# Firewall Builder User's Guide

**Firewall Builder User's Guide**
$Revision: 1.24 $ $Date: 2003/11/04 06:46:19 $ Edition
Copyright © 2003 by NetCitadel LLC

The information in this manual is subject to change without notice and should not be construed as a commitment by NetCitadel LLC. NetCitadel LLC assumes no responsibility or liability for any errors or inaccuracies that may appear in this manual.

# Table of Contents


**1. Introduction ..... 1**
1.1. Introducing Firewall Builder ..... 1
1.2. Overview of Firewall Builder Features ..... 1
**2. Packaging ..... 3**
**3. Firewall Builder GUI ..... 5**
3.1. The Main Window ..... 5
3.2. GUI Options ..... 5
**4. Object Tree ..... 9**
4.1. The Standard Tree Structure ..... 9
4.2. Creating Objects ..... 10
4.3. Navigating The Object Tree and Editing Objects ..... 11
**5. Working With Objects ..... 15**
5.1. Common Properties ..... 15
5.2. The Host Object ..... 15
5.2.1. Creating a Host Object ..... 15
5.2.2. Editing a Host Object ..... 17
5.3. The Host's Interface ..... 19
5.3.1. Interface Types ..... 20
5.4. The Address Object ..... 21
5.5. The Physical Address Object ..... 22
5.5.1. Using The Physical Address Object in the Policy ..... 23
5.6. The Network Object ..... 24
5.7. The Address Range Object ..... 25
5.8. A Group of Objects ..... 26
5.9. The Firewall Object ..... 27
5.9.1. Creating a Firewall Object ..... 27
5.9.2. Editing a Firewall Object ..... 29
5.10. Configuring Target Firewall Software ..... 32
5.10.1. Generic Parameters ..... 32
5.10.2. iptables ..... 34
5.10.3. ipfilter ..... 37
5.10.4. pf ..... 39
5.10.5. PIX ..... 41
5.11. Configuring Host OS ..... 47
5.11.1. FreeBSD ..... 47
5.11.2. Linux ..... 48
5.11.3. OpenBSD ..... 49
5.11.4. Solaris ..... 50
5.11.5. PIX ..... 51
5.12. The Firewall's Interface ..... 51
5.12.1. The Network Zone ..... 53
**6. Network Discovery: A Quick Way to Create Objects ..... 55**
6.1. Reading the /etc/hosts file ..... 56
6.2. DNS zone import ..... 58
6.3. Network Discovery ..... 61
**7. Working With Services ..... 67**
7.1. IP Service ..... 67
7.2. ICMP Service ..... 69
7.3. TCP Service ..... 70
7.4. UDP Service ..... 74
7.5. Custom Service ..... 75

**8. Firewall Policies** ..... **79**
8.1. Policies and Rules ..... 79
8.1.1. Actions ..... 80
8.1.2. Directions ..... 81
8.1.3. Global Firewall Policy ..... 81
8.1.4. Interface Policy ..... 82
8.2. Network Address Translation Rules ..... 83
8.2.1. Basic NAT Rules ..... 84
8.2.2. Source Address Translation ..... 85
8.2.3. Destination Address Translation ..... 88
8.3. Editing firewall Policy and NAT Rules ..... 89
8.3.1. Adding and removing rules ..... 90
8.3.2. Adding, removing and modifying objects in the policy and NAT rules ..... 90
8.3.3. Changing rule action ..... 91
8.3.4. Changing rule direction ..... 92
8.3.5. Changing rule options and logging ..... 93
8.4. Support for Rule Elements and Features on Various Firewalls ..... 93
8.5. Using Objects With Multiple Addresses in the Policy and NAT rules ..... 94
**9. Examples of Policy Rules** ..... **97**
9.1. Letting certain protocols through, while blocking everything else ..... 97
9.2. Interchangeable and non-interchangeable objects ..... 97
9.3. Using groups ..... 97
9.4. Firewall talking to itself ..... 98
9.5. Blocking unwanted types of packets ..... 98
9.6. Anti-spoofing rules ..... 100
9.7. Protecting local host ..... 101
9.8. Using Action 'Reject': blocking Ident protocol ..... 101
9.9. Using negation in policy rules ..... 102
**10. Examples of Network Address Translation Rules** ..... **105**
10.1. Providing Internet connection for workstations behind the firewall ..... 105
10.2. Server behind the firewall using address of the firewall for access ..... 106
10.3. Server behind the firewall using virtual address for access ..... 108
10.4. Server behind the firewall with port mapping ..... 109
10.5. Destination NAT Onto the Same Network ..... 110
10.6. "No NAT" rules ..... 111
10.7. Redirecting the traffic ..... 111

# Chapter 1. Introduction

## 1.1. Introducing Firewall Builder

Whether you are an experienced professional, switching from Checkpoint Firewall-1 to Cisco PIX or to an Open Source firewall, or a hobbyist trying to secure your home network, Firewall Builder can help you manage your firewall policy in a simpler and more efficient way.

With Firewall Builder, you can manage the security policy of your firewall efficiently and accurately, without the learning curve usually associated with command line interfaces of many free and commercial firewalls. You do not have to think of a policy in terms of obscure port numbers, firewall interfaces and making choices about what interface a certain rule should belong to and so on. Instead, you create a set of objects describing your firewall, servers, subnets of your network and then implement your firewall policy by dragging objects into policy rules. You can also take advantage of a large collection of predefined objects describing many standard protocols and services.Once a policy is built in the GUI, you can compile it and install it on the firewall machine.

## 1.2. Overview of Firewall Builder Features

- Over 100 predefined objects for the most popular and widely used protocols and services
- Ability to create custom objects describing IP,ICMP,TCP,UDP or custom services
- Ability to create objects to describe hosts, networks and address ranges
- Powerful wizards and other tools to help you jumpstart your firewall policy, and do it right from the first try. Wizards implement standard policies for typical network layouts, which can later be extended and edited by hand
- The Network discovery tool which automates the creation of multiple objects
- An object oriented approach to the policy management: any change made to an object gets immediately reflected in all policy rules of all firewalls using that object. The configuration for multiple firewalls shares the same object database.
- The ability to turn a firewall policy into a configuration file or a script and then install it on a firewall with just a few clicks of the mouse.
- Convenient GUI that supports standard Copy/Paste operations for both text strings and network objects and allows for policy editing using drag and drop
- Support for several firewall platforms, including Cisco PIX and Open Source firewalls iptables, ipfilter and pf.
- The ability to print a single object, a firewall policy, the whole hierarchy, or export it to the text file in either plain text or html format

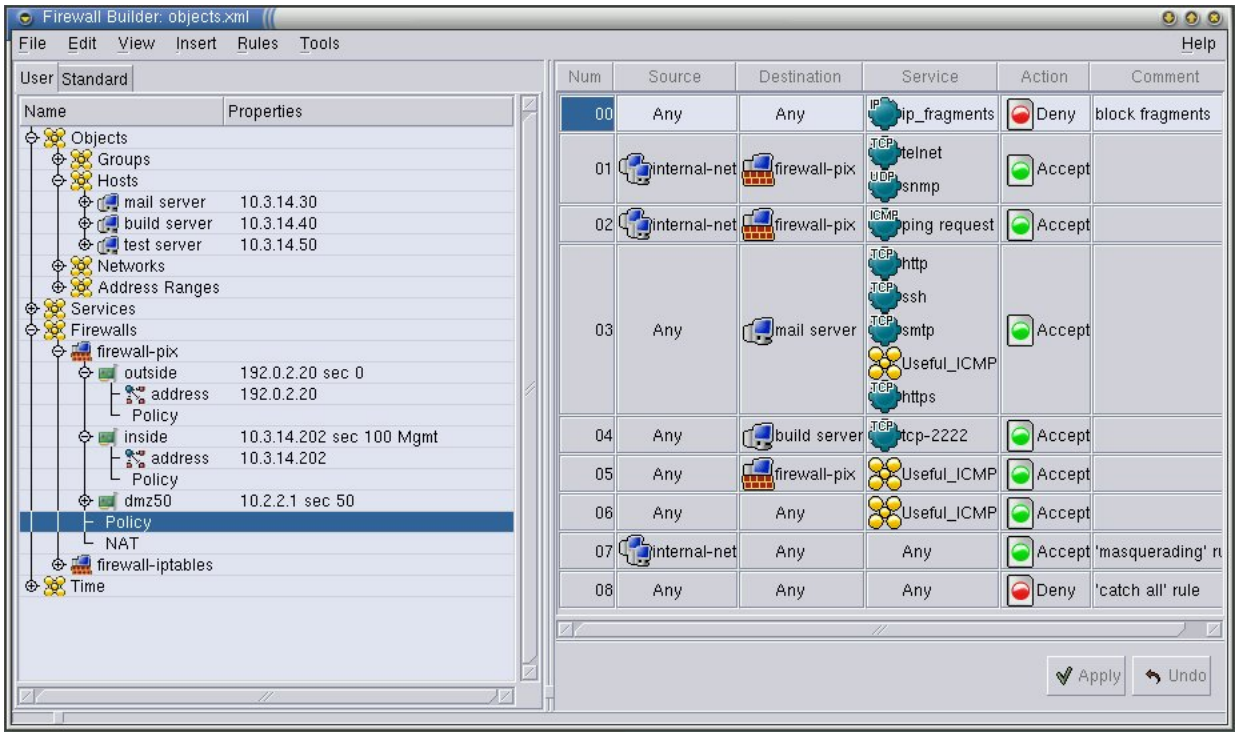



**Figure 1-1. Sample firewall policy**

# Chapter 2. Packaging

Firewall Builder is an Open Source project. Source code for the API library, GUI and policy compilers for iptables, ipfilter and pf is available for download from our web site at http://www.fwbuilder.org/. Firewall Builder for PIX is a licensed commercial product available as a binary package.

Installation instructions are available online at the following URL : http://www.fwbuilder.org/pages/Documents/Build.html . The same document outlines procedures for building Firewall Builder from the source code. Firewall Builder can be compiled and works on the following OS and distributions:

- Debian Linux
- Mandrake Linux
- RedHat Linux
- SuSE Linux
- FreeBSD
- OpenBSD
- Mac OS X
- Solaris

Document http://www.fwbuilder.org/pages/Documents/Build.html always has the latest up-to-date information about release and version numbers of these OS and distributions.

Firewall Builder comes in a several packages:

- Firewall Builder *API library*: package "libfwbuilder"
- Firewall Builder *GUI*: package "fwbuilder"
- Parts of GUI and policy compiler for *iptables*: package "fwbuilder-ipt"
- Parts of GUI and policy compiler for *ipfilter*: package "fwbuilder-ipf"
- Parts of GUI and policy compiler for *pf*: package "fwbuilder-pf"
- Parts of GUI and policy compiler for *Cisco PIX*: package "fwbuilder-pix"

You need to install the API library *libfwbuilder*, the GUI *fwbuilder* and at least one module that provides support for the target firewall platform; this could be either *fwbuilder-ipt* or one of the others.

Pre-built binary packages for many OS and distributions can be downloaded from our web site at http://www.fwbuilder.org/

Typical installation problems are addressed in the document "Firewall Builder Frequently Asked Questions (FAQ)" which can be found at the following URL: http://www.fwbuilder.org/pages/Documents/FAQ/FAQ.html

# Chapter 3. Firewall Builder GUI

## 3.1. The Main Window

The main window is divided into two parts: the objects tree and the dialog area. The object which is currently selected in the tree is highlighted in color and is shown in the dialog in the right half of the main window. You can create more than one main window using the main menu **"File"/"New Window"**. All windows are synchronized and work with the same object tree. Changes made to the object in one window are immediately reflected in all windows where it is visible, both in the tree and in the dialog. We recommend using the multi-window mode if you work with a firewall with a very large policy. In this case opening the policy may take some time and switching between the policy view and individual objects can slow the work down. Opening the policy in one window and working with individual objects in another helps avoid this problem.

Drag and drop and Copy/Paste operations work across different windows.

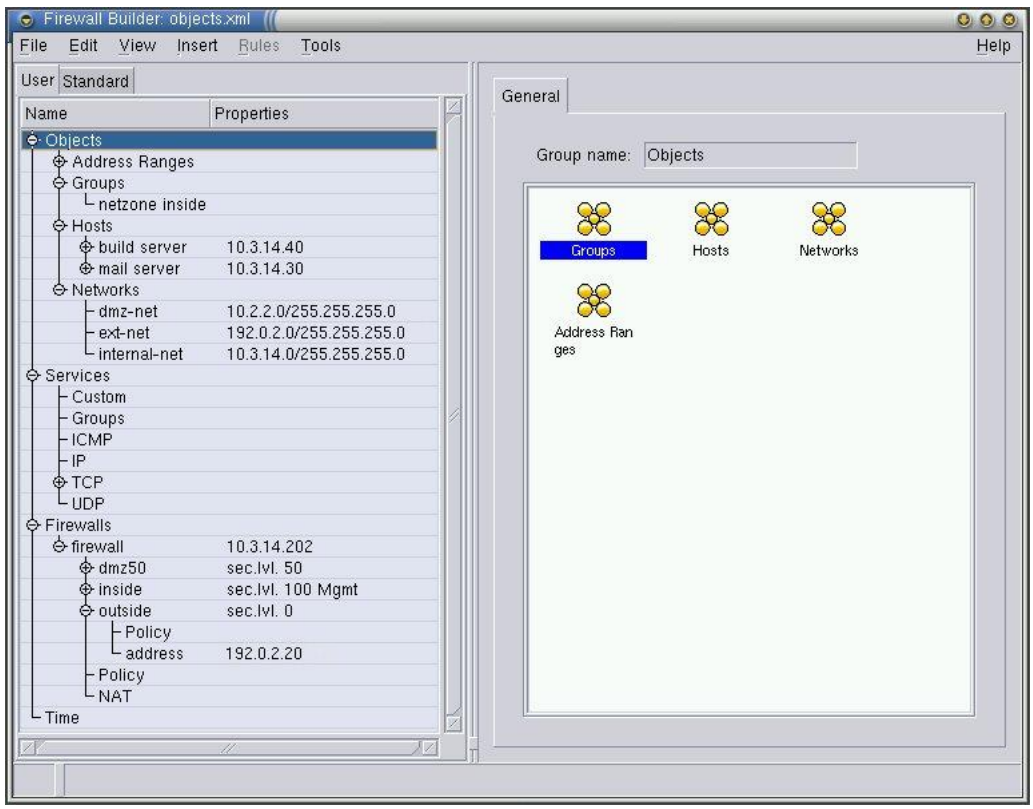



**Figure 3-1. The main window**

## 3.2. GUI Options

Certain aspects of the GUI behavior can be changed by the user. These are gathered in the dialog called "Options" which can be accessed through the main menu "**Edit/Options**". The GUI Options dialog is represented in Figure 3-2.

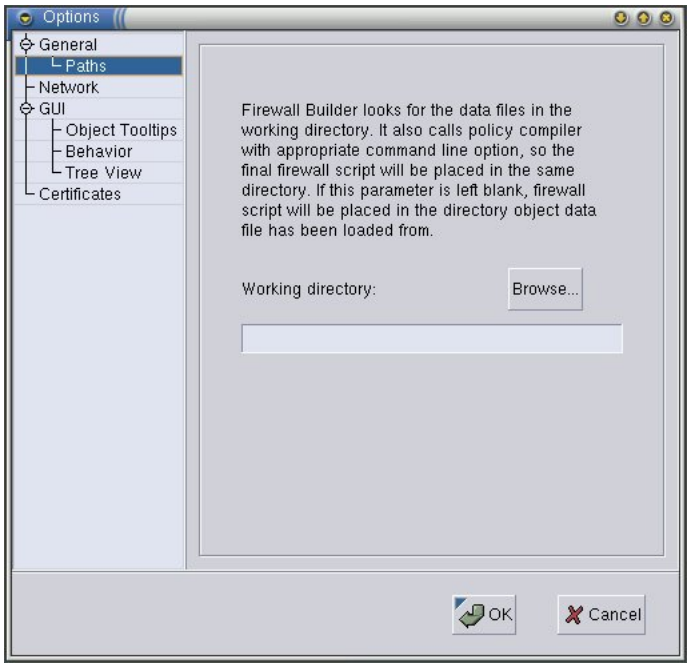



**Figure 3-2. The GUI Options Dialog**

- **General/Paths**
  - Working directory:

    This option tells the program where the data file should be stored. Policy compilers also store firewall configuration files or scripts they produce in the same directory. If this parameter is left blank, then the policy compiler will store the firewall configuration it generates in the same directory the original data file was loaded from.

- **Network**
  - SNMP Timeout and Retries

    Here you can configure timeout values and retry counters for SNMP queries. These values are used when the program runs a SNMP query to collect host "sysinfo" data and its interfaces. Note that the Network discovery Druid has its own settings for the SNMP timeout value and retry counter.

  - DNS Timeout and Retries

    Here you can configure timeout values and retry counters for DNS queries. These values are used when the program uses DNS to determine the host's address by its name.

- **GUI**
  - Remember window size at exit

    If this option is checked, the GUI remembers the main window size when the program finishes. The information is stored in the user's preferences file so that when the program is started again next time, it opens the window in the same size as before.

  - Hide navigation bar in object dialog

The GUI can show the navigation bar on top for the object's dialog. The navigation bar shows the current position in the tree and familiar buttons that allow navigation "up", "down" and "left" in the tree. If this option is checked, then the bar will disappear.

- **GUI / Object Tooltips**

  Firewall Builder can show the object's properties in a quick pop-up window (this feature is often called "tooltip") when the user places the mouse cursor over the object in a group view or in the firewall policy view, where objects are represented by their icons. This is a convenient way to quickly check the object's properties without having to leave the current dialog and open the object's dialog. There are three choices for the tooltip feature:

  - Off
  - Show properties in a tooltip pop-up window
  - Show properties on the main window's status bar

  For the second choice (showing in tooltip window) an additional parameter is avaliable; it controls the delay before the tooltip window appears ( after the mouse cursor is positioned over the object's icon.)

- **GUI / Behavior**
  - Automatically save data in dialogs while switching between objects

    The GUI shows a dialog for every opened object in the same place (the right half of the main window). Normally if the user has made some changes to the object's data and wants to switch to another object without clicking on the **"Apply"** or **"Undo"** button, GUI shows a pop-up dialog asking whether data should be saved or changes should be ignored. If this option is turned on though, the program automatically saves data every time the user leaves the dialog with unsaved data to open another object.

- **GUI / Tree View**

  As was mentioned before, objects can be organized into different libraries. Usually there are at least two libraries: "User" and "Standard". GUI can represent objects either in one large tree, or in a separate tree for each library.

  - Show object libraries in separate trees

    If this option is checked, the GUI shows objects belonging to different libraries in separate trees

  - Combine all libraries in a single tree

    If this option is checked, the GUI combines all objects in one large tree.

  - Show properties of objects in the tree

    If this option is checked, then GUI shows a brief summary of the object's properties, or one most frequently used property in an additional column in the tree view.

# Chapter 4. Object Tree

## 4.1. The Standard Tree Structure

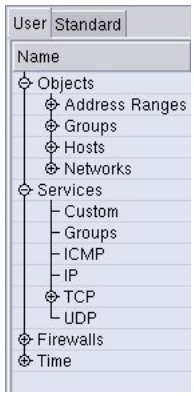



**Figure 4-1. Standard Tree Structure**

The object tree provides standard places for all object types so that they can be arranged in a hierarchy. Types that correspond to network objects (hosts, address ranges, networks and groups of these) are located in the "Objects" branch, types that correspond to all services are in the "Services" branch, time intervals are in the "Time" branch, and all firewalls are in the "Firewalls" branch. Newly created objects are automatically placed in their corresponding position in the tree. Each branch of the tree is automatically sorted by the object's name.

