the network, the second option has to be selected. By marking the third item, network configurations stored in files on the local file system can be used as base for the importation.

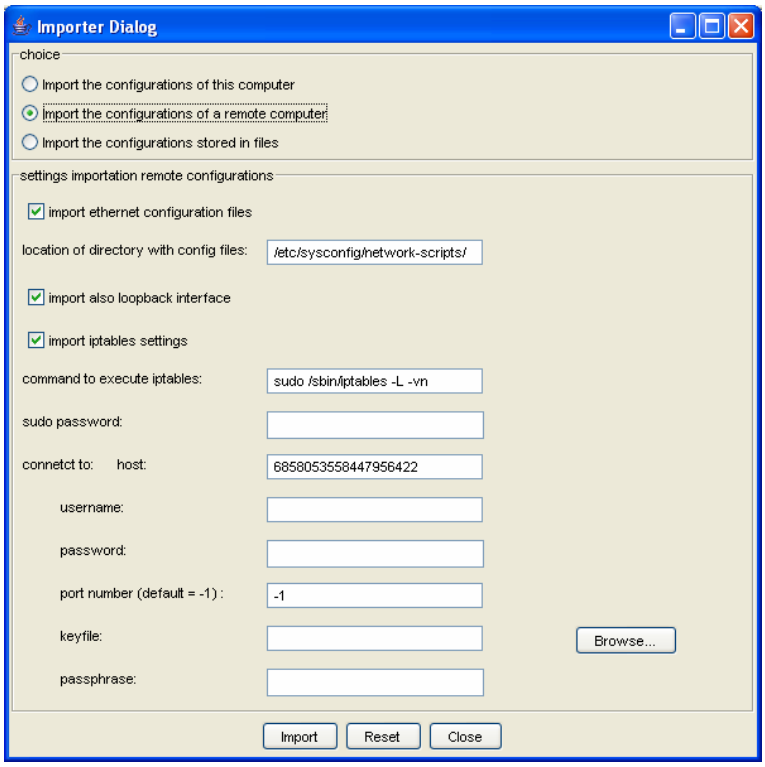


Figure 8: The Importer Dialog window

The settings field

To import the network configurations, some additional information needs to be given by the user. Depending on the selected option in the choice field, this is different information. The appearance of the window changes slightly by selecting another item in the choice field: to each item a different settings field is assigned.
In each settings field, the user has to mark a check box "import ethernet configuration files" or a check box "import iptables configuration settings" if he wants to import the one or the other or both.

When the user chooses the local system, the following items have to be set in addition:

- The location of the directory containing the configuration files. This is set to "/etc/sysconfig/network-scripts/" by default but can be changed if it exceptionally should be different. To select the desired directory the "Browse…" button can be used or the path can be written directly in the text field. If the Ethernet check box is not selected, this field is ignored.
- Whether or not to import the configurations of the loopback interface. This works only if also Ethernet configurations are imported.
- The command to execute iptables. This is set to the command line "sudo /sbin/iptables -L –vn". The "sudo" string can be deleted if the user is root. The command string can also be changed if iptables is executed on a different location. The option "-L –vn" should not be changed to guarantee a correct parsing of the iptables output. This filed has only influence if the iptables check box is selected.
- The sudo password. This must be given if the "sudo" string in the command is kept.

If the user wishes to import the settings of a remote machine, the same settings described above have to be set. Additionally, the information about the host to connect to needs to be given:

- The host name, the user name and the user's password need to be written in the text fields. The host name is already set to the name of the node selected in the VeriNeC's GUI but can be changed if it is not the desired host.
- Additionally, a port number, a key file and a pass phrase can be set for ssh.

If the configuration settings should be taken from files stored on the local file system, the location of the Ethernet configuration files and the location of the file containing the iptable output must be given. These paths can be selected by using the "Browse…" buttons or by directly writing them in the text fields.

The buttons field

Once all the decisions are made in the settings field, the "Import" button can be pushed to start ConfigImporter. The configurations are then imported and stored to the selected node of the GUI.
The "Reset" button can be useful to restore the default values in the different text fields.
By pressing the "Close" button the window is closed and the user is returned to the VeriNeC's GUI.

Warnings and Errors

After pressing the "Import" button, ConfigImporter is doing its job. Once the importation of the configurations is done, warnings are printed out to inform the user about the items that could not be imported. The warnings are grouped to Ethernet and iptables warnings. An example of such a dialog is given in Figure 9.