> **Note:** Future versions of Firewall Builder will support creation of custom branches in the tree and placing of objects and groups of objects in any branch.

The program can maintain all the objects in a single tree, or divide them into "libraries". The latter case is illustrated by Figure 4-2 where each library is represented by an individual tree; one tree is always called "User" and holds user-defined objects. The other tree is called "Standrard" and holds a collection of standard objects that come with the program. Some of the standard objects are visible in the screenshot (Figure 4-2.) Standard objects represent over 100 of frequently used protocols and services. There may be more libraries in the future. Splitting objects tree into "libraries" is just a convenient way to sort objects in the tree, you can think of it as a kind of the "view". It only affects representation of the data in the GUI; objects are all equal in all other senses and you can use an object from either part of the tree in your policy. You do not need to (and can not) insert the firewall in the "Standard" tree.

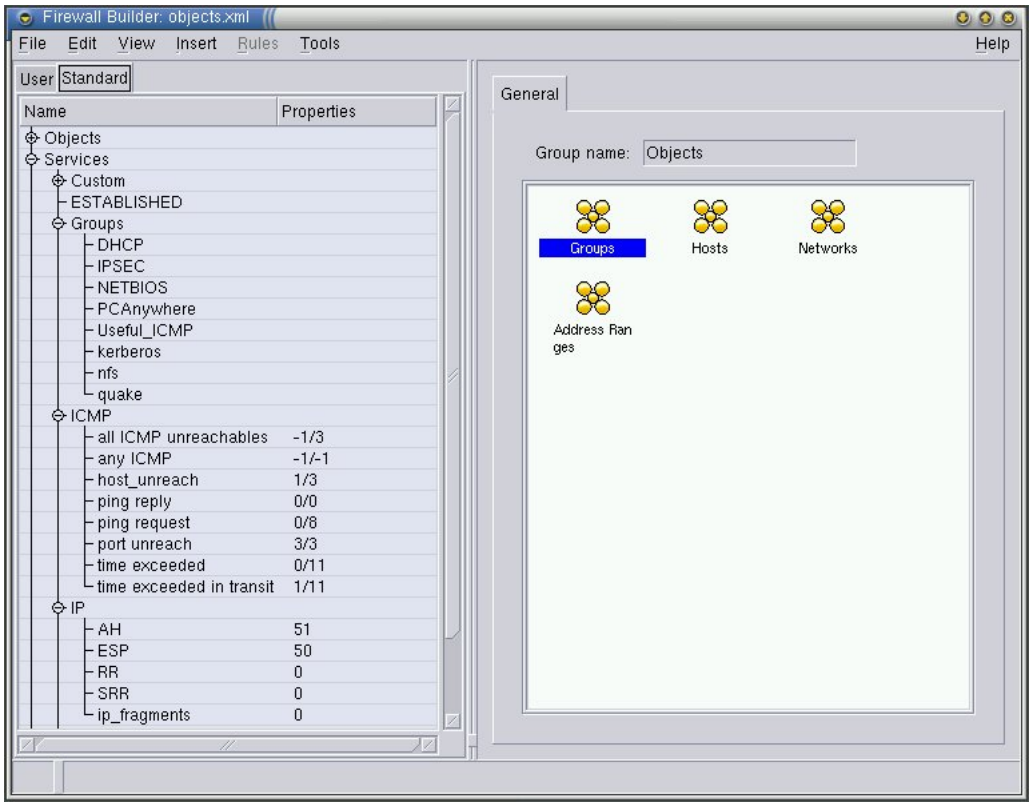



**Figure 4-2. Standard Objects**

## 4.2. Creating Objects

New objects can be created using the **"Insert"** main menu. Certain objects can also be created via a pop-up menu that can be opened by clicking the right mouse button on the object in the tree.

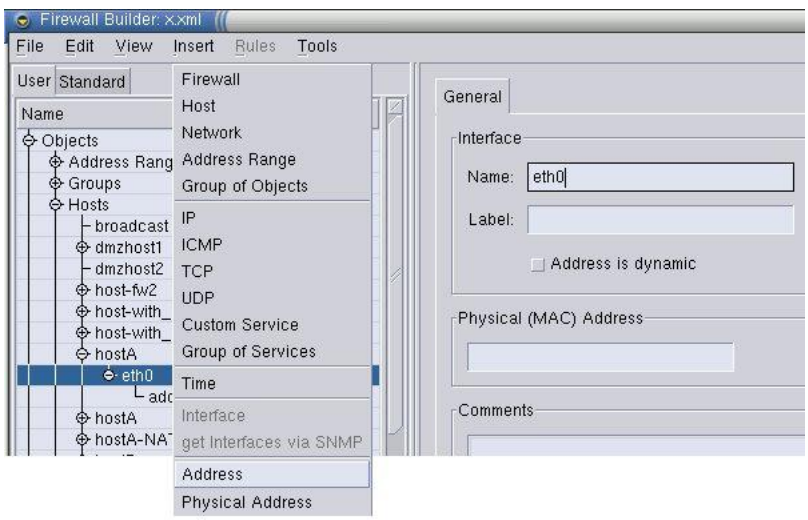



**Figure 4-3. Creating Objects Using The 'Insert' Main Menu**

You can create all objects through the main menu 'Insert' (Figure 4-3). Since some objects can be placed in the tree only under certain object types, corresponding menu items get enabled only when the right object is displayed in the dialog in the main window. For example, the interface object can only be placed under the host or the firewall, therefore the menu item **"Insert"/"Interface"** is enabled only if the host or firewall object is opened in the main window. Likewise, the menu items **"Address"** and **"Physical Address"** are enabled only when the interface object is opened.

Another way to create objects uses the pop-up menu which appears if you click the right mouse button on the object label in the tree (Figure 4-4).

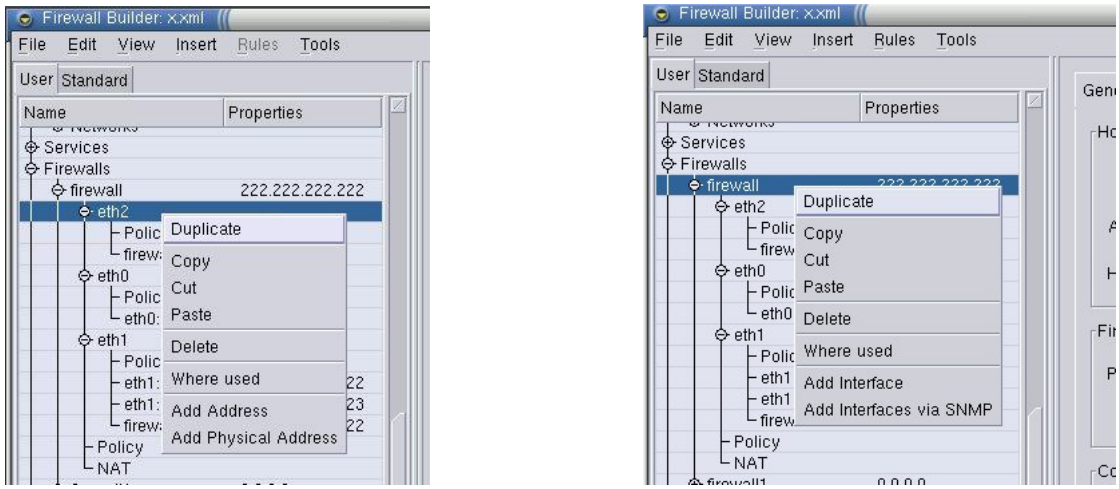



**Figure 4-4. Adding Objects Using The Pop-up Menu**

Items of this menu are context-sensitive just like items under the **"Insert"** main menu. In particular, menu items that add interfaces appear only when the object opened in the main window is either the host or the firewall (Figure 4-4). You can add interface objects by hand (using the **"Add Interface"** menu item) or import them via SNMP query (using the **"Add Interfaces via SNMP"** menu item)

## 4.3. Navigating The Object Tree and Editing Objects

The object selected in the tree automatically appears in the dialog area where it can be reviewed or modified. Objects are opened by a single mouse click in the tree. Clicking the right button when the mouse cursor is over the object's label in the tree brings up the pop-up menu. This menu is context-sensitive and provides items for the following functions (Figure 4-4):

- Duplicate:

  This function creates a copy of the current object, places it in the tree and opens it in the main window. The new object has the generic name "New Object", but once you change the name and click **"Apply"**, its name will immediately change in the tree.

- Copy:

  This copies an object onto the clipboard.

- Cut:

  This copies an object onto the clipboard and removes it from the tree.

- Paste:

  This creates a new object which is a copy of the current content on the clipboard.

- Delete:

  This deletes an object without making a copy on the clipboard. This is an unreversible operation and the program shows a pop-up dialog, asking the user to confirm.

- Where used:

  This scans the whole tree, including all groups and policies of all firewalls, looking for references to the current object. In the end the program shows a pop-up dialog with icons corresponding to groups and firewalls that use current object. Double-clicking on the icon opens the corresponding object in the main window.

The pop-up menu can also have items to add interface and address objects, depending on what type of object it is called for (Figure 4-4).

In general, GUI has been designed to "look and feel" like a typical file browser (e.g. Windows Explorer). It shows object groups as a collection of icons, where each icon represents an individual object (Figure 4-5). The user can navigate between objects either using the mouse or keyboard "arrow" keys. The active object is highlighted with a different color. Douple-clicking on the icon in the group view opens the corresponding object. Clicking the right mouse button or spacebar on the keyboard opens a pop-up menu, which acts as a shortcut to frequently used operations such as "Open Object", "Copy","Cut", and "Paste"

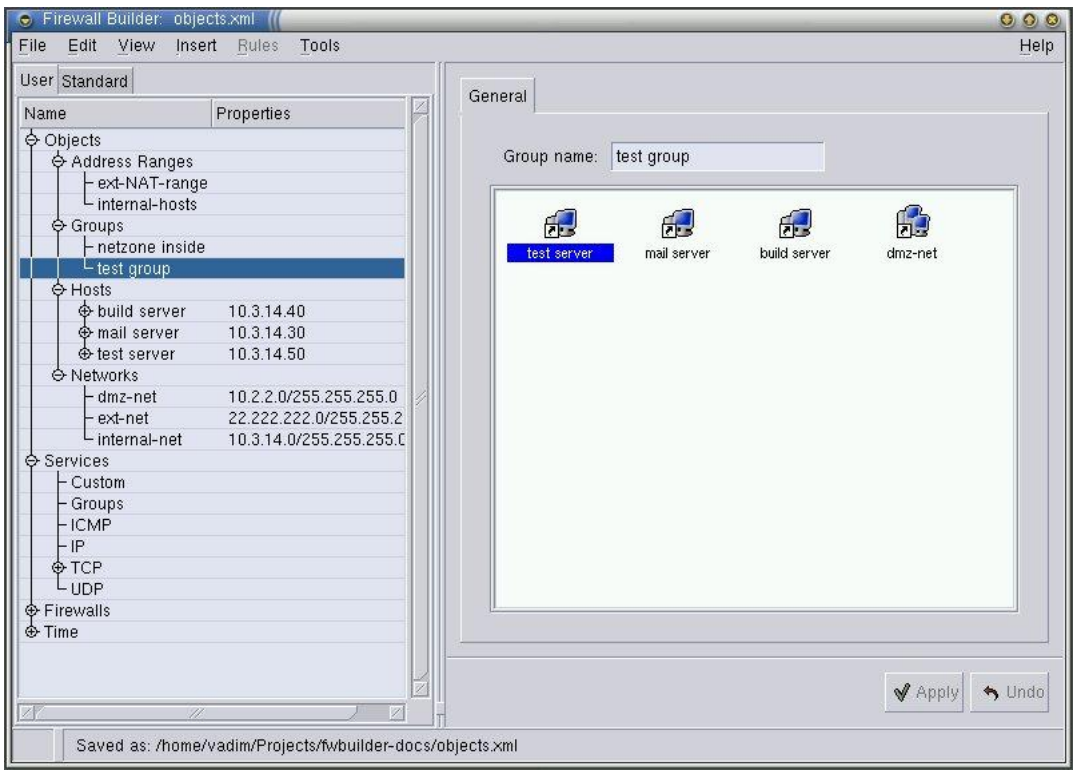



**Figure 4-5. Group View**

Dialogs for objects that can be edited have **"Apply"** and **"Undo"** buttons at the bottom. Initially these buttons are inactive, but once the object's parameters have been modified in the dialog, or group members have been added or removed, both buttons get activated so that the user can click **"Apply"** to save new data or **"Undo"** to reload original values into the dialog. If the user tries to switch to another object without either saving or reloading data in the current dialog, the program shows a pop-up dialog asking the user to take one of these actions. There is also an option to make the program automatically save modified data when the user switches between objects (see below).

# Chapter 5. Working With Objects

## 5.1. Common Properties

All objects visible to the user have some common editable fields, such as their name and comment. The object name can contain white spaces and can be arbitrarily long. The comment can contain any text of any length.

Dialogs for those objects that can have an address use the standard GUI widget that visually separates the four octets of IP address and provides "on the fly" control for the validity of an address.

## 5.2. The Host Object

The host object in Firewall Builder is designed to represent real hosts in the network: workstations, servers, and any other network node with an address. Just like real hosts, the host objects have interfaces, representing different physical connections to the network. Most internet hosts will have just a single (visible) interface with a single IP address, in that case the actual interface and its name do not matter. For most foreign hosts, fwbuilder will assign an arbitrary name “interface1” to the hosts interface. By using a tree-like hierarchy of hosts -> interfaces -> addresses it is possible however to specify the exact address and/or interface of a host in the case when it does matter. Both interfaces and addresses are represented by objects, which are organized in a tree. Interface objects sit in the tree directly under the host, and the address objects are located under their interfaces. The interface object can have either one or multiple addresses. Examples of hosts with one interface with multiple addresses are shown in Figure 5-1. Host “test server” is located on the LAN and has three virtual IP addresses that all belong to the same interface “eth0”. Host “www.netcitadel.com” is remote and also has three IP addresses. This host object may in fact represent a web farm that accepts connections on three IP addresses rather than a single computer. Host object in Firewall Builder is an abstraction used to represent both real hosts and addresses in general, regardless of their location in respect to the firewall. We will go over the features of interface and address objects below.

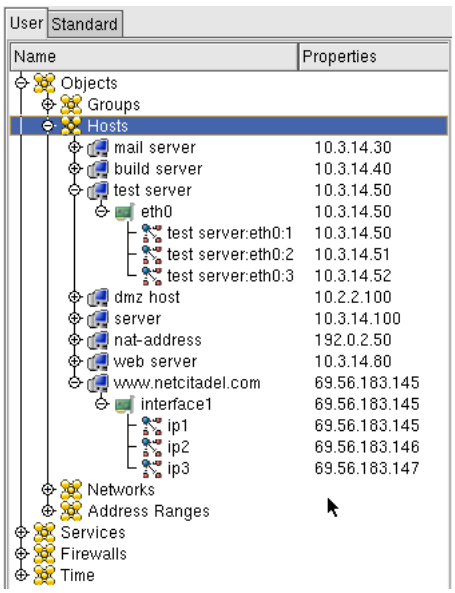



**Figure 5-1. A Host Object With One Interface And Multiple Virtual Addresses**

**Note:** The host object can not have any security or Network Address Translation policy associated with it; only firewall objects can.

### 5.2.1. Creating a Host Object




**Figure 5-2. First Page of the Druid**

To speed up the process and make it simpler, creating a new host object is aided by the Druid which collects all data needed for both the host and its interfaces and then creates the object. The screenshot (Figure 5-2) represents the first page of the Druid. First of all you need to enter the host name. Generally this name does not have to be the same as the real host's name. If the new host object has a single interface (this, perhaps, is the most common case) then you just need to fill the entry field for its address and maybe MAC address and click **"Next"**. If the object's name is the same as the real host name, then you can use the **"DNS Lookup"** button to get the address from DNS.

In case the new host object should have multiple interfaces, you need to check the option at the bottom of the dialog and then switch to the next page of the Druid using the **"Next"** button. The next page is represented in the screenshot (Figure 5-3.)

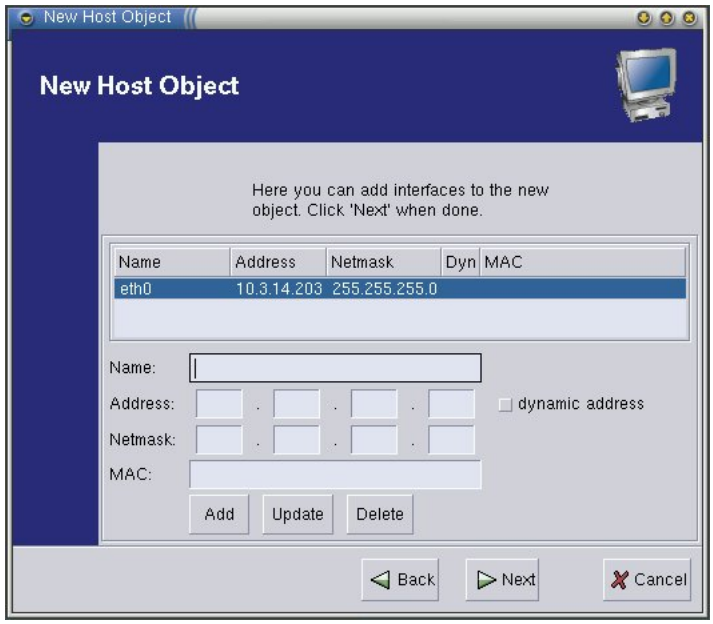



**Figure 5-3. Adding Interfaces to the New Host Object**

Here you can add interfaces to the new host object. Enter the interface name, address, and netmask in the appropriate fields, then click **"Add"** to add it to the list. The **"Update"** button updates information for the interface that is selected in the list, and the **"Delete"** button deletes the currently selected interface.

> **Note:** You can always add, modify and remove interfaces of the new host object later using controls provided by the main window and the object tree view.

### 5.2.2. Editing a Host Object

The Host object dialog is split into two tabs: "General" (Figure 5-4) and "Sysinfo" (Figure 5-5).

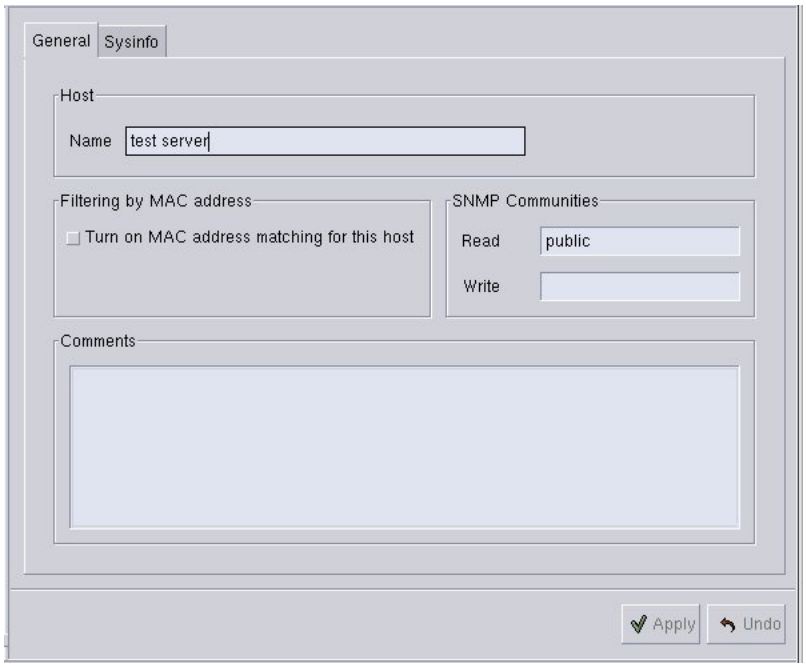



**Figure 5-4. Editing The Host Object**

The "General" tab provides GUI controls for the following parameters:

- Name:

  The Host object name.

- MAC address filtering:

  If this option is activated, the policy compiler uses the MAC addresses of all interfaces of this host in the firewall rules. Not all firewall platforms support MAC address filtering, so this option may have no effect on the generated firewall script. This is treated as a non-critical situation, and the policy compiler will only generate a warning while processing a firewall policy where such a host is used. See below Section 5.5.1.

- SNMP communities:

  The SNMP 'read' community is used if the host runs SNMP agent and you pull the host's "sysinfo" variables via SNMP query. The SNMP 'write' community is currently not used but we may add features using it in the future. SNMP communities usually can not contain spaces. If your host does not run SNMP agent, or if you do not need to pull the host info using SNMP, then SNMP community strings may be left blank.

- Comment:

  This is a free-form text field which can be used to add comments.

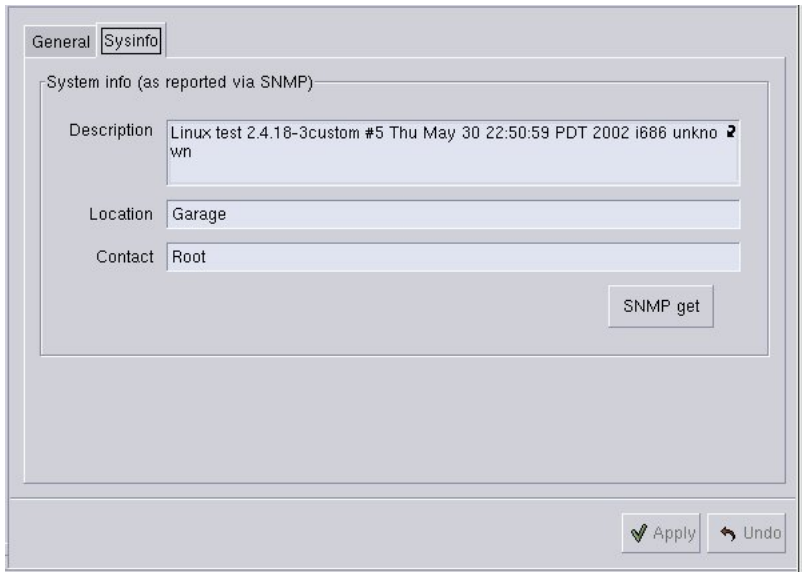



**Figure 5-5. Host Object System Information**

The "Sysinfo" tab holds data retrieved from the host via SNMP query. There, you can see the strings that the host returned for standard SNMP MIB "Description", "Location" and "Contact" variables. These fields are read-only in the GUI.

## 5.3. The Host's Interface

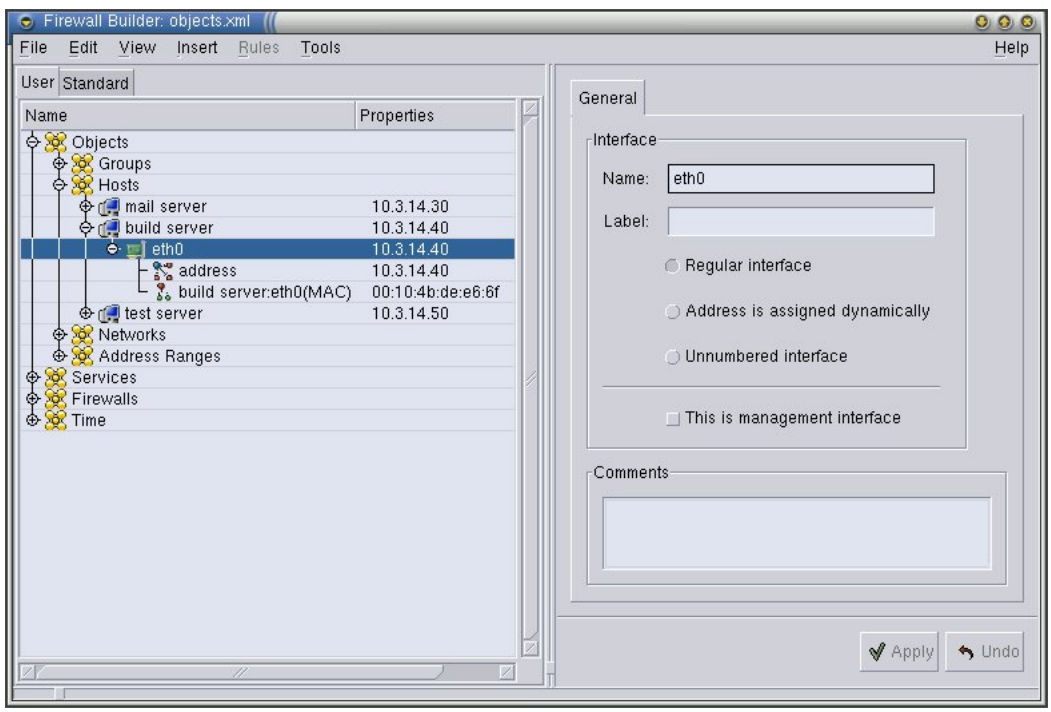



**Figure 5-6. The Host's Interface**

The dialog for the interface object that belongs to the host has only one tab named "General", which provides controls for the following parameters:

- Name

  This is the name of the object. Usually it should be the real interface's name. For example, on Linux this could be 'eth0' or 'eth1', on FreeBSD this could be 'lnc0' or 'fxp0', on Solaris this could be 'hme0', on Cisco PIX it could be 'ethernet0' and so on.

- Label

  On most OS this field is not used and serves the purpose of a descriptive label. Firewall Builder GUI uses a label, if it is not blank, to show interfaces in the tree. One of the suggested uses for this field is to mark interfaces to reflect the network topology ('outside', 'inside') or the purpose ('web frontend' or 'backup subnet'). The label is mandatory for Cisco PIX though, where it must reflect the network topology. See below for the chapter describing interfaces of the firewall

- The "Regular Interface" Option

  Use this option if the interface has an IP address assigned to it manually. This is the default.

- The "Address is assigned dynamically" Option

  Use this option if the interface has a dynamic address (obtained by means of DHCP or PPP or another protocol); in this case an address is unknown at the moment when Firewall Builder generates the firewall policy.

- The "Unnumbered interface" Option

  Use this option if the interface can never have an IP address, such as the ethernet interface used to run PPPoE communication on some ADSL connections, tunnel endpoint interface, or an interface on a bridging firewall. See below Section 5.3.1 for more detailed discussion of these different types of interfaces.

- "This is management interface" Checkbox

  Sometimes the host has several network interfaces in which case one of them can be marked as the 'management interface'. The management interface is used for all communication between Firewall Builder and the host. In the case of a host object this communication is currently limited to SNMP query to pull host information (standard SNMP MIB variables 'Description', 'Location', 'Contact'), however in the future we will add more.

- Physical (MAC) Address

  This field holds the interface's MAC address.

- Comment

  This is a free-form text field for a comment.

### 5.3.1. Interface Types

Firewall Builder recognizes three types of the interface depending on its IP address: *regular*, *dynamic* and *unnumbered*. *Regular* interface has an IP address assigned statically, that is the address does not change and Firewall Builder can use it in the firewall policy rules. It may need to use this address if the policy rule controls access to or from the firewall itself.

*Dynamic* interface has its IP address assigned to it dynamically by DHCP or PPP protocol. In this case the address is not known beforehand and can not be used in the policy rules. Some firewalls allow for using the interface name in the policy instead of its IP address; the firewall engine then picks its address either when the policy is activated or even at run-time. Some other firewalls support a special syntax for rules that are supposed to match packets headed to or from the firewall machine. Examples of these two cases are OpenBSD PF and Netfilter. PF rules can be constructed using interface names; PF automatically uses the current interface address when it loads rules into the memory. Netfilter supports special "chains" called "INPUT" and "OUPUT" that are guaranteed to inspect only packets headed for the firewall machine ("INPUT") or originated on it ("OUTPUT"). Both methods allow Firewall Builder to build correct firewall policy rules that affect the interface with a dynamic IP address, however the interface must be marked as such for the policy compiler to use proper technique depending on the target firewall platform.

*Unnumbered* interface can never have an IP address. Exapmples of such interfaces include endpoints of various tunnels (GRE, PPPoE, somtimes IPSEC), interfaces of a bridging firewall and perhaps some other cases. Firewall Builder ignores interfaces marked as *Unnumbered* and never includes them in the policy rules.

Interfaces of all three types can have policy rules associated with them. This means that in the GUI, any interface object can have "Interface Policy" object situated under it in the tree. The main difference is that *Regular* interface can always be used in the *Source* and *Destination* rule elements, *Dynamic* interface can be used there only on some firewall platforms, and *Unnumbered* interface can never be used in these policy rule elements. For example, in iptables and PF *dynamic* interface can be used in rules. PF can automatically use the address it has at the moment the firewall rules are loaded; iptables can not do this, but Firewall Builder provides emulation to work around this limitation.

## 5.4. The Address Object

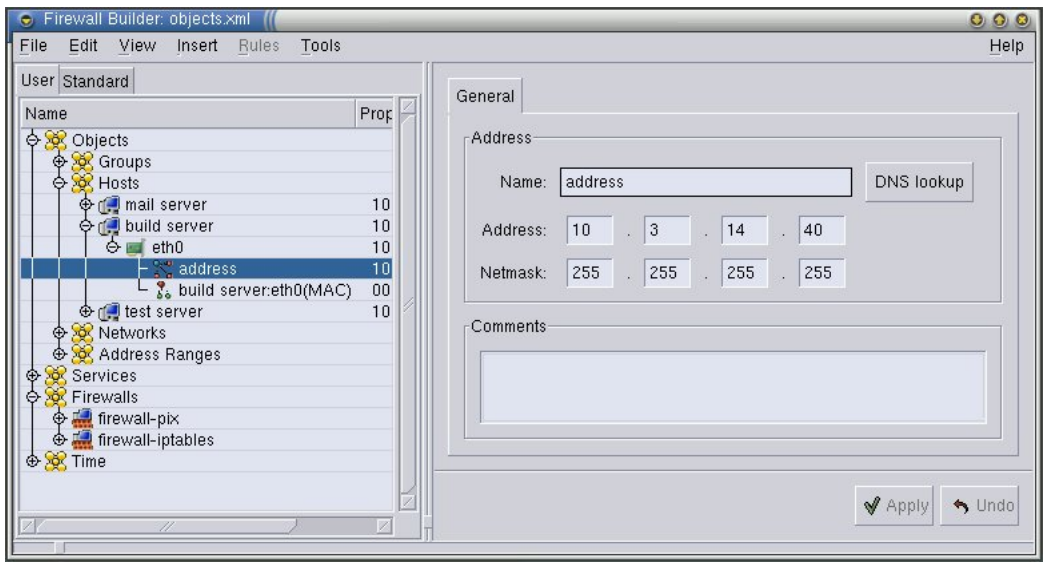



**Figure 5-7. The Address Object**

The address object describes the IP address of the interface; its dialog provides the following entry fields:

- Name

  This is the name of the object. It is recommended that you use some descriptive name because when the address object is used in the firewall policy, it is labeled with this name. It may be hard to tell one address from another if their names are similar.

- Address

  This is an IP address. The GUI widget visually separates octets and provides syntax control for the values entered in the octet fields.

  **Note:** A typical error is to interpret this object as an address of the subnet to which the interface of the host or firewall belongs. This object represents an address of the interface, not a network address.

- Netmask

  This is a netmask assigned to the interface.

- Comment

  This is free-form text field for a comment.

  **Note:** Currently Firewall Builder does not support IPv6 addresses, although support for IPv6 will be added in the future.

## 5.5. The Physical Address Object

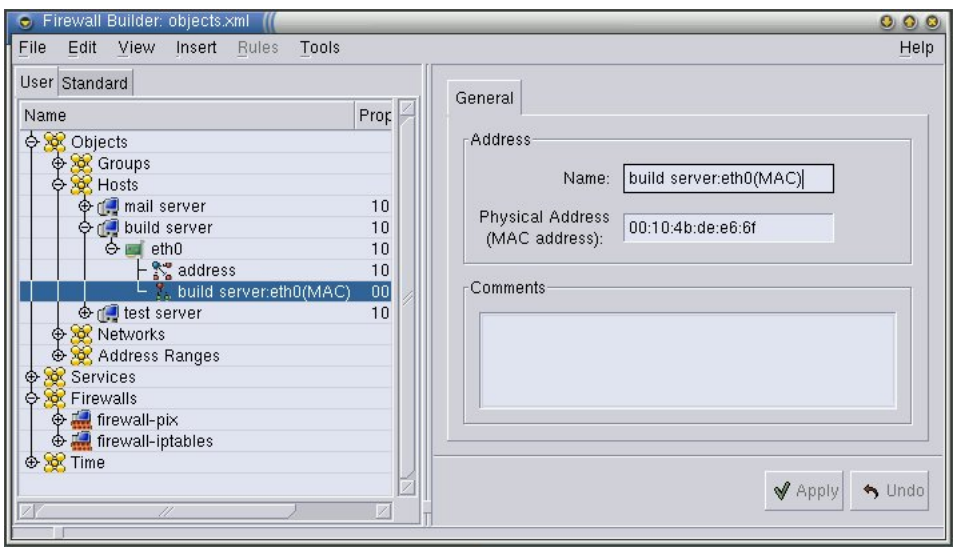



**Figure 5-8. The Physical Address Object**

the Physical Address object describes the hardware or media address. For example for Ethernet this would be a MAC address of the interface.

- Name

This is the name of the object. It is recommended that you use some descriptive name because when the address object is used in the firewall policy, it is labeled with this name. It may be hard to tell one address from another if their names are similar.

- Address

  This is a string representation of the physical or media address. For many types of media, this will be in a binary representation. For example, an ethernet address would be represented as a string of 6 octets.

- Comment

  This is free-form text field for a comment.

### 5.5.1. Using The Physical Address Object in the Policy

Only few firewall platforms really support physical address filtering. Currently netfilter is the only firewall platform that can do it and is supported by Firewall Builder.

As described in Section 8.5, if the Interface object that has multiple Address and Physical Address child objects is used in the policy or the NAT rule element (either *Source* or *Destination*), then the policy compiler tries to generate a rule using all of them. Section 8.5 explains that compiler actually does this by generating multiple rules using each address in turn. This roughly corresponds to using logical operation "OR" on these addresses: if our interface had two addresses *Address1* and *Address2*, then the generated rule would match if address in the packet is either *Address1* OR *Address2*. The case of a Physical Address is different though. If the Interface has a physical address, then compiler tries to build a set of rules that matches on combination of each IP address and its MAC address. The reason for this change is as follows:

Suppose we have a very important host on the network. We create a Host object, then add an interface to it. The interface should have both Address and Physical Address objects as shown on Figure 5-9

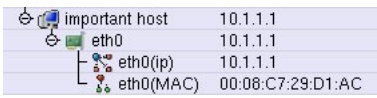



**Figure 5-9. The Host object with Address and Physical Address**

Because this is a very important host, we would like to be sure that packets whose source IP is that of this host are really coming from it and are not spoofed. The best way to achieve this goal is to use strong authentication, for example with IPSEC protocol. Using IPSEC is outside the scope of this document though; our goal right now is to show that inspecting MAC address of the packet can improve security.

Both the real packet originated from this host and a spoofed packet have a source IP address of the interface of this host, but a source MAC address is going to be different if spoofing is going on. We can use this to catch and drop spoofed packets. Here are three possible ways to build security policy in this situation:

- Using only Address object in the rule element. This means the firewall inspects only IP address and ignores the MAC address of the packets.




**Figure 5-10. Policy rule using only Address object**

- Using only Physical Address object. A rule built this way permits all kinds of traffic coming from the trusted host even if its IP address changes.




**Figure 5-11. Policy rule using only Physical Address object**

- Using Host or Interface object. This way we end up with a rule that matches on a *combination* of the IP address and MAC address. This may be used as a sophisticated anti-spoofing rule.




**Figure 5-12. Policy rule using Host object**




**Figure 5-13. Policy rule using Interface object**

Using Address and Physical Address objects in a rule is not the same as using a Host or Interface object to which these Address and Physical Address belong. As described in Section 8.5, using an object with multiple addresses or multiple objects in the rule element is like bundling them together using logical operation *OR*. If we were to put Address and Physical Address in the rule like in Figure 5-14, we would end up with policy matching packets that have the source address 10.1.1.1 or MAC address 00:08:C7:29:D1:AC, but not necessarily both at the same time. Any host that manages to pretend to have the IP address 10.1.1.1 would be able to send packets through our firewall even if its MAC address is different. To achieve our goal and make sure packets with the source IP 10.1.1.1 really belong to our important host, we should be checking its IP address and MAC address at the same time and let a packet through only if its IP address *AND* MAC address are what we expect them to be. That is why Firewall Builder treats physical addresses differently and generates firewall code that inspects both IP address and physical address.




**Figure 5-14. Policy rule using Address and Physical Address objects**

Firewall Builder generates firewall code to inspect MAC address only for Host objects with the option "MAC address filtering" turned on. If this option is off, the Physical Address object will be ignored even if it is present in the Host object's Interface. This is because Host objects created using the Network Discovery Druid ( Chapter 6 ) are often populated with both IP address and MAC address information which is available through SNMP query, but inspection of MAC addresses is rarely needed. The administrator can use the "MAC address filtering" option in the Host object to specify what Hosts he wants MAC addresses to be checked for.