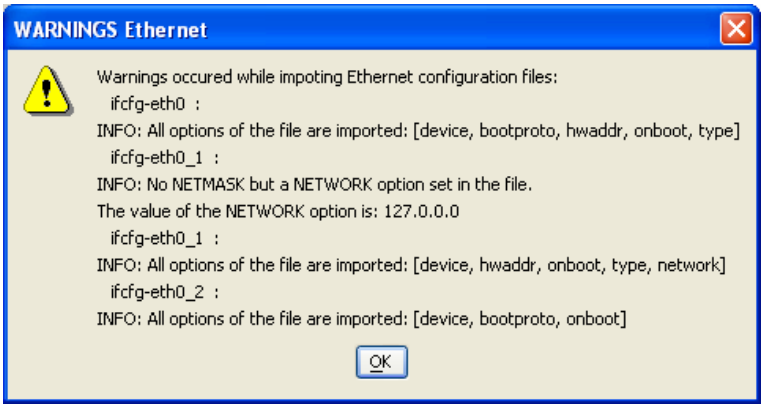


Figure 9: The WARNINGS Dialog

If during the importation an error occurs the user is informed by an ERROR dialog like the one in Figure 10.

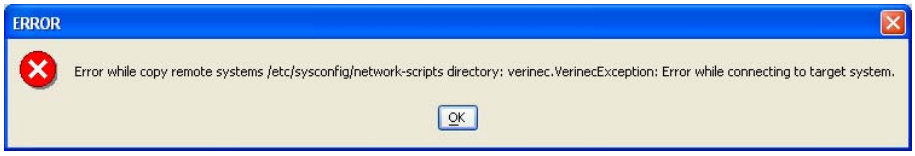


Figure 10: The ERROR Dialog

6 Conclusion

This thesis shows that it is possible to integrate importer functionalities to the VeriNeC project. ConfigImpoter can now be used to generate abstract descriptions from network configurations. System administrators could take this tool to start using VeriNeC without redefining their existing network definitions from scratch.
But ConfigImporter has its limitations: only a few possible configuration settings are imported for only a few services.

1. Limits of schema
As the services described in the VeriNeC's network definition are an abstraction of the functionality of service type, the different implementation's special features may be lost. So ConfigImporter only imports a small set of parameters in an Ethernet configuration file and only very few options of the iptables command are supported by the VeriNeC's network definition. While the omitted items may are not the important things, something gets still lost during the importation.

2. Implement more things
To give VeriNeC the functionality of exchanging one machine of the network by another machine with a different operating system (and other uses described in the beginning of this paper), a lot more things need to be implemented:

- The importation of the DNS service needs to be implemented. The format of the configuration files holding the DNS information requires the implementation of a parser.
- Further services as serial and WLAN interfaces and routing need a realisation.
- ConfigImporter is only implemented for the Linux distribution Fedora Red Hat. This is a good choice as it belongs to a widely used type of Linux distributions. But there are other distributions that need an implementation of ConfigImporter. Furthermore the importation of network configuration settings in other operating systems should be realised.

A good choice was to take the information about iptables settings form the command output. That part of ConfigImporter works not only for a single distribution but could work for all UNIX-like systems (using iptables).

ConfigImporter is a small tool for VeriNeC and only a first step in the implementation of a huge, versatile Importer module.

Bibliography

[1] *The VeriNeC project homepage*: http://diuf.unifr.ch/tns/projects/verinec/

[2] *The Role of Simulation in a Network Configuration Engineering Approach.* Dominik Jungo, David Buchmann and Ulrich Ultes-Nitsche. International Conference on Information & Communication Technology (ICICT 2004), Cairo, Egypt.

[3] *VeriNeC Translation Module*. Working Paper. David Buchmann. University Fribourg

[4] *LINUX Administration Handbook.* Evi Nemeth, Garth Snyder, Trent R. Hein. Prentice Hall PTR, 2002.

[5] *Red Hat Linux 9, Red Hat Linux Reference Guide*. Red Hat Inc., 2003.

[6] *IniEditor homepage*: http://www.ubique.ch/code/inieditor/

[7] *Networking HOWTO*: http://www.netfilter.org/documentation /HOWTO//networking-concepts-HOWTO.html

[8] *Packet Filtering HOWTO*: http://www.netfilter.org/ documentation /HOWTO/packet-filtering-HOWTO.html

[9] *Netfilter Hacking HOWTO*: http://www.netfilter.org/documentation /HOWTO//netfilter-hacking-HOWTO.html

[10] *Iptables – Die Firewall des Kernels 2.4*, Wolfgang Kinkeldei, http://www.pl-forum.de/t_netzwerk/iptables.html

[11] *NAT HOWTO*: http://www.netfilter.org/documentation /HOWTO//NAT-HOWTO.html

[12] *Man page of IPTABLES*: http://www.manpage.org/cgi-bin/man/man2html?query=iptables

[13] *Network Sniffer Ein Modul für VeriNeC.* Patrick Aebischer. Universität Freiburg, 2005

[14] *Computer Networks* Andrew S. Tanenbaum. Prentice Hall, 2003.

Appendix A
Acronyms

API	Application Programming Interface
BIND	Berkeley Internet Name Domain
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DSCP	Differentiated Services Code Point
GUI	Graphical User Interface
ICMP	Internet Control Message Protocol
IP	Internet Protocol
LAN	Local Area Network
MAC	Media Access Control
NIC	Network Interface Card
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VeriNeC	Verified Network Configuration
WLAN	Wireless LAN
XML	eXtensible Markup Language

Appendix B
Grammer of iptables output

Table = Chain*

Chain = name: ChainName;
info: Parenthesis;
rules: Rules

ChainName = BuiltinChain | UserdefChain
BuiltinChain = "INPUT"| "OUTPUT"| "FORWARD"| "PREROUTING"|
"POSTROUTING"
UserdefChain = chainname: **S**

Parenthesis = Policy | References
Policy = targetname: BuiltinTarget
BuiltinTarget = "ACCEPT" | "DROP"
References = quantity: **Z**

Rules = Rule*
Rule = tar: Target;
prot: Protocol;
opt: Fragment;
in: Interface;
out: Interface;
source: Address;
destination: Address;
options: Options;

Target = BuiltinTarget | SpecialBuiltinTarget | ExtensionTarget | UserdefChain
SpecialBuiltinTarget = "QUEUE"| "RETURN"
ExtensionTarget = "LOG" | "REJECT"|
// -> and others, not supported by VeriNeC

Prot = neg: **B**;
name: ProtName
ProtName = CommonProtocol | ProtocolName | ProtocolNumber
CommonProtocol = "tcp" | "udp" | "icmp" | "all"
ProtocolName = name: **S**
ProtocolNumber = number: **Z** // not supported by VeriNeC

Fragment = "--" | "-f" | "!f"

Interface = "*" | DefinedInterface
DefinedInterface = neg: B;
interface: SpecificInterface

SpecificInterface = InterfaceName | InterfaceType
InterfaceName = name: **S**
InterfaceType = type: **S** "+" // z.b. eth+

Address = neg: **B**;
address: IPAddr;
mask: **Z**

Options = Option*
Option = key: OptionKey;
value: OptionValue

OptionKey = "DSCP" | "ECN" | "MAC" | "icmp" | "tcp" | "udp" | "reject-with" | "LOG"
// -> and others, not supported by VeriNeC

OptionValue = DscpOption | EcnOption | IcmpOption | MacOption |
TcpOption, | UdpOption | RejectOption | LogOption
//depends on OptionKey
// -> and others, not supported by VeriNeC

DscpOption = MatchDscp | TargetDscp
MatchDscp = neg: **B**;
filed: DscpField
DscpField = HexString
TargetDscp = DscpField // -> not supported by VeriNeC

EcnOption = MatchEcn | TargetEcn
MatchEcn = CWR | ECE | ECT
CWR = bit: **B**
ECE = bit: **B**
ECT = bits: **Z**
TargetEcn = **B** // -> not supported by VeriNeC

IcmpOption = neg: **B**;
type: **Z**;
code: **Z**

MacOption = neg: **B**;
addr: MacAddr

TcpOption = Source | Dest | TcpFlags | TcpOptions

Source = neg: **B**;
ports: Ports;

Dest = neg: **B**;
ports: Ports;
TcpFlags = neg: **B**;
flags: Flags;

TcpOptions = neg: **B**;
opt: **Z**;

Ports = Port | Range

Port = port: **Z**
Range = lo:**Z**;
hi:**Z**

Flags = mask: HexString
comp: HexString

UdpOption = Source | Dest

RejectOption = Type
Type = "icmp-net-unreachable" | "icmp-host-unreachable" |
"icmp-proto-unreachable" | "icmp-port-unreachable" |
"icmp-net-prohibited" | "icmp-host-prohibited" |
"tcp-reset" | "icmp-admin-prohibited"