## 5.6. The Network Object

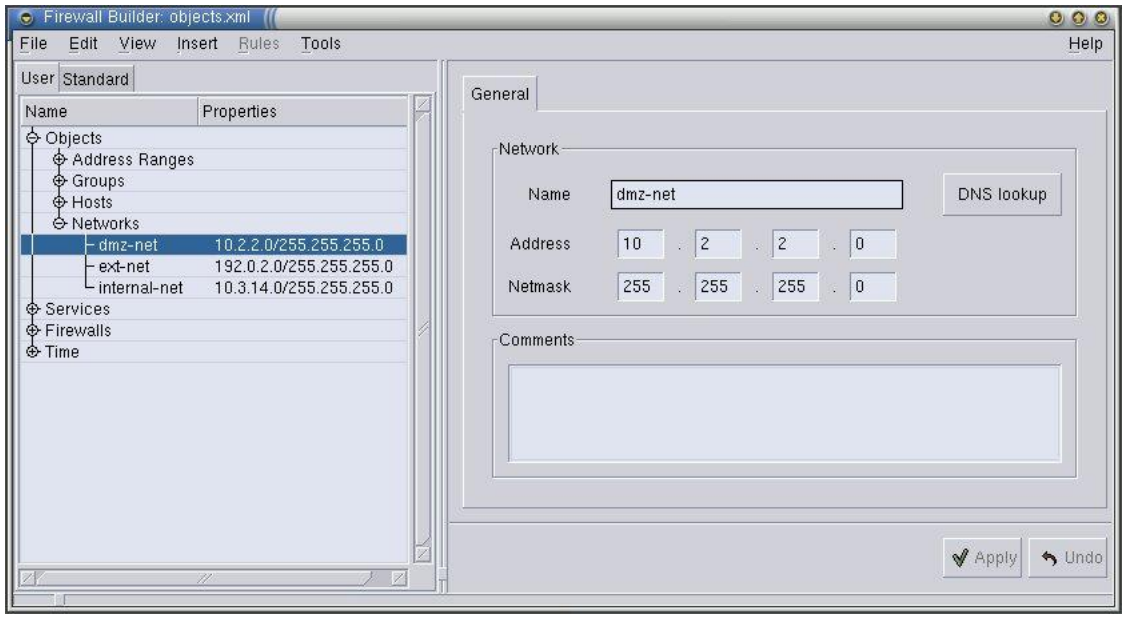



**Figure 5-15. The Network Object**

The network object describes the IP network or a subnet; its dialog provides the following entry fields:

- Name:
  Network object name
- 'DNS Lookup' button:
  Clicking on this button runs a DNS query for an "A" RR using the name entered in the "Name" field
- Address:
  This is an address of the network
- Netmask:
  The netmask, in combination with an Address, defines the subnet.
- Comment:
  This is a free-form text field used for comments

## 5.7. The Address Range Object

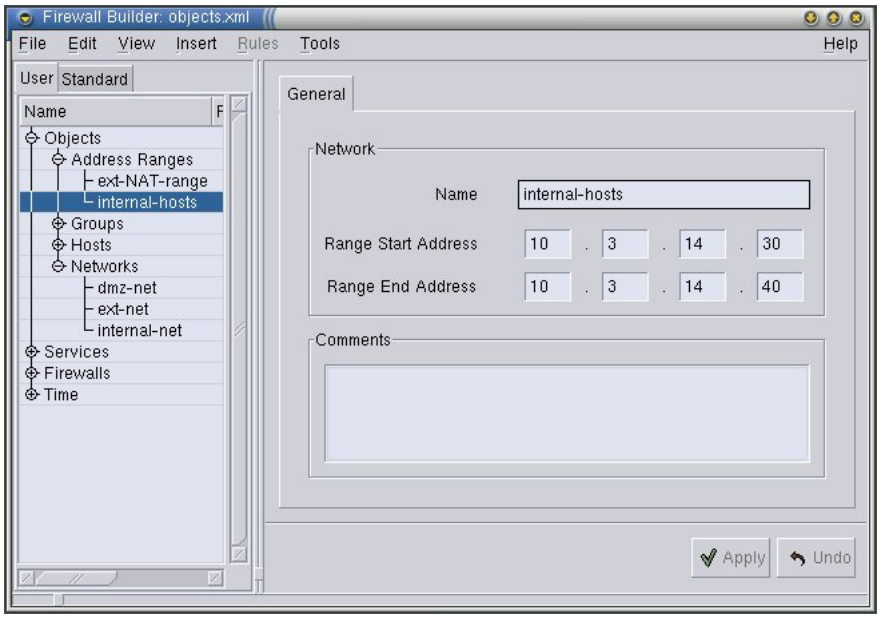



**Figure 5-16. The Address Range Object**

The Address Range object describes a continuous range of IP addresses; its dialog provides the following entry fields:

- Name:

  The name of the Address Range object

- Range start:

  The address of the start of the range

- Range end:

  The address of the end of the range

- Comment:

  A free-form text field used for comments

The Address range is inclusive, that is, both the start and the end addresses are included in the range.

## 5.8. A Group of Objects

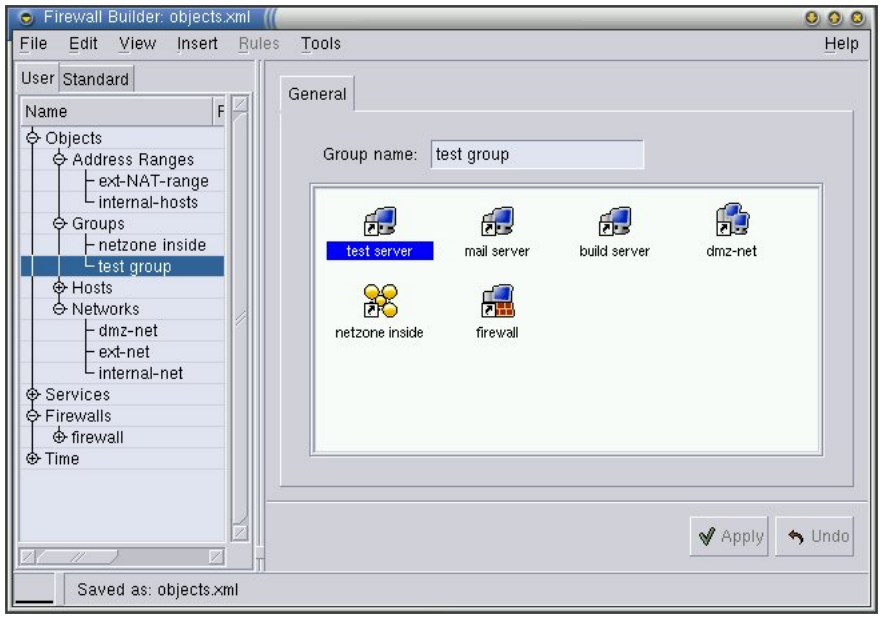



**Figure 5-17. Group of Objects**

The group of objects holds references to Hosts, Networks, Address Ranges, Firewalls and other groups of objects (Figure 5-17). Objects can be added to the group using the following methods:

- Using drag and drop:

  Objects can be dragged from the tree into the group dialog. The tree switches to the object in a single click, however pointing a mouse cursor at the object and then pressing the button does not switch, but rather allows to start a drag and drop operation.

- Using the popup menu:

  You can use the "Copy"/"Paste" operations between the tree and group dialog. Right mouse clicking on the object in the tree brings a pop-up menu. Choose the "Copy" or "Cut" item in this menu, then switch to the group dialog and right mouse click in the icon field. This also brings up a pop-up menu, where you choose the "Paste" item. This inserts a reference to the object in the group.

- Using the "Edit" main menu:

  Just like in the case of a pop-up menu, select the object in the tree, use the **"Edit"/"Copy Object"** main menu, then switch to the group dialog and use the **"Edit"/"Paste Object"** main menu

## 5.9. The Firewall Object

### 5.9.1. Creating a Firewall Object

Similar to creating of a Host Object, creating a new Firewall Object is aided by the Druid. The first page of

this Druid is represented in a screenshot ( Figure 5-18 ). First of all you need to specify the name for the new firewall; then click **"Next"**. This brings you to the next page of the Druid (Figure 5-19).

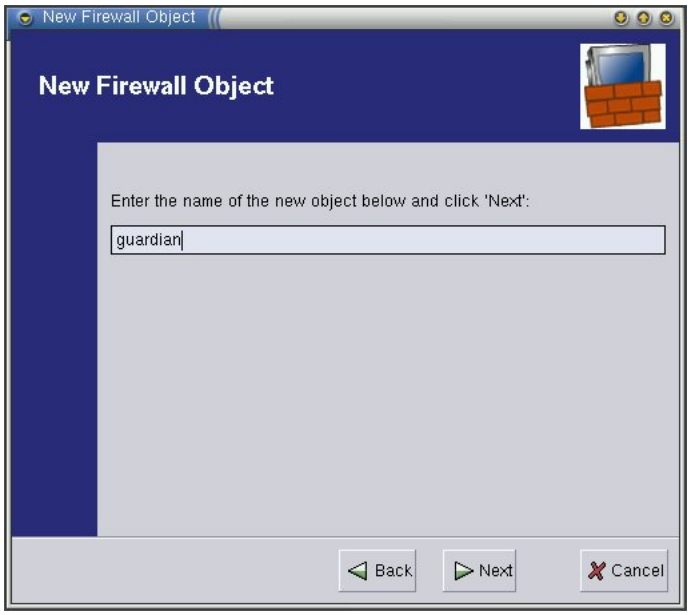



**Figure 5-18. First Page of the Druid**

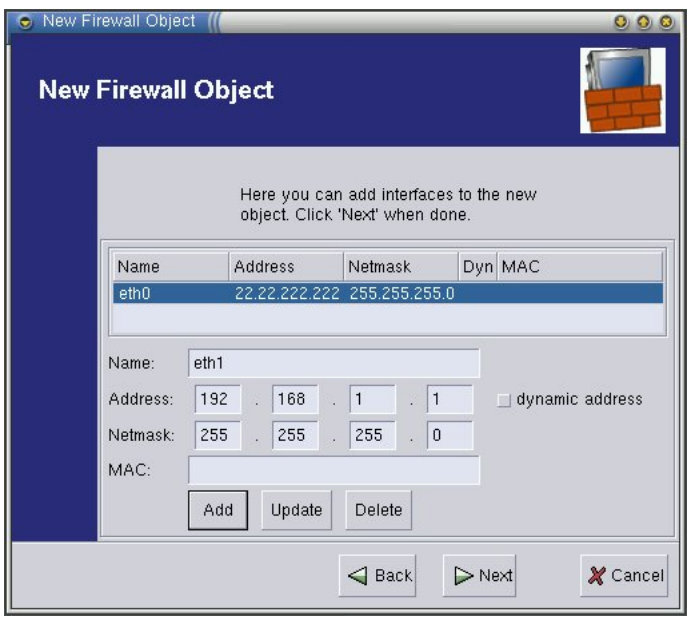



**Figure 5-19. Adding Interfaces to the new Firewall Object**

This page is similar to that of the new Host object druid page where you add interfaces. You enter the name, address and netmask of the interface in the appropriate fields, then click **"Add"**. The **"Update"** button updates parameters of the currently selected interface and the **"Delete"** button deletes it. Once all the interfaces are configured, click **"Next"** to create a new firewall object.

**Note:** You can always add, modify and delete interfaces later using controls provided by the main window and the object tree view.

### 5.9.2. Editing a Firewall Object

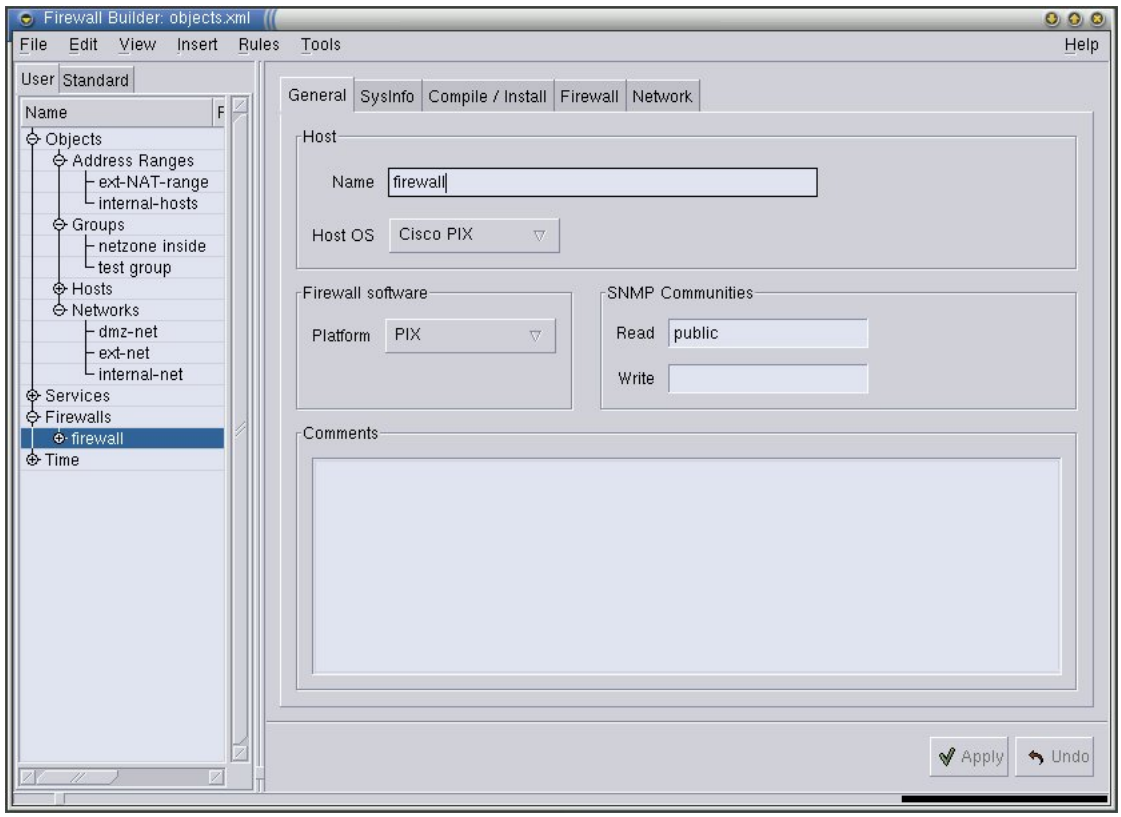



**Figure 5-20. Firewall Object, 'General' tab**

The Firewall Object represents the firewall machine and is the most complex object in Firewall Builder. Its dialog has the following tabs: 'General', 'Sysinfo', 'Compile/Install', 'Firewall' and 'Network'

The *General* tab provides the following controls:

- Name:

  This is the name of the object

- Host OS:

  Modules that provide support for target OS and firewall platforms are distributed separately from the GUI and can be installed in various combinations. This pull-down menu provides a choice of the host OS and is dynamically updated depending on the combination of installed modules.

- Firewall platform:

Like the "Host OS" menu, this pull-down menu provides a choice of the target firewall platform and is dynamically updated depending on a combination of installed modules.

- SNMP Communities:

  The SNMP "read" community is used to pull "sysinfo" MIB variables from the firewall machine, provided it runs a SNMP agent. It is also used if you want to automatically create interfaces for the firewall using SNMP query. The SNMP "write" community is not used at this time.

- Comment:

  This is a free-form text field used for comments

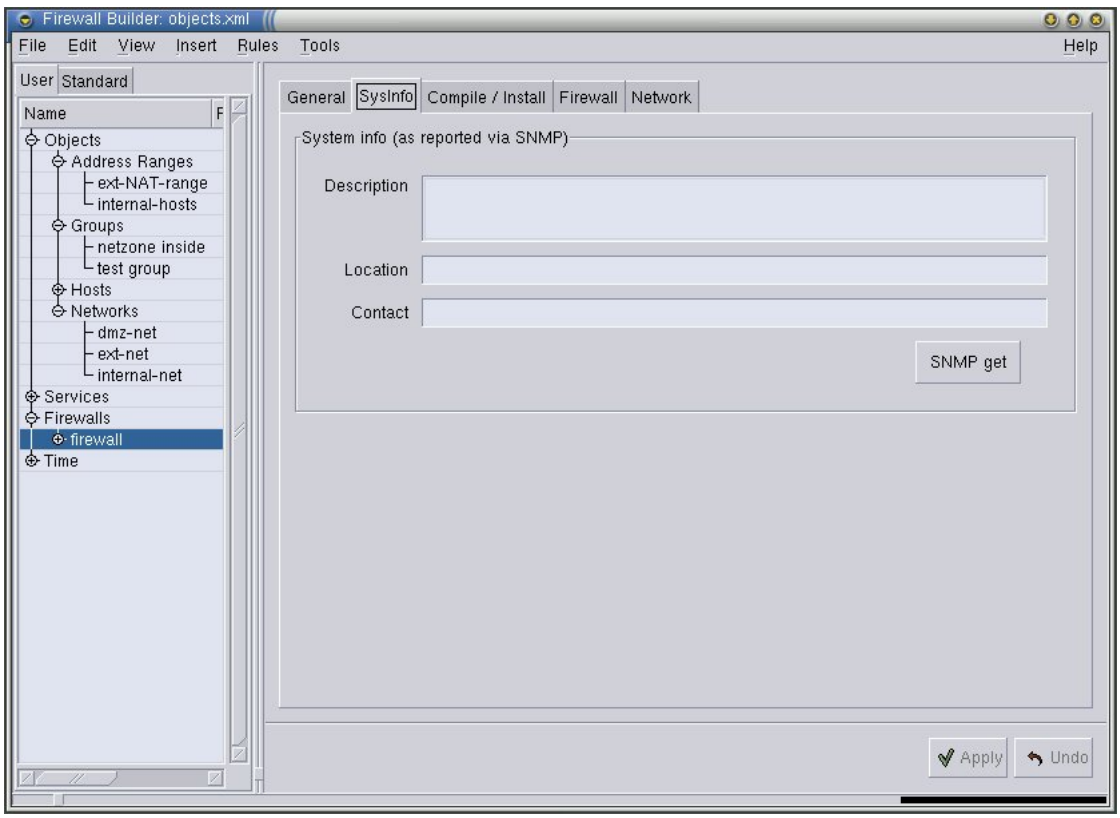



**Figure 5-21. Firewall Object, tab 'Sysinfo'**

This tab holds the following fields:

- Description:
- Location:
- Contact:

These fields present values of corresponding MIB variables.

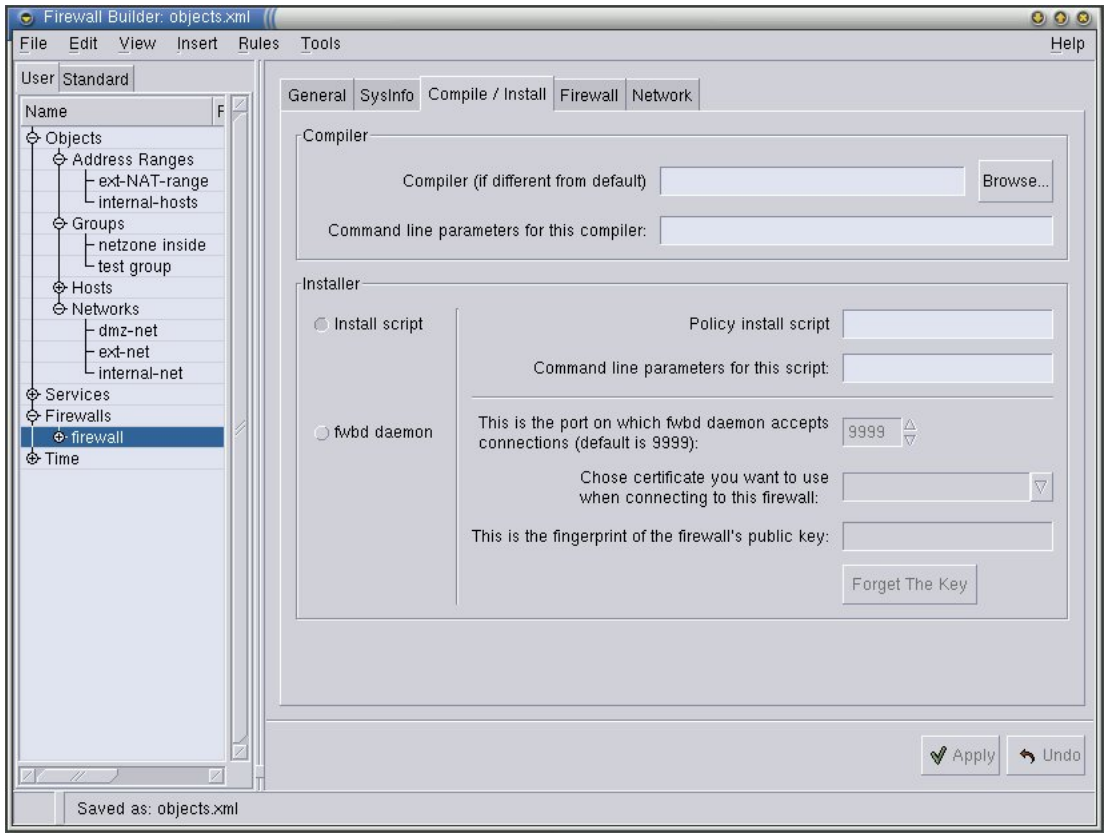



**Figure 5-22. Firewall Object, 'Compile/Install' tab**

This tab holds the following fields:

- Compiler:

  The full directory path and file name for the policy compiler program that the GUI invokes when you use the **"Rules"/"Compile"** main menu item. If this field is left blank, then the GUI calls the default policy compiler program which is preconfigured for the firewall platform chosen in the 'General' tab. Use this field if you use a non-standard policy compiler, or if you use a wrapper script which calls a policy compiler.