LogOption = Level
//other log-options not supported by VeriNeC

Level = level: **Z**

Appendix C
Files of ConfigImporter Demo

1. Ethernet configuration files

ifcfg-eth0

```
# ifcfg-eth0
DEVICE=eth0
BOOTPROTO=dhcp
HWADDR=00:06:5B:A9:EF:60
ONBOOT=yes
TYPE=Ethernet
```

ifcfg-eth0:0

```
# alias file ifcfg-eth0:0
DEVICE=eth0:0
HWADDR=00:06:5B:A9:00:60
ONBOOT=yes
TYPE=Ethernet
NETWORK=127.0.0.0
```

ifcfg-eth1

```
# ifcfg-eth1
DEVICE=eth1
BOOTPROTO=dhcp
ONBOOT=yes
```

ifcfg-lo

```
DEVICE=lo
IPADDR=127.0.0.1
NETMASK=255.0.0.0
NETWORK=127.0.0.0
BROADCAST=127.255.255.255
ONBOOT=yes
NAME=loopback
```

2. iptables output

```
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target   prot opt in   out   source      destination
 679K  182M ACCEPT   all  --  lo   *     0.0.0.0/0   0.0.0.0/0
    0     0 DROP    !icmp --  *    *     0.0.0.0/0   0.0.0.0/0     state INVALID
 785K   93M eth0_in  all  --  eth0 *     0.0.0.0/0   0.0.0.0/0
    0     0 LOG      all  --  *    *     0.0.0.0/0   0.0.0.0/0     LOG flags 0 level 6
    0     0 REJECT   all  --  *    *     0.0.0.0/0   0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target   prot opt in   out   source      destination
    0     0 DROP    !icmp --  *    *     0.0.0.0/0   0.0.0.0/0     state INVALID
    0     0 LOG      all  --  *    *     0.0.0.0/0   0.0.0.0/0     LOG flags 0 level 6
    0     0 REJECT   all  --  *    *     0.0.0.0/0   0.0.0.0/0

Chain OUTPUT (policy DROP 1 packets, 60 bytes)
 pkts bytes target   prot opt in   out   source      destination
 679K  182M ACCEPT   all  --  *    lo    0.0.0.0/0   0.0.0.0/0
    0     0 DROP    !icmp --  *    *     0.0.0.0/0   0.0.0.0/0     state INVALID
    0     0 LOG      all  --  *    *     0.0.0.0/0   0.0.0.0/0     LOG flags 0 level 6
    0     0 REJECT   all  --  *    *     0.0.0.0/0   0.0.0.0/0


Chain eth0_in (1 references)
 pkts bytes target   prot opt in   out   source      destination
 785K   93M ACCEPT   all  --  *    *     0.0.0.0/0    0.0.0.0/0
```

3. The resulting XML File node.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<vn:node xmlns:vn="http://diuf.unifr.ch/tns/projects/verinec/node"
                  hostname="test">
<!--Configuration written by ConfigImporter on Jul 26, 2005 11:25:40 AM-->
  <vn:hardware>
     <vn:ethernet name="Ehernet card eth0" hwaddress="00:06:5B:A9:EF:60">
        <vn:hint system="pc" slot="0" />
        <vn:ethernet-binding id="bind--1096002608" name="eth0">
           <vn:nw id="nw-996848155" onboot="yes" type="ip">
              <vn:dyn type="dhcp" />
           </vn:nw>
           <vn:nw id="nw-1298759089" onboot="yes"
                  hwaddress="00:06:5B:A9:00:60" type="ip" />
        </vn:ethernet-binding>
     </vn:ethernet>
     <vn:ethernet name="Ehernet card eth1">
        <vn:hint system="pc" slot="1" />
        <vn:ethernet-binding id="bind--644411961" name="eth1">
           <vn:nw id="nw--292886184" onboot="yes" type="ip">
              <vn:dyn type="dhcp" />
           </vn:nw>
        </vn:ethernet-binding>
     </vn:ethernet>
     <vn:ethernet name="loopback interface">
        <vn:ethernet-binding id="lobind--1579874649" name="lo">
           <vn:nw id="nw--400720729" address="127.0.0.1"
                 subnet="255.0.0.0" onboot="yes" type="ip" />
        </vn:ethernet-binding>
     </vn:ethernet>
  </vn:hardware>
  <vn:services>
     <vn:packet-filters global-in="chain1" global-out="chain3"
forward="chain2">
        <vn:packet-filter-chain name="INPUT" id="chain1">
           <vn:default-policy>
              <vn:drop-action />
           </vn:default-policy>
           <vn:packet-filter-rule>
              <vn:packet-match-list>
                 <vn:match-in-interface if="lobind--1579874649" />
              </vn:packet-match-list>
              <vn:packet-action-list>
                 <vn:accept-action />
              </vn:packet-action-list>
           </vn:packet-filter-rule>
           <vn:packet-filter-rule>
              <vn:packet-match-list>
                 <vn:match-icmp negate-icmp="yes" />
              </vn:packet-match-list>
              <vn:packet-action-list>
                 <vn:drop-action />
              </vn:packet-action-list>
           </vn:packet-filter-rule>
           <vn:packet-filter-rule>
              <vn:packet-match-list>
                 <vn:match-in-interface if="bind--1096002608" />
              </vn:packet-match-list>
              <vn:packet-action-list>
                 <vn:gosub-action goto="chain4" />
              </vn:packet-action-list>
```