- Command line parameters for the compiler:

  This is where you can specify command line parameters for the policy compiler. Here is the list of command line options supported by all policy compilers:

  - *"-f":* specifies the data file name to be used by the compiler
  - *"-d":* specifies the working directory where the data file is located and where produced firewall script should be placed
  - *"-v":* turn on verbose diagnostics print during compile

  Normally the GUI automatically supplies "-f" and "-d" options when it calls the policy compiler. See the manual page for the corresponding policy compiler for the complete set of supported command line options.

- Installer: install script

In the future there will be two ways to install and activate a policy on the firewall machine: using install script or fwbd daemon. The daemon is not supported yet, so the install script is the only method available at this time. The GUI calls the install script when you use the main menu **"Rules"/"Install"** item.

- Istaller: fwbd daemon:

  *Not supported*

- Policy install script:

  The full directory path and file name for the install script.

- Command line parameters for the script:

  This is where you specify command line parameters for the install script. The GUI expects the install script to accept the same command line parameters as all policy compilers (see above) and automatically supplies "-f" and "-d" options. You can add more parameters in this entry field if your script supports them.

- Options for fwbd:

  Since the current version of Firewall Builder does not come with fwbd and does not support it yet, these options are reserved for future use.

*'Firewall'* and *'Network'* tabs provide fields specific to the chosen firewall platform and host OS configurations.

## 5.10. Configuring Target Firewall Software

The Firewall object's 'Firewall' dialog tab provides a set of controls to adjust the behavior of the policy compiler and underlying firewall software. This dialog tab changes when the user changes setting in the 'Firewall Platform' option in the 'General' tab.

### 5.10.1. Generic Parameters

These are compilation-time modes and algorithms that are common for all target firewall platforms.

#### 5.10.1.1. Rule Shadowing

Rule shadowing happens when a certain rule is a superset of a subsequent rule and any packets potentially matched by the subsequent rule have already been matched by the prior rule. For example, if rule #1 uses the network object for the source and rule #2 uses the host object located on that network for the source, rule #1 'shadows' rule #2 because any packet matched by #1 will never be matched by #2. This may be important if, say, rule #1 permits and #2 denies access to the same destination. If rule #1 'shadows' rule #2, then rule #2 will never fire. This situation is most often an error; the compiler can detect it and abort processing of the policy with an appropriate error message.

#### 5.10.1.2. Find and eliminate duplicate rules

Policy compiler can detect and eliminate rules that lead to identical code in the target firewall configuration.

#### 5.10.1.3. Empty Groups

This option is specifically designed to help use the following trick. It is sometimes necessary to control access to an often changing group fo hosts or networks. In order to avoind having to make changes to the policy rules, these hosts or networks should be put in a group object which is then used in the policy rule. You can then add and remove hosts to this group and regenerate the policy. This simple technique works just fine until you get a rule where this group is a sole object in the rule element (source or destination) and by some reason this group has become empty because you had to remove all the objects from it. The policy compiler should be careful not to treat an empty group as 'any' because that would be something exactly opposite to the intended meaning of the rule.

If this option is off, the compiler treats the empty group which is used as a sole object in the rule element as an error and aborts processing firewall policy with an appropriate error message.

If this option is on, the compiler tries to remove all empty groups from the rule element. If after the last one has been removed the rule element becomes empty (that is, 'any'), compiler automatically disables this rule and continues.

#### 5.10.1.4. Assume firewall is part of *'any'*

On some firewall platforms there is a difference in the configuration command used to control access through the firewall versus the command that controls access to the firewall itself. For example, in iptables, packets headed for the firewall machine should be inspected in the INPUT chain, while packets going through the firewall are inspected in the FORWARD chain. If this option is ON, the compiler generates code for both FORWARD and INPUT chains if it sees *'any'* as the destination of the policy rule. In other words, it assumes that the firewall is part of *'any'*, and packets may either go to or through the firewall. If this option is off, the compiler only generates code for the FORWARD chain. Compiler treats *'any'* in the source of the rule in a similar way, generating code either for the OUTPUT and FORWARD, or only for the FORWARD chain.

Another example of this is PIX, where one should use special commands to control access to the firewall on *telnet*, *ssh* and *icmp* protocols.

#### 5.10.1.5. Log Level

This option usually refers to the syslog log level and is used with firewalls support logging via the syslog protocol. Here is the list of the standard log levels: 'debug', 'info', 'notice', 'warning', 'error', 'crit' and 'alert'

#### 5.10.1.6. Log facility

This option usually refers to the syslog facility parameter and is used with firewalls that support logging via the syslog protocol. The facility is one of the following keywords: auth, priv, cron, daemon, kern, lpr, mail, mark, news, syslog, user, uucp and local0 through local7.

#### 5.10.1.7. Custom Logging Prefixes

The custom string may show up in the log together with information logged for each rule that has the 'logging' option activated. Firewall Builder supports the following macros in the log prefix that are expanded at the compile time:

%N rule number in the GUI.

%A rule action

%I interface the rule is associated with

%C (iptables only) iptables chain that this rule generated code for.

#### 5.10.1.8. Create Virtual Addresses for NAT Rules

The compiler can generate commands to add a virtual address to one of the interfaces of the firewall machine if this option is turned on. This is needed if a NAT rule uses an IP address that does not belong to any interface of the firewall. The firewall either needs the static 'published' ARP entry for this address, or it should be added to one of the interfaces as an 'alias' or virtual address.

The policy compiler adds code to add an 'alias' address to the interface on top of the firewall activation script. Depending on the OS the firewall is running on, it may use `ifconfig` (FreeBSD, OpenBSD, Solaris) or `ip` (Linux) to do that.

The activation script also adds virtual addresses to interfaces of the firewall if such addresses are configured in the firewall object in the GUI but are missing on the real firewall.

#### 5.10.1.9. Default setting for rules using the 'Reject' action

When the rule with the 'Reject' action matches the packet, it not only blocks it, but also sends a message back to the sender to notify them that their original packet has been denied access. The firewall can send back several types of ICMP 'unreachable' messages or a TCP RST packet, depending on the setting of this option and the service object used in the rule.

#### 5.10.1.10. Turn on debugging in the generated script

This option makes the generated firewall script print all commands when it is executed. Depending on the format of the target firewall configuration, the policy compiler chooses appropriate command line option or a configuration flag to indicate debugging. For example, configuration for iptables is just a shell script. In this case the compiler adds `"-x"` to the shell command line. For PF, the compiler generates two files: an activation script and a `"pf.conf"` file. In this case, the compiler not only adds `"-x"` to the shell command line but also passes `"-v"` to the `pfctl` utility, which activates the firewall confguration.

### 5.10.2. iptables

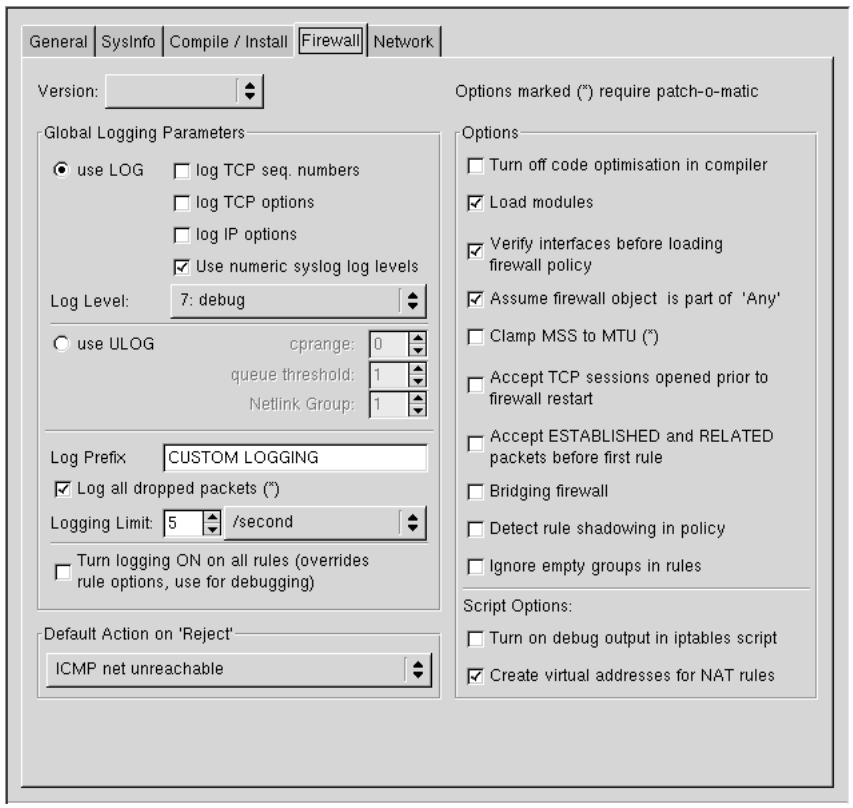



**Figure 5-23. Firewall Object, 'Firewall' tab for iptables**

**Table 5-1. General**

| GUI control | Description |
|---|---|
| **Version:** | Version of iptables used on the firewall machine. |

**Table 5-2. Global Logging Parameters**

| GUI control | Description |
|---|---|
| **Use LOG** | Use standard LOG target for logging (as opposed to using ULOG) |
| **log TCP seq. numbers** | Log TCP sequence numbers. |
| **log TCP options** | Log options from the TCP packet header. |
| **log IP options** | Log options from the IP packet header. |
| **Use a numeric syslog levels** | Use numeric setting for "--log-level". This option was introduced as a workaround for a bug in iptables 1.2.3 which could not interpret properly the standard symbolic syslog level (priority) names 'debug', 'info', 'notice', 'warning', 'error', 'crit' and 'alert' |

| GUI control | Description |
|---|---|
| **Log level** | See Section 5.10.1.5 |
| **Use ULOG** | Use ULOG target for a more advanced packet logging mechanism. The ULOG target is only available if the corresponding patch is applied to iptables. See http://www.gnumonks.org/projects/ulogd |
| **cprange** | The number of bytes to be copied to userspace. A value of 0 always copies the entire packet, regardless of its size. Default is 0 |
| **queue thrershold** | Sets the amount of packets batched together in one multipart netlink message. |
| **Netlink Group** | This specifies the netlink group (1-32) to which the packet is sent. Default value is 1. |
| **Log Prefix** | See Section 5.10.1.7 |
| **Log all dropped packets** | This option makes the firewall log all the packets it drops, including those that do not match any entry in the state table and some others. This option can only be used if the corresponding patch has been applied to iptables running on the firewall. The patch is called 'dropped-table' and is part of patch-o-matic. |
| **Logging limit:** | This option allows to set a limit on the number of log entries that the firewall records per a unit of time. This could be useful on high traffic firewalls, where writing too much to the log can overload the disk subsystem. |
| **Turn logging ON on all rules** | This option overrides the logging option in each rule to make the firewall act as if logging was turned on in each rule. This is useful for debugging of the policy. |

**Table 5-3. Options**

| GUI control | Description |
|---|---|
| Default action on 'Reject' | See Section 5.10.1.9 |
| **Turn off optimization in compiler** | This option turns off some optimization algorithms in the policy compiler. |
| **Load modules** | The compiler generates code to load iptables kernel modules before activating policy rules. |
| **Verify interfaces before loading firewall policy** | The compiler generates code to check if interfaces configured in the firewall object exist in the real firewall machine. This check is executed before policy rules are loaded. If the check fails, firewall policy won't be modified. |
| **Assume firewall object is part of 'any'** | See Section 5.10.1.4 |
| **Clamp MSS to MTU** | This option allows you to examine and alter the MSS value of TCP SYN packets and control the maximum size for that connection. This option can only be used if the corresponding patch from patch-o-matic is applied to iptables running on the firewall. |

| GUI control | Description |
|---|---|
| **Accept TCP sessions opened prior to firewall restart** | If this option is ON, the firewall will accept TCP sessions opened before the firewall software has been activated. These sessions are special because the firewall never saw their initiation phase. If this option is on, the firewall creates an entry in the state table when it sees a packet that apparently belongs to the unknown TCP session. |
| **Accept ESTABLISHED and RELATED packets before the first rule** | This option generates an implicit rule to accept ESTABLISHED and RELATED packets before the very first rule specified in the GUI. This helps to reduce the size of the policy script generated by the policy compiler. ESTABLISHED packets are basically reply packets coming back from the server to the client. Iptables keeps state information about each TCP session opened through the firewall and thus can detect and permit these kind of packets automatically. RELATED packets belong to another stream somehow associated with the initial stream opened through the firewall. FTP protocol is a good example of this situation. FTP uses two TCP sessions: command channel and data channel. You do not need to add a special rule for the data channel to your policy because iptables 'understands' when it needs to permit packets that belong to the data channel that has been created per request sent via a known command channel. This option is ON by default. If for some reason you do not want to have an implicit rule to permit these types of packets on top of all the rules defined in the GUI, you should uncheck this option. Remember that in this case you need to add a rule to permit these types of packets yourself. You can use the Custom Service object named 'ESTABLISHED' in this rule. This service object can be found in the 'Standard' object tree. |
| **Bridging firewall** | This option changes algorithms used by the policy compiler and makes it generate script for a bridging firewall. A bridging firewall treats broadcast packets differently, never uses INPUT and OUTPUT chains, and has some other differences. |
| **Detect rule shadowing in policy** | See Section 5.10.1.1 |
| **Ignore empty groups in rules** | See Section 5.10.1.3 |
| **Turn on debug output in iptables script** | See Section 5.10.1.10 |
| **Create virtual addresses for NAT rules** | See Section 5.10.1.8 |

### 5.10.3. ipfilter

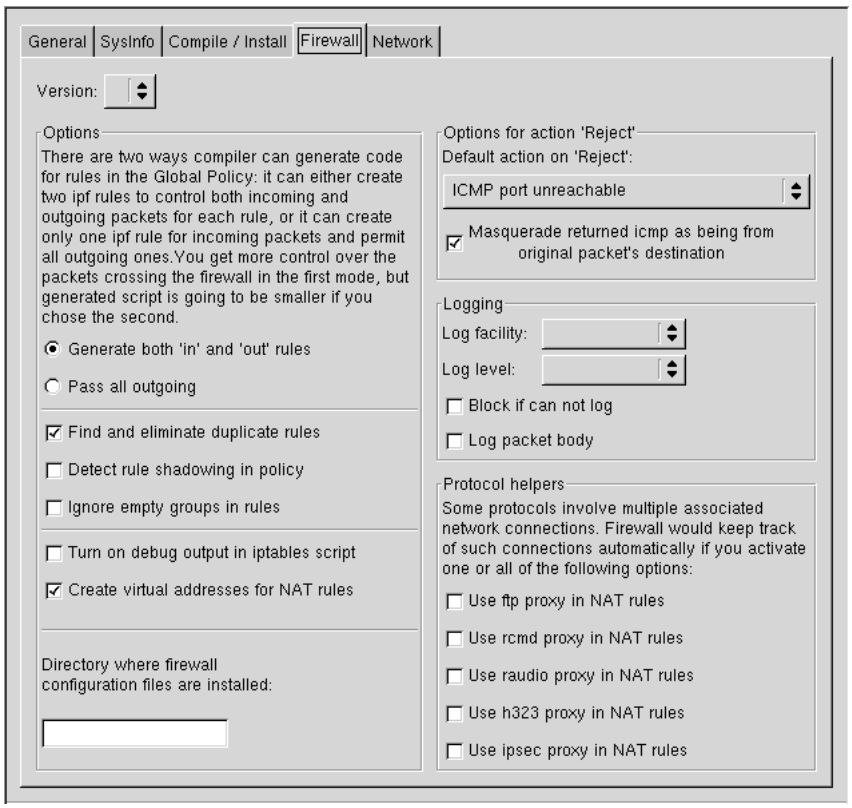



**Figure 5-24. Firewall Object, tab 'Firewall' for ipfilter**

**Table 5-4. General**

| GUI control | Description |
|---|---|
| **Version** | OS version the firewall is running on |

**Table 5-5. Options**

| GUI control | Description |
|---|---|
| **Generate both 'in' and 'out' rules** | There are two ways compiler can generate code for rules in the Global Policy: it can either create two pf rules for one rule in the GUI to control both incoming and outgoing packets, or it can create only one pf rule for incoming packets and always permit all outgoing packets. If this option is ON, the compiler creates rules for incoming and outgoing packets. This way you get more control over the data flows crossing your firewall, but the configuration is going to be larger. This option is common for compilers for *ipfilter* and *pf* |

| GUI control | Description |
|---|---|
| **Pass all outgoing** | See above. If this option is ON, the compiler puts an implicit rule to permit all outgoing packets on the top of the policy and then creates rules only for incoming packets. This way you get a smaller configuration but have little control over outgoing sessions. This option is common for compilers for *ipfilter* and *pf* |
| **Find and eliminate duplicate rules** | See Section 5.10.1.2 |
| **Detect rule shadowing in policy** | See Section 5.10.1.1 |
| **Ignore empty groups in rules** | See Section 5.10.1.3 |
| **Turn on debug output in the script** | See Section 5.10.1.10 |
| **Create virtual addresses for NAT rules** | See Section 5.10.1.8 |
| **Directory where firewall configuration files should be installed** | This parameter tells the firewall initialization script produced by the compiler where the firewall configuration file is going to be located. |

**Table 5-6. Options for action 'Reject'**

| GUI control | Description |
|---|---|
| **Default action on 'Reject'** | See Section 5.10.1.9 |
| **Masquerade returned icmp as being from original packet's destination** | The firewall will construct an ICMP 'unreachable' packet using the source IP of the original packet's destination. This helps to hide the firewall which sits in between the sender and their original destination. If this option is OFF, the icmp 'unreachable' packet has the source address of the firewall. |

**Table 5-7. Logging**

| GUI control | Description |
|---|---|
| **Log facility** | See Section 5.10.1.6 |
| **Log level** | See Section 5.10.1.5 |
| **Block if can not log** | If for some reason the firewall is unable to log the packet (such as the log reader being too slow), then the rule should be interpreted as if the action was *Deny* for this packet. |
| **Log packet body** | Logs not only the packet's header, but also its payload data. |

**Protocol Helpers:** IP Filter comes with a few simple proxies built into the code that is loaded into the kernel to allow secondary channels to be opened without forcing the packets through a user program. See documentation for ipfilter and FAQ for additional information.

### 5.10.4. pf

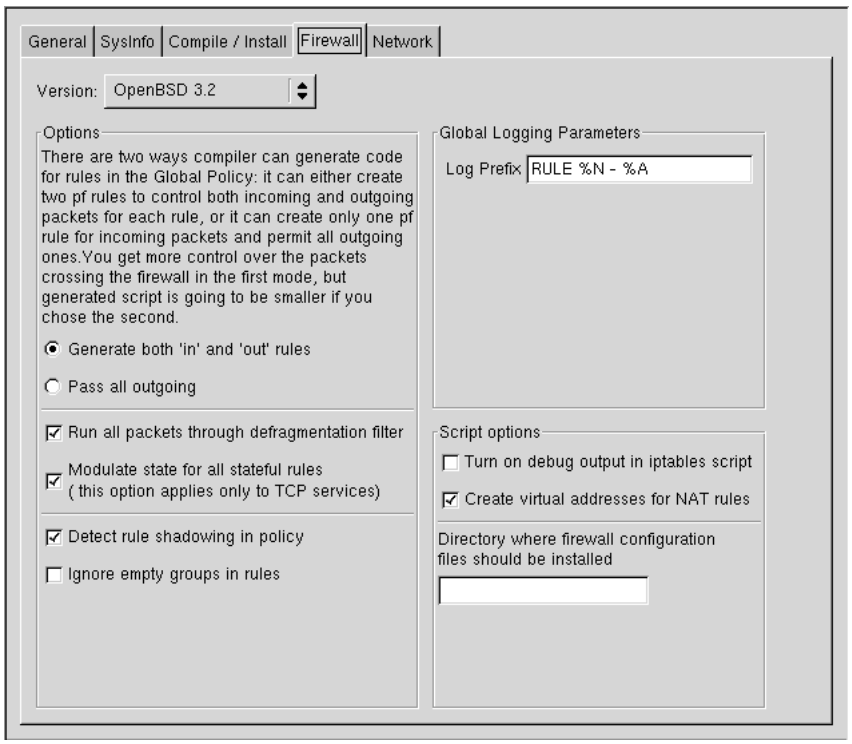



**Figure 5-25. Firewall Object, tab 'Firewall' for pf**

**Table 5-8. General**

| GUI control | Description |
|---|---|
| **Version** | OS version the firewall is running on |

**Table 5-9. Options**

| GUI control | Description |
|---|---|
| **Generate both 'in' and 'out' rules** | There are two ways the compiler can generate code for rules in the Global Policy: it can either create two pf rules for one rule in the GUI to control both incoming and outgoing packets, or it can create only one pf rule for incoming packets and always permit all outgoing packets. If this option is ON, the compiler creates rules for incoming and outgoing packets. This way you get more control over the data flows crossing your firewall, but the configuration is going to be larger. This option is common for compilers for *ipfilter* and *pf* |

| GUI control | Description |
|---|---|
| **Pass all outgoing** | See above. If this option is ON, the compiler puts an implicit rule to permit all outgoing packets on the top of the policy and then creates rules only for incoming packets. This way you get a smaller configuration but have little control over outgoing sessions. This option is common for compilers for *ipfilter* and *pf* |
| **Run all packets through defragmentation filter** | This is also called packet normalization. Packet normalization is invoked via the scrub directive. Normalization is used to sanitize packet content in such a way that there are no ambiguities in packet interpretation on the receiving side. The normalizer does IP fragment reassembly to prevent attacks that confuse intrusion detection systems by sending overlapping IP fragments. See more details in the man page `pf.conf(5)` |
| **Modulate state for all stateful rules** | By applying a "modulate state" rule to a TCP connection, pf will create a high quality random sequence number for each connection endpoint. The "modulate state" directive implicitly keeps state on the rule and is only applicable to TCP connections. See more details in the manual page `pf.conf(5)` |
| **Detect rule shadowing in policy** | See Section 5.10.1.1 |
| **Ignore empty groups in rules** | See Section 5.10.1.3 |

**Table 5-10. Global Logging Parameters**

| GUI control | Description |
|---|---|
| **Log Prefix** | See Section 5.10.1.7 |

**Table 5-11. Script Options**

| GUI control | Description |
|---|---|
| **Turn on debug output in the script** | See Section 5.10.1.10 |
| **Create virtual addresses for NAT rules** | See Section 5.10.1.8 |
| **Directory where firewall configuration files should be installed** | This parameter tells the firewall initialization script produced by the compiler where the firewall configuration file is going to be located. |

### 5.10.5. PIX

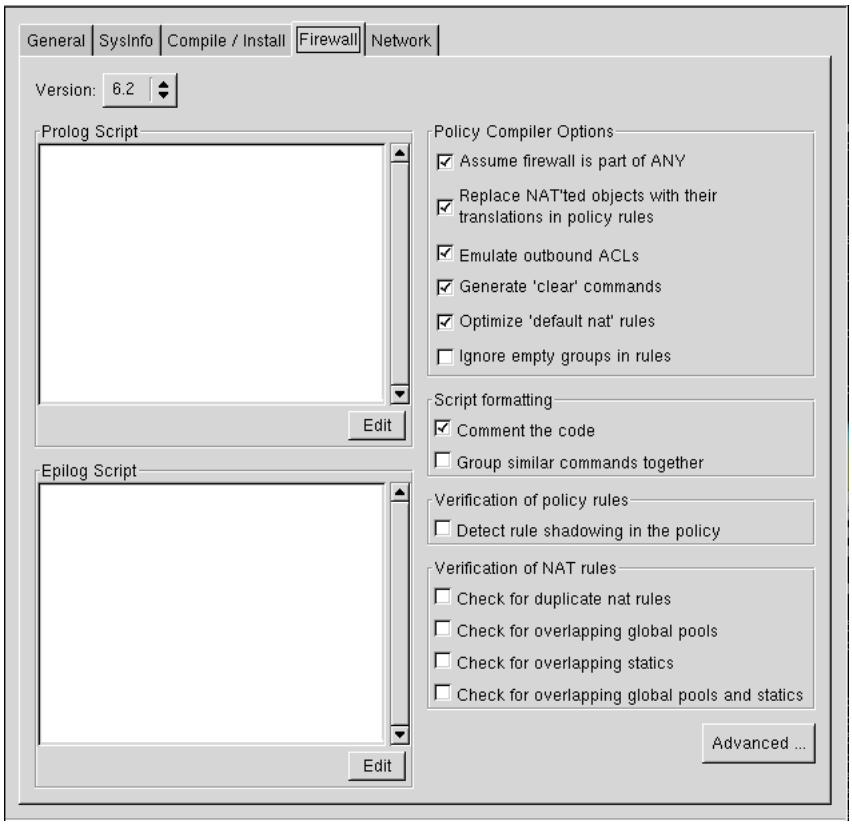



**Figure 5-26. Firewall Object, tab 'Firewall' for PIX**

**Table 5-12. General**

| GUI control | Description |
|---|---|
| **Version** | Version of the software PIX firewall is running. |
| **Prolog script** | This is a list of any PIX configuration commands that will be included on top of the generated configuration file. No syntax or other checks are done on commands in this list. |
| **Epilog script** | this is a list of any PIX configuration commands that will be appended at the end of the generated configuration file. No syntax or other checks are done on commands in this list |

**Table 5-13. Policy compiler options**

| GUI control | Description |
|---|---|
| **Assume firewall is part of 'any'** | See Section 5.10.1.4 |

| GUI control | Description |
|---|---|
| **Replace NAT'ed objects with their translations in policy rules** | PIX inspects a packet with access lists before it performs an address translation. Many other firewall platforms do it the other way around. This option turns on emulation to make PIX look like a firewall that does NAT first. |
| **Emulate outbound ACLs** | Normally PIX does not support outbound access lists.This option turns on an emulation of outbound ACLs |
| **Generate 'clear' commands** | If this option is ON, the compiler generates *'clear'* commands to reset any pre-existing access lists, object-group, nat, global, static, telnet, ssh and other commands. |
| **Optimize 'default nat' rules** | This option simplifies nat rules if the object in Original Source is the same as the Network zone of one of the interfaces. The network zone of the interfaces defines all networks that are located "behind" this interface. This means that packets entering the interface may have a source address only belonging to the Network zone of this interface. Since the policy compiler can correctly assign a NAT rule to the interface using information about its Network Zone, explicit specification of the source address can be omitted. |
| **Ignore empty groups in rules** | See Section 5.10.1.3 |

**Table 5-14. Script formatting**

| GUI control | Description |
|---|---|
| **Comment the code** | If this option is activated, the compiler adds comments to the configuration file |
| **Group similar commands together** | If this option is activated, the compiler groups similar commands next to each other, just like the PIX device does it in the output of the "show config" command. Otherwise commands are grouped logically: first go all *object-group* commands, then all *access-lists*, then all *nat*,*global* and *static* commands. Commands *access-list*, *nat*,*global* and *static* are grouped by the rules they were generated for, as they appear in the GUI. If one rule requires several *access-list* commands assigned to different interfaces, these commands are grouped together. |

**Table 5-15. Verification of policy rules**

| GUI control | Description |
|---|---|
| **Detect rule shadowing in the policy** | See Section 5.10.1.1 |

**Table 5-16. Verification of NAT rules**

| GUI control | Description |
|---|---|

| GUI control | Description |
|---|---|
| **Check for duplicate nat rules** | If this option is activated, the compiler checks the generated configuration for duplicate *'nat'* commands |
| **Check for overlapping global pools** | If this option is activated, the compiler checks the generated configuration for overlapping *'global'* address pools |
| **Check for overlapping statics** | If this option is activated, the compiler checks the generated configuration for *'static'* commands that use overlapping address ranges. |
| **Check for overlapping global pools and statics** | If this option is activated, the compiler checks the generated configuration for *'global'* and 'static commands using overlapping address ranges. |

#### 5.10.5.1. 'Advanced' firewall setting dialog for PIX

Clicking button **'Advanced'** in the 'Firewall' tab of the PIX firewall object brings up a new dialog. This dialog helps configure protocol timeouts, *'fixup'* commands, logging and some other general configuration option.

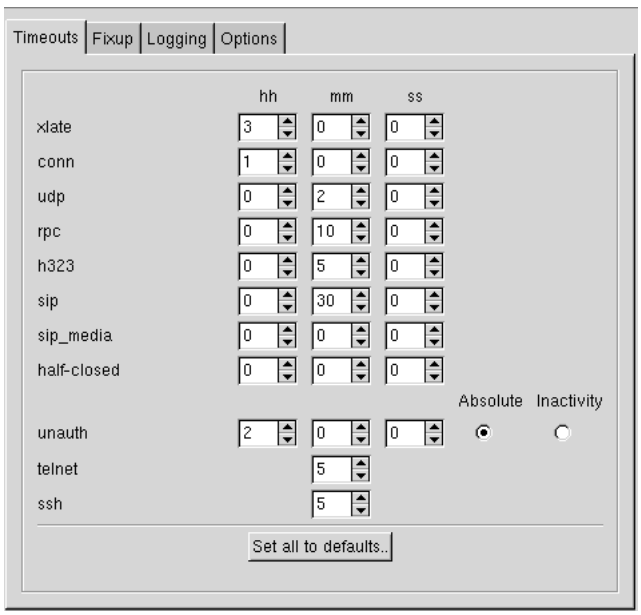



**Figure 5-27. Configuring protocol timeouts.**

Parameters set in this dialog are mapped directly to the *'timeout'* command in the PIX configuration which sets timeout values for various protocols supported by PIX. Clicking **'Set all to defaults'** button resets all timeout parameters to their default values.

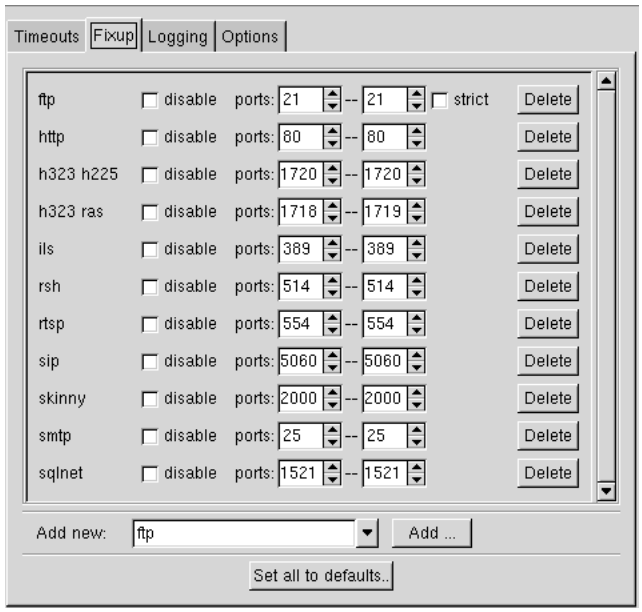



**Figure 5-28. Configuring protocol helpers ('fixup' commands).**

Parameters set in this dialog are mapped to the *'fixup'* commands in the PIX configuration. The **'Add'** button adds a new parameter of the type chosen in the drop-down menu to the left. Clicking the *'Delete'* button deletes the corresponding parameter. Clicking the *'Set all to defaults'* button returns all settings to their default values.

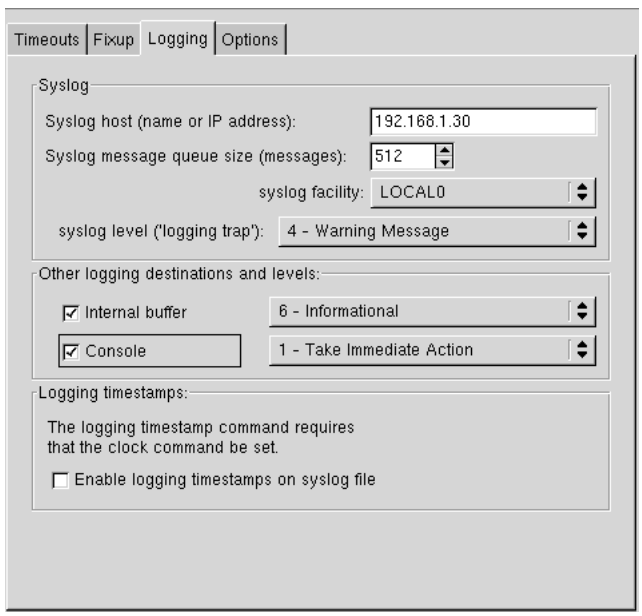



**Figure 5-29. Configuring logging parameters.**

This dialog helps configure logging parameters for the PIX firewall:

**Table 5-17. Syslog**

| GUI control | Description |
|---|---|
| **Syslog host** | This is an address of the syslog host the firewall will send all logging to. |
| **syslog message queue size** | The size of the internal logging queue, in messages. |
| **Facility** | Syslog facility, See Section 5.10.1.6 |
| **Level** | Syslog level, See Section 5.10.1.5 |

**Table 5-18. Other logging destinations, levels and parameters**

| GUI control | Description |
|---|---|
| **Internal buffer** | Send syslog messages to an internal buffer that can be viewed with the show logging command. The level pull-down menu allows to set a minimal logging level of messages that are going to be recorded. |
| **Console** | Specify that syslog messages appear on the PIX Firewall console as each message occurs. You can limit the types of messages that appear on the console using the level parameter. |
| Logging timestamps | Specify that syslog messages sent to the syslog server should have a time stamp value on each message. |

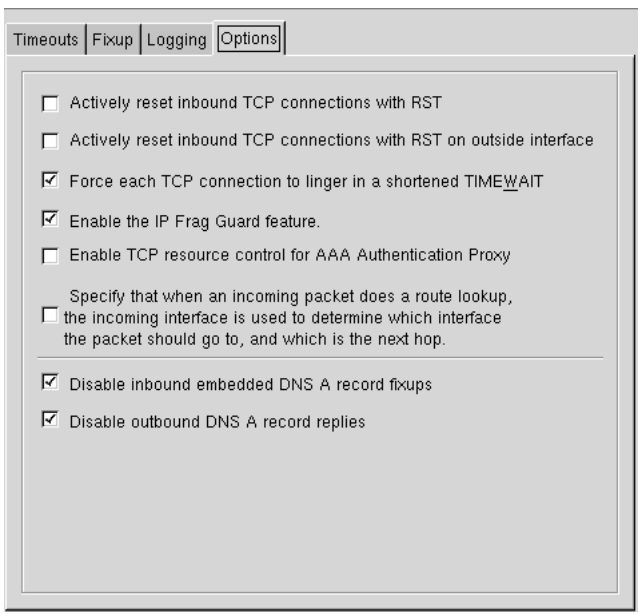



**Figure 5-30. Configuring other parameters.**

This dialog helps to configure some global parameters for the PIX firewall. See PIX documentation for a detailed explanation of each command.

**Table 5-19. Global parameters controlled through the 'Advanced' firewall dialog**

| GUI control | PIX command |
|---|---|
| Actively reset inbound TCP connections with RST | *service resetinbound* |
| Actively reset inbound TCP connections with RST on outside interface | *service resetoutside* |
| Force each TCP connection to linger in a shortened TIME_WAIT | *sysopt connection timewait* |
| Enable the IP Frag Guard feature | *sysopt security fragguard* |
| Enable TCP resource control for AAA Authentication proxy | *floodguard enable* |
| Specify that when an incoming packet does a route lookup, the incoming interface is used to determine which interface the packet should go to, and which is the next hop. | *sysopt route dnat* |
| Disable inbound embedded DNS A record fixups | *sysopt nodnsalias inbound* |
| Disable outbound DNS A record replies | *sysopt nodnsalias outbound* |

## 5.11. Configuring Host OS

Firewall object's 'Network' dialog tab provides a set of controls to manipulate network settings and some kernel parameters for the host OS that the firewall machine is running on.

### 5.11.1. FreeBSD

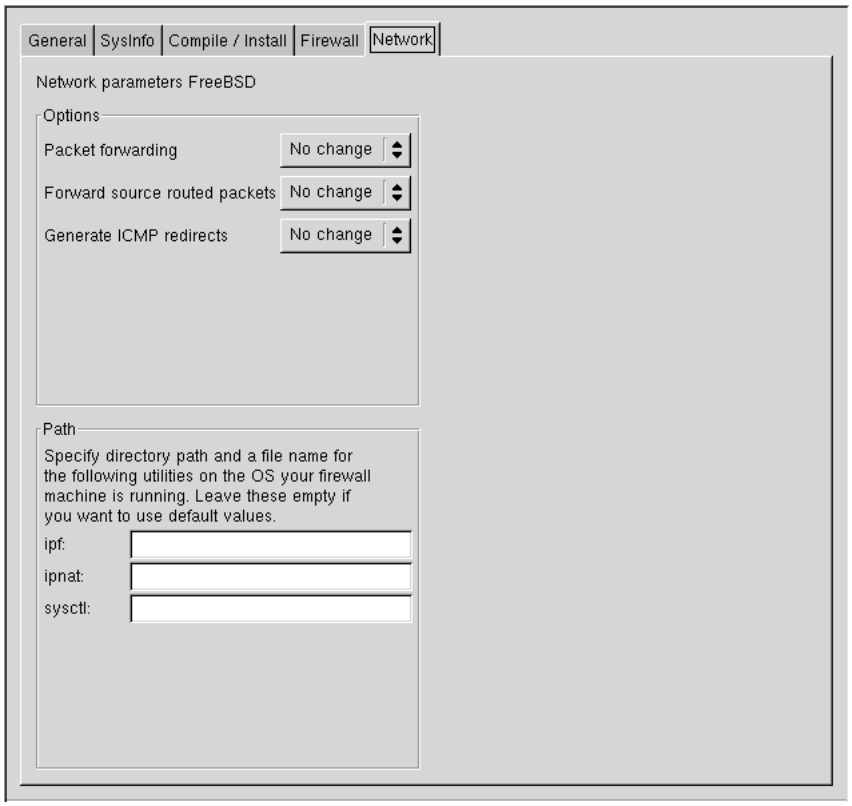



**Figure 5-31. Firewall Object, 'Network' tab for FreeBSD**

### 5.11.2. Linux

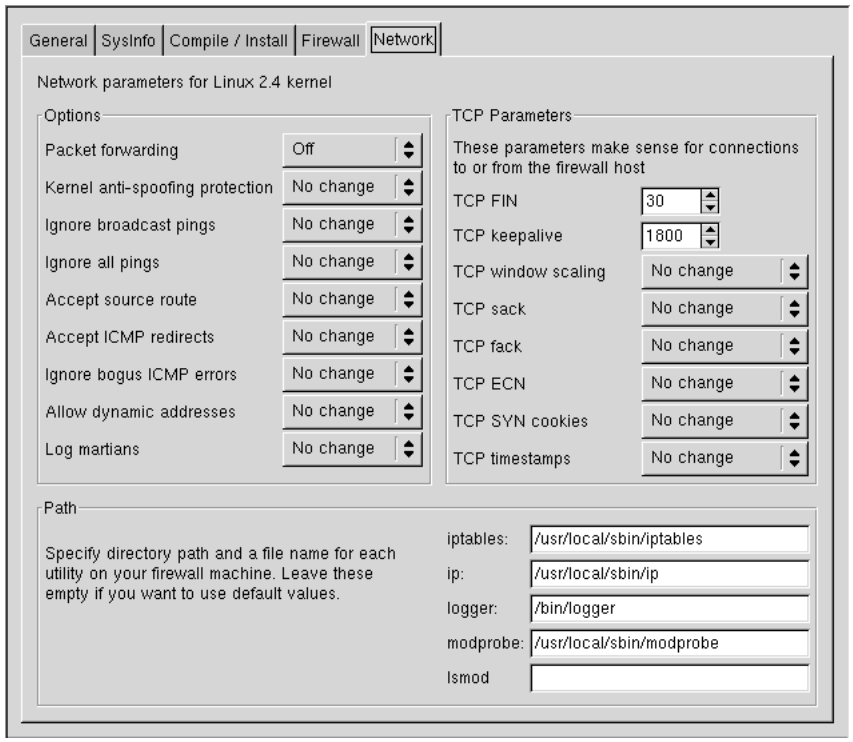



**Figure 5-32. Firewall Object, 'Network' tab for Linux**

### 5.11.3. OpenBSD

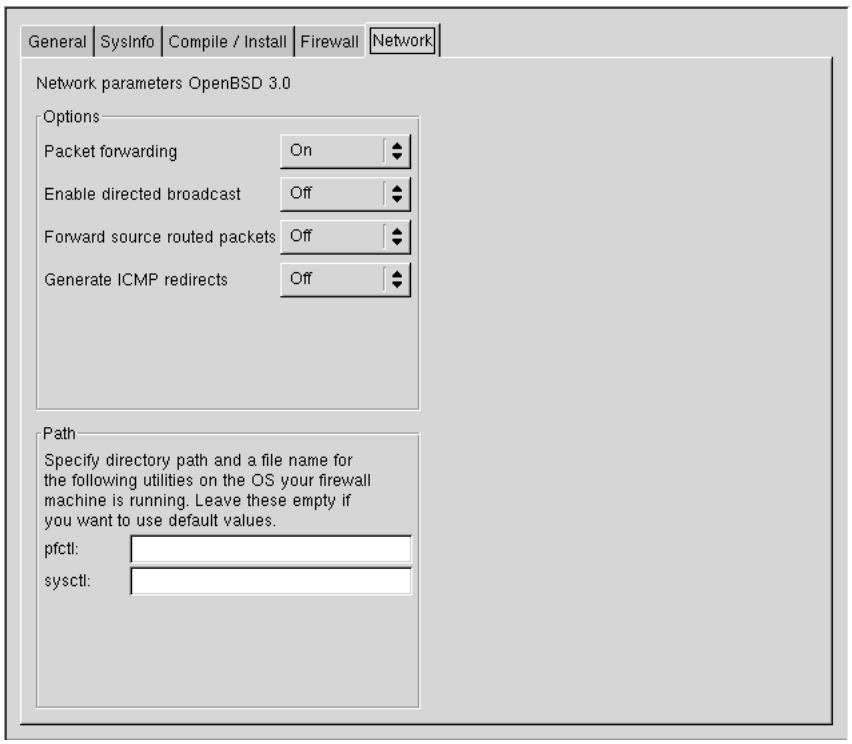



**Figure 5-33. Firewall Object, 'Network' tab for OpenBSD**

### 5.11.4. Solaris

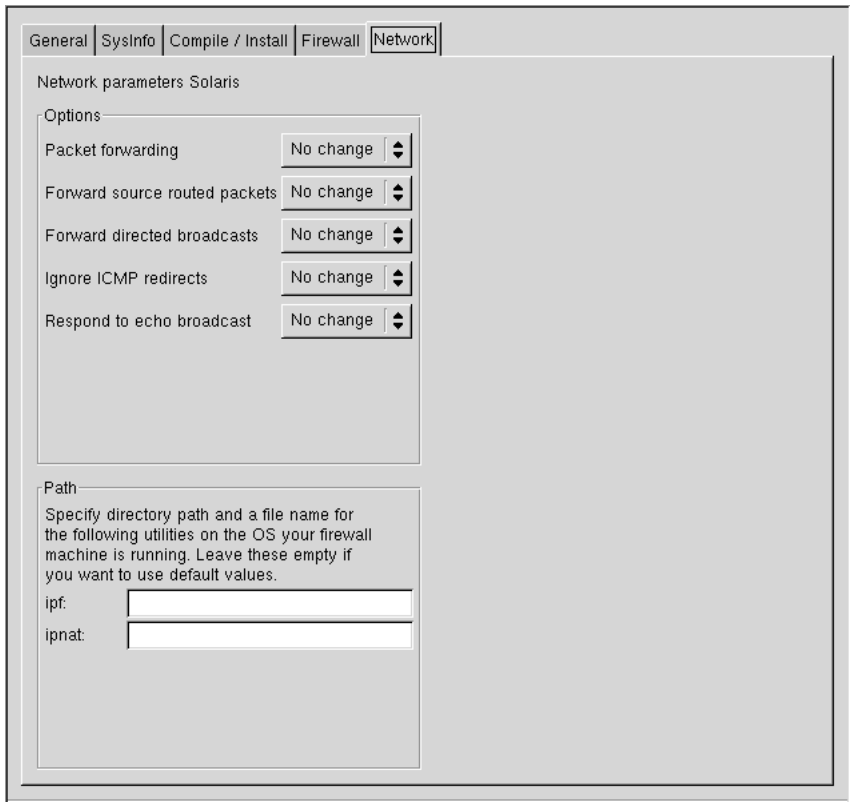



**Figure 5-34. Firewall Object, 'Network' tab for Solaris**

### 5.11.5. PIX

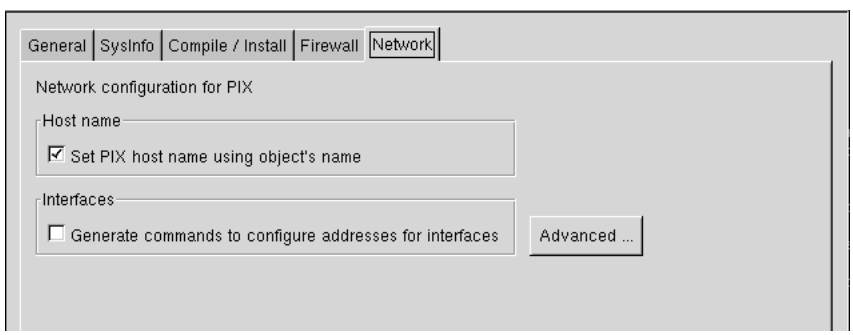



**Figure 5-35. Firewall Object, 'Network' tab for PIX**

## 5.12. The Firewall's Interface

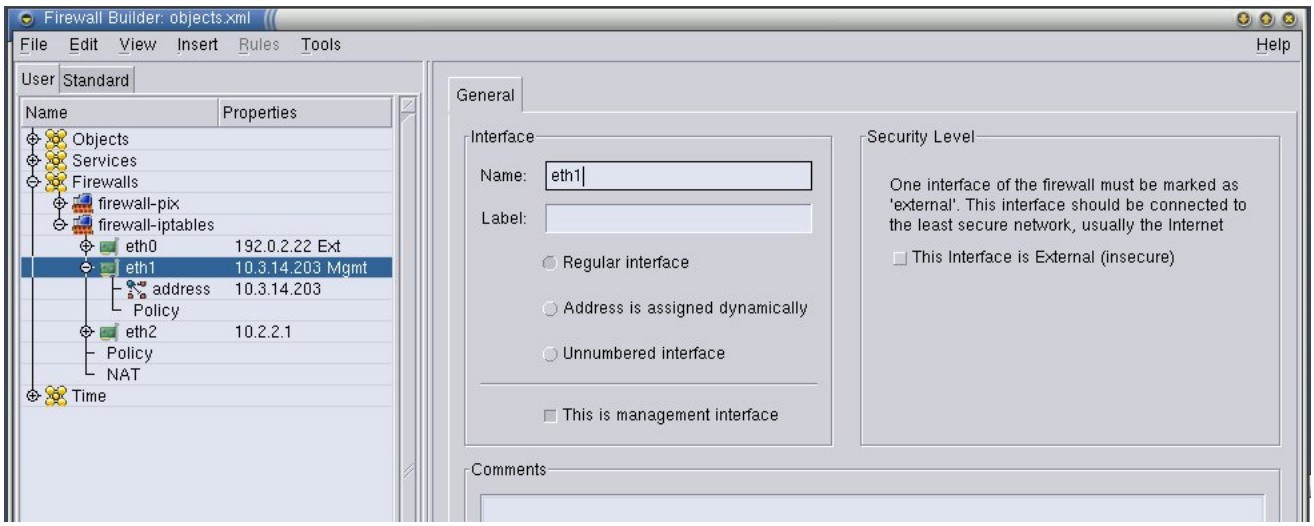



**Figure 5-36. The Interface Object belonging to a firewall**

The dialogue for the interface object that belongs to a firewall is similar to that for the Host's interface (Section 5.3) but has a few extra controls.

The interface of the Firewall and Host objects has the same types described in Section 5.3.1

Unlike interface of the Host object, interface of the Firewall has a parameter *"Security Level"*. Depending on the firewall platform, the security level may either have two values *"External"/"Internal"* or a numeric value between 0 and 100, with 0 being least secure and 100 being most secure levels. The GUI dialogue adjusts automatically and shows appropriate controls depending on the choice of the firewall platform in the interface's parent firewall object.

Figure 5-36 shows a screenshot of the dialogue for the interface object that belongs to an iptables firewall object named "firewall-iptables". Figure 5-37 shows a screenshot of the dialogue for the interface object that belongs to a Cisco PIX firewall object named "firewall-pix".

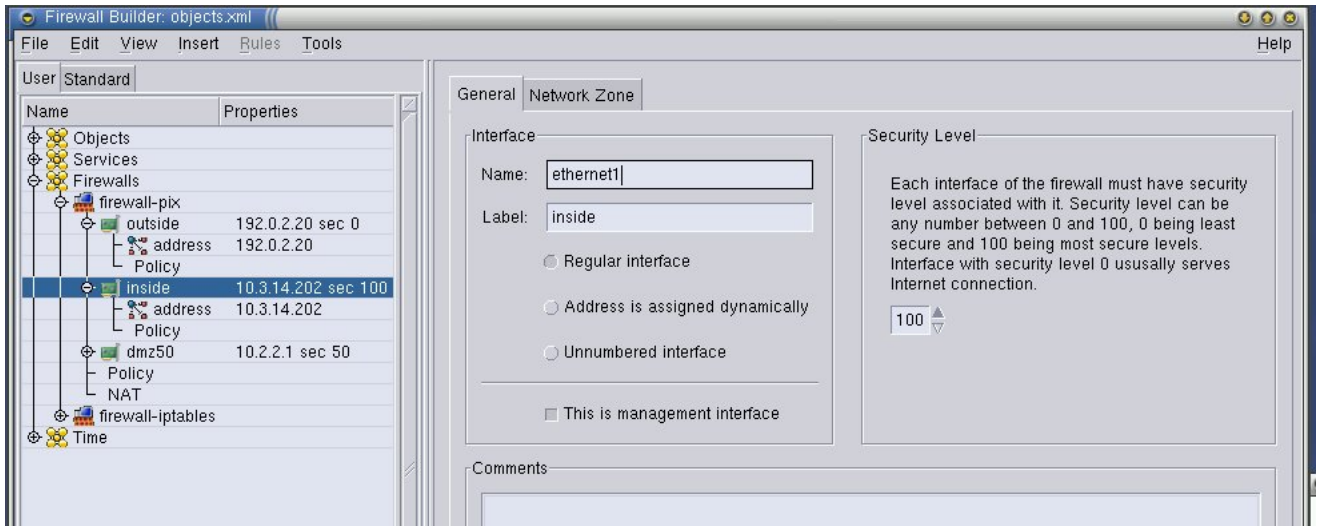



**Figure 5-37. An Interface with a numeric Security Level**

### 5.12.1. The Network Zone

One of the platform-specific parameters of the interface is a reference to the Network Zone. Currently only a Cisco PIX firewall requires the Network Zone. The interface dialogue shows controls for the Network Zone in a separate tab (Figure 5-38), which appears only if the parent firewall object specifies the firewall platform which requires it. The GUI dynamically generates a pull-down menu with all available network and group objects and puts it in the Network Zone dialog. You need to choose one object in the menu. Usually the external interface (the one which connects your firewall to the Internet) has the Network Zone set to *"Any"*. It is also recommended to create a group of objects to represent Network Zones of all other interfaces of the firewall.

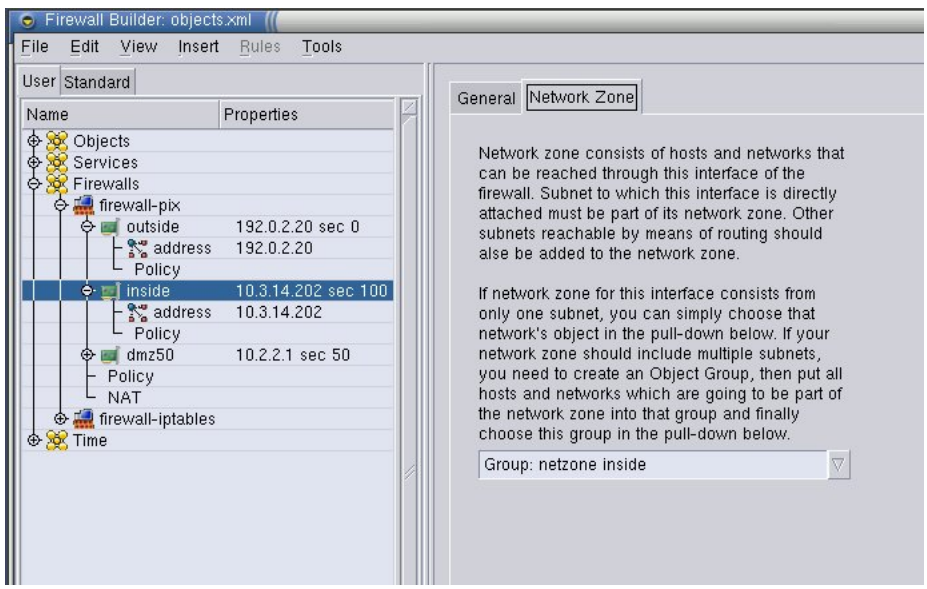



**Figure 5-38. The Network Zone dialogue**

Consider the network layout as in Figure 5-39. Here the firewall has three interfaces: 'outside', 'dmz', and 'inside'. Behind the firewall there is a router which in turn is connected to three subnets 'subnet A', 'subnet B', and 'subnet C'. Subnet A is shared between the router and the firewall (each device has an interface on this subnet). Let's also suppose that we have created Network Objects for each subnet and called them 'subnet DMZ', 'subnet A', 'subnet B' and 'subnet C' (remember, spaces are allowed in object names). For this setup, network zones should be configured as follows:

| Interface | Network Zone |
|---|---|
| `outside` | *ANY* |
| `dmz` | *subnet DMZ* |
| `inside` | *subnet A*, *subnet B*, *subnet C* |

Since the network zone for the *'inside'* interface consists of multiple objects, a group must be created so that you could use this group as a Network Zone object.

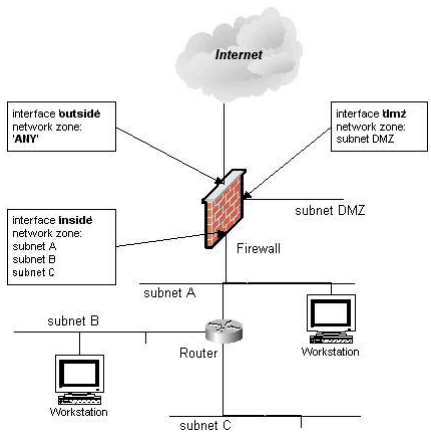



**Figure 5-39. Choosing Network Zones**

Table 5-20 explains the differences in the way firewall platforms interpret values in the Security Level and Network Zone parameters of the firewall interfaces.

**Table 5-20.**

| Firewall Platform | Security Level Values | Network Zone |
|---|---|---|
| iptables | two values: 'External' or 'Internal' | N/A |
| ipfilter | two values: 'External' or 'Internal' | N/A |
| pf | two values: 'External' or 'Internal' | N/A |
| Cisco PIX | numeric, 0 - 100 | a reference to a group or network object |

# Chapter 6. Network Discovery: A Quick Way to Create Objects




**Figure 6-1. Calling the Object Discovery Druid**

One of the distinguishing features that Firewall Builder provides is support for automated object creation. This helps populate the objects tree for large networks with lots of hosts and subnets. It would have taken hours if the user always had to do it manually, while the Objects Discovery Druid helps complete this task in minutes. You can bring this Druid up using the **"Tools"/"Discover Objects"** main menu

The Druid supports three main methods for automated object creation:

- Reading file /etc/hosts
- Import of DNS zone
- Network discovery using SNMP queries

You choose the method on the first page of the Druid ( Figure 6-2. )

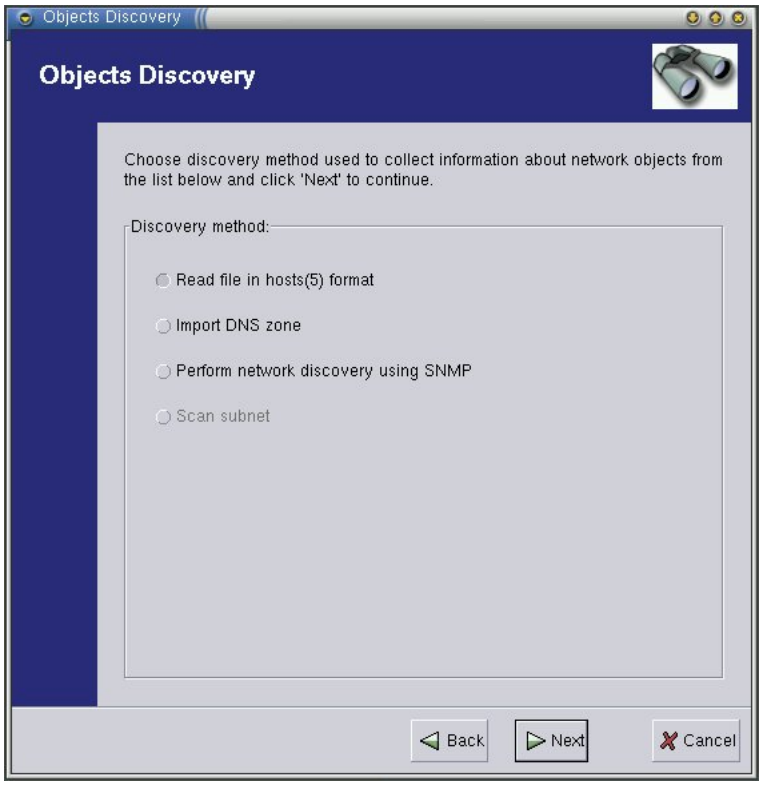



**Figure 6-2. Calling The Object Discovery Druid**

Just check the radio-button next to the method you want to use and click the **'Next'** button.

## 6.1. Reading the /etc/hosts file

This method imports the host records present in the standard */etc/hosts* file or any other file that contain records in the following format (this format is actually described in the manual page hosts(5)).

IP_Address host_name

The IP address must be separated from the host name with any number of spaces or tab symbols. Lines starting with '#' are considered comments and are ignored. Currently Firewall Builder does not support ipv6 addresses and ignores them.

When you choose the import from /etc/hosts on the first page, the Druid asks you for the file path and name on the next page. Once that information is entered, it reads the contents of that file and presents a table of new objects (Figure 6-3):

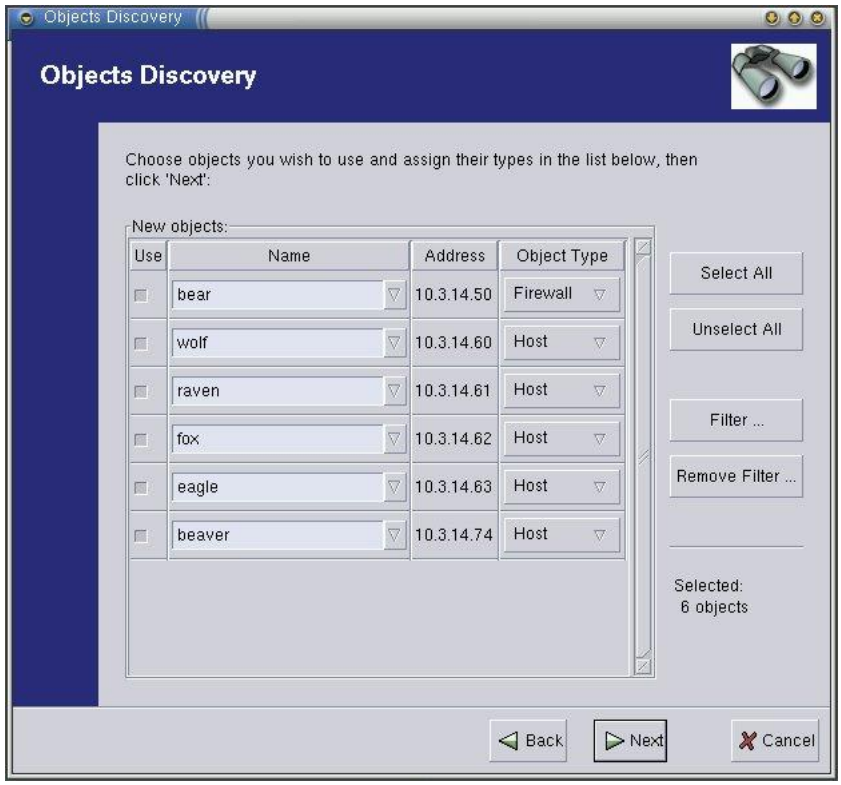



**Figure 6-3. Creating objects using gathered information**

This page of the druid is the same for all discovery methods, so let's discuss it in detail.

Discovered objects are shown in the table. Since the source of information used for discovery could have had different host names for the same IP address, object names are shown in a pull-down menu. If there was only one name for the address, then the pull-down menu would have only one entry, otherwise it has several entries. The rightmost column in the table shows choices for the object type to be created: it can become either Host, Network, or Firewall. Sometimes, the Druid can guess the type of the object or use the proper type depending on the discovery method used.

The Druid creates objects only for selected records in the table. Use the checkbox in the very first column of the table to select objects. The comment to the right shows the total number of selected objects at any given moment (text "Selected: N objects"). 6 objects are selected in the screenshot (Figure 6-3)

This page of the Druid also has the following buttons:

- **"Select All"**

  Selects all records in the table

- **"Unselect All"**

  Deselects all records in the table

- **"Filter"**

  Brings up a filter dialog. Filtering helps manage long lists of objects.

- **"Remove Filter"**

  Removes the currently applied filter and shows all records in the table




**Figure 6-4. Filter Dialog**

The Druid can filter records in the table either by their address, by their name, or by both. To filter by address enter part of it in the "Address" field. The program compares the text entered in the filter dialog with an address in the table and shows only those records whose address starts with the text of the filter. For example, to only filter out hosts with addresses on the net 10.3.14.0 we could use the filter "10.3.14". Likewise, to remove hosts "bear" and "beaver" (addresses 10.3.14.50 and 10.3.14.74) we could use the filter "10.3.14.6". Note that the filter string does not contain any wildcard symbols like "*". The filter shows only records that have addresses which literarily match the filter string.

Filtering by the object name uses the POSIX regular expressions syntax described in the manual page regex(7). For example, to find all records whose names start with 'f' we could use the regular expression '^f'. The '^' symbol matches the beginning of the string, so this regular expression matches any name that starts with 'f'. To find all names that end with 'somedomain.com', we could use the regular expression '.*somedomain.com$'

## 6.2. DNS zone import

This method works only on those OS where there is a library supporting DNS zone transfers. Currently it is available on Linux and Solaris.

First, the Druid asks you for the domain name you wish to import (Figure 6-5)




**Figure 6-5. Entering the domain name**

Once the domain name is entered and you have clicked **"Next"**, the program first runs the DNS query to find out what name servers are responsible for that domain. It then presents a list of name servers on the next page for you to choose from (Figure 6-6).




**Figure 6-6. List of name servers**

You can use one of the registsred name servers, or, if you have your own local name server which is authoritative for this domain (for example the local secondary server), you can enter its name or address in the entry field at the bottom of the dialog. Clicking **"Next"** initiates the zone transfer process (Figure 6-7).

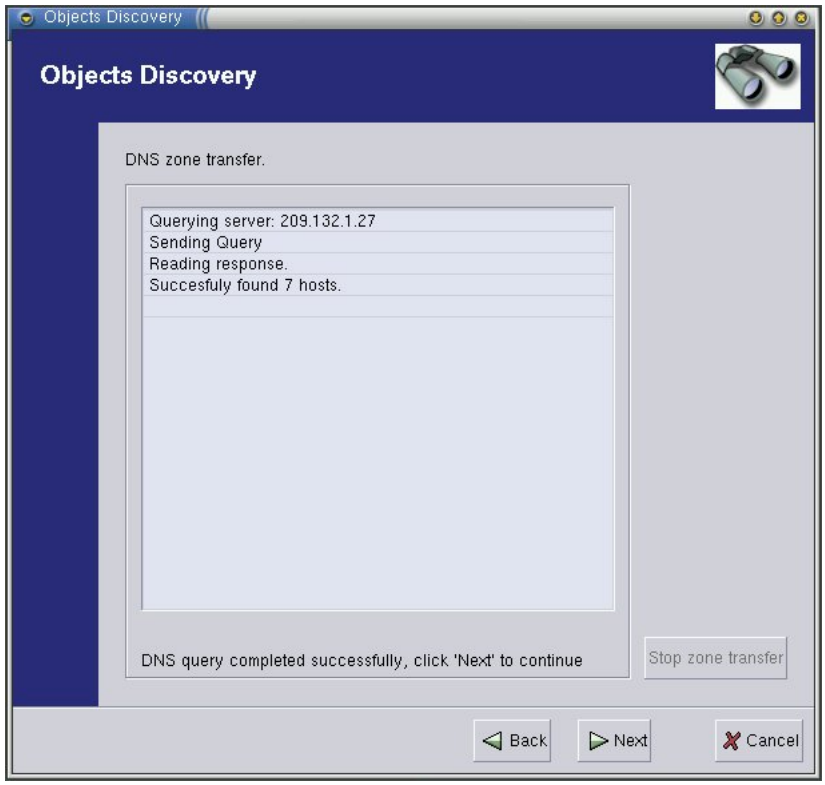



**Figure 6-7. DNS zone transfer**

If the DNS zone transfer succeeded and the druid was able to pull the information it needs to create objects, you can switch to the next page where you choose and create objects. This page is the same as the one presented in Figure 6-3.

## 6.3. Network Discovery




**Figure 6-8. Initial Parameters for the Network Discovery program**

The Network discovery program (sometimes referred to as the "Network Crawler") needs a host to start from. This host is called the "seed host"; you enter it in the first page of the Druid (Figure 6-8). The crawler implements the following algorithm (this is a somewhat simplified explanation):

First, it runs several SNMP queries against the seed host trying to collect the following data: the list of its interfaces and its ARP and routing tables. This host is then added to the table of discovered network objects, together with its interfaces, their addresses and netmasks and the host's "sysinfo" parameters. Then the crawler analyses the routing table of that host; this allows it to discover the networks and subnets which in turn are also added to the list of discovered objects. Then it analyses the ARP table, which holds MAC and IP addresses of neighboring hosts. It takes one host found in this table at a time and repeats the same algorithm using it as a seed host. When it pulls an ARP table from the next host, it discards entries that describe objects it already knows about. However if it finds new entries, it tries them as well and thus travels further down the network. Eventually it will visit every host on all subnets on the network.

This algorithm relies on hosts answering to the SNMP queries. If the very first host (the "seed" host) does not run SNMP agent, the crawler will stop on the first run of its algorithm and won't find anything. Therefore it is important to use a host which does run SNMP agent as a "seed" host. Even if most of the hosts on the network do not run SNMP agents, but a few do, the crawler will most likely find all of them. This happens because it discovers objects when it reads the ARP tables from the host which answers; so even if discovered hosts do not answer to SNMP queries, the crawler already found them anyway.

One of the ways to limit the scope of the network that the crawler will visit is to use the "Confine scan to the network" parameter. You need to enter both a network address and a netmask; the crawler will then check if the hosts it discovers belong to this network and if they do not, discard them.

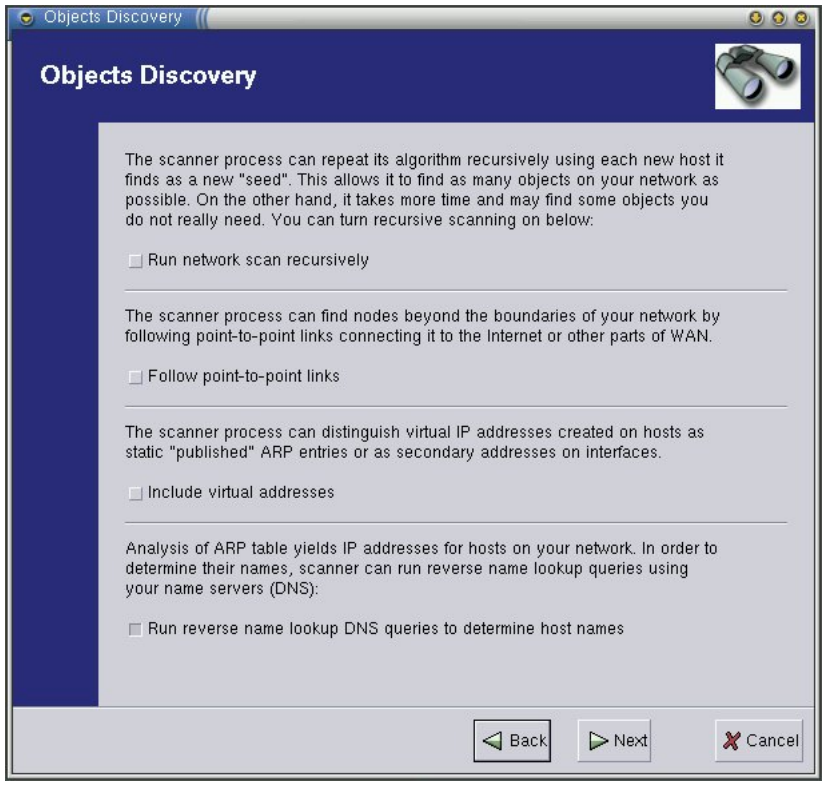



**Figure 6-9. Parameters for Network Discovery: Page 1**

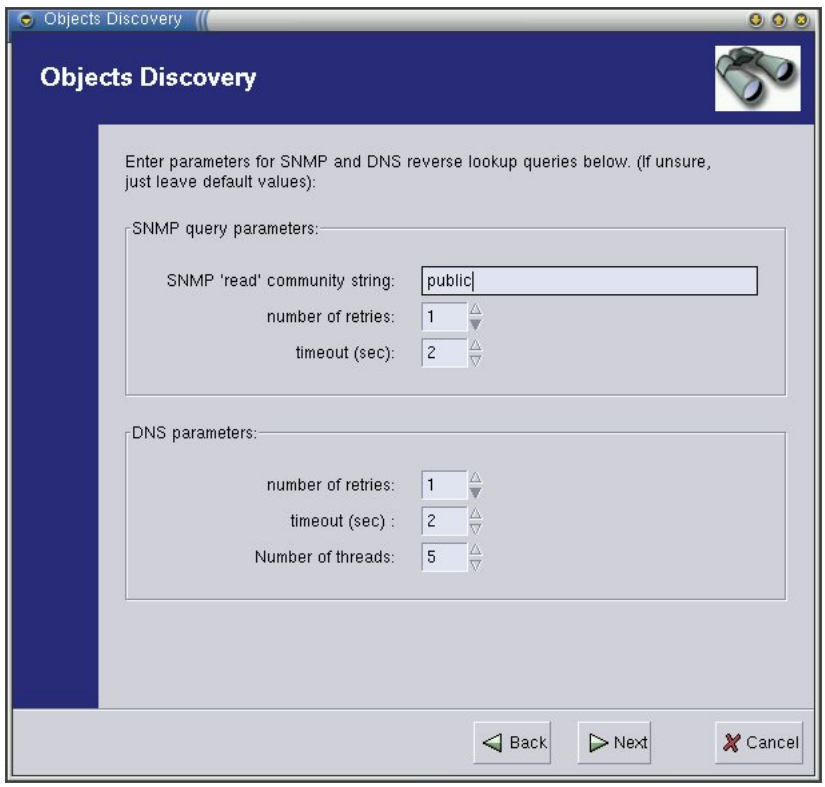



**Figure 6-10. Parameters for Network Discovery: Page 2**

There are a few settings that affect the crawler's algorithm (see Figure 6-9 and Figure 6-10). Here is the list:

- Run network scan recursively

  As was described above, the crawler starts with the "seed" host and then repeats its algorithm using every discovered host as a new "seed". If this option is turned OFF, then the crawler runs its algorithm only once and stops.

- Follow point-to-point links

  If a firewall or a router has a point-to-point interface (for example, PPP interface), then the crawler can automatically calculate the IP address of the other side of this interface. It then continues the discovery process by querying a router on the other side. Very often, the point-to-point link connects the organization's network to an ISP and you are not really interested in collecting data about your ISP network. By default crawler won't cross point-to-point links, but this option, if activated, permits it.

- Include virtual addresses

  Sometimes servers or routers have more than one IP address assigned to the same interface. If this option is turned on, the crawler "discovers" these virtual addresses and tries to create objects for them.

- Run reverse name lookup queries to determine host names

  If the host discovered by the crawler answers to SNMP queries, it will report its name, which the crawler will use to create an object in Firewall Builder. However, if the host does not answer the query, the crawler can not determine its name and only knows its IP address. The crawler can use DNS to back-resolve such addresses and determine host names if this option is turned ON.

- SNMP query parameters

  You must specify the SNMP "read" community string which will be used for SNMP queries. You can also specify the number of retries and a timeout for the query.

- DNS parameters:

  You can specify the number of retries and a timeout for the DNS lookup query, as well as the number of threads. Firewall Builder uses a multithreaded resolver in order to speed up the process. In other words, it can run several DNS lookups simultaneously. The "Number of threads" parameter defines how many lookups it can run in parallel.

Once all parameters are entered, the crawler actually gets to work, which may take a while. Depending on the size of the network and such parameters as the SNMP timeout value, scanning may take minutes or even hours. The progress of the scanner can be monitored on the page in the Druid (Figure 6-11). You can always stop the crawler using the **"Stop network scan"** button. Data does not get lost if you do this as the Druid will use whatever objects the crawler discovered before you stopped it.

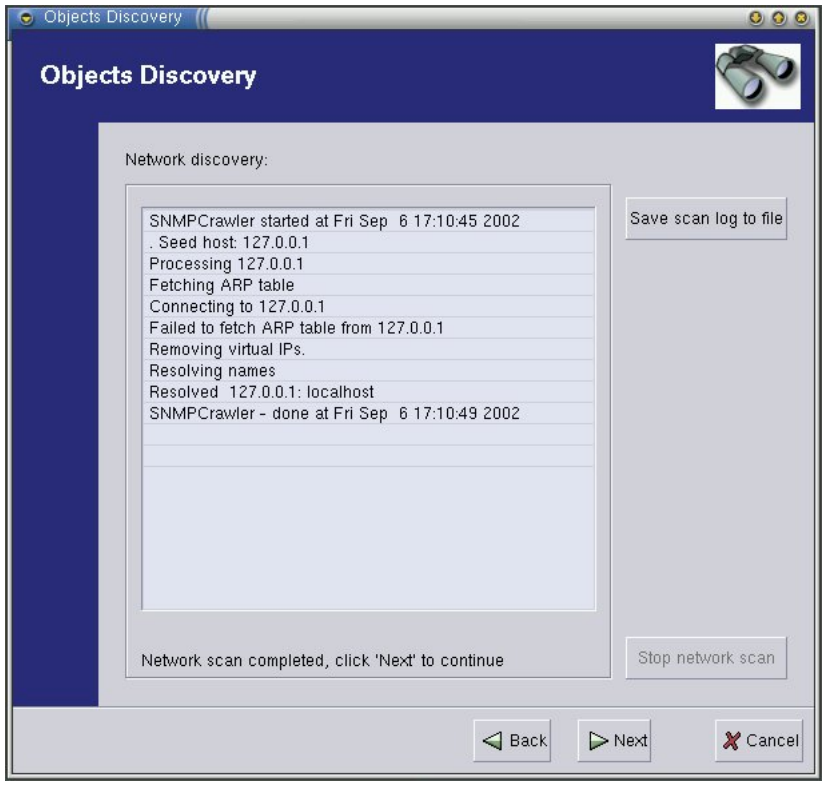



**Figure 6-11. Parameters for Network Discovery: Page 2**

The **"Save scan log to file"** button saves the content of the progress window to a text file and is mostly used for troubleshooting and bug reports related to the crawler.

If the crawler succeeded and was able to collect information it needed to create objects, you can switch to the next page where you choose and create objects. That page is the same as the one shown in Figure 6-3.

# Chapter 7. Working With Services

## 7.1. IP Service

The simple IP