```
      </vn:packet-filter-rule>
      <vn:packet-filter-rule>
         <vn:packet-match-list />
         <vn:packet-action-list>
            <vn:log-action level="info" />
            <vn:noop-action />
         </vn:packet-action-list>
      </vn:packet-filter-rule>
      <vn:packet-filter-rule>
         <vn:packet-match-list />
         <vn:packet-action-list>
            <vn:reject-action />
         </vn:packet-action-list>
      </vn:packet-filter-rule>
   </vn:packet-filter-chain>
   <vn:packet-filter-chain name="FORWARD" id="chain2">
      <vn:default-policy>
         <vn:drop-action />
      </vn:default-policy>
      <vn:packet-filter-rule>
         <vn:packet-match-list>
            <vn:match-icmp negate-icmp="yes" />
         </vn:packet-match-list>
         <vn:packet-action-list>
            <vn:drop-action />
         </vn:packet-action-list>
      </vn:packet-filter-rule>
      <vn:packet-filter-rule>
         <vn:packet-match-list />
         <vn:packet-action-list>
            <vn:log-action level="info" />
            <vn:noop-action />
         </vn:packet-action-list>
      </vn:packet-filter-rule>
      <vn:packet-filter-rule>
         <vn:packet-match-list />
         <vn:packet-action-list>
            <vn:reject-action />
         </vn:packet-action-list>
      </vn:packet-filter-rule>
   </vn:packet-filter-chain>
   <vn:packet-filter-chain name="OUTPUT" id="chain3">
      <vn:default-policy>
         <vn:drop-action />
      </vn:default-policy>
      <vn:packet-filter-rule>
         <vn:packet-match-list>
            <vn:match-out-interface if="lobind--1579874649" />
         </vn:packet-match-list>
         <vn:packet-action-list>
            <vn:accept-action />
         </vn:packet-action-list>
      </vn:packet-filter-rule>
      <vn:packet-filter-rule>
         <vn:packet-match-list>
            <vn:match-icmp negate-icmp="yes" />
         </vn:packet-match-list>
         <vn:packet-action-list>
            <vn:drop-action />
         </vn:packet-action-list>
      </vn:packet-filter-rule>
      <vn:packet-filter-rule>
         <vn:packet-match-list />
```

```
            <vn:packet-action-list>
              <vn:log-action level="info" />
              <vn:noop-action />
            </vn:packet-action-list>
          </vn:packet-filter-rule>
          <vn:packet-filter-rule>
            <vn:packet-match-list />
            <vn:packet-action-list>
              <vn:reject-action />
            </vn:packet-action-list>
          </vn:packet-filter-rule>
        </vn:packet-filter-chain>
        <vn:packet-filter-chain name="eth0_in" id="chain4">
          <vn:default-policy>
            <vn:return-action />
          </vn:default-policy>
          <vn:packet-filter-rule>
            <vn:packet-match-list />
            <vn:packet-action-list>
              <vn:accept-action />
            </vn:packet-action-list>
          </vn:packet-filter-rule>
        </vn:packet-filter-chain>
      </vn:packet-filters>
  </vn:services>
</vn:node>
```

Appendix D
IniEditor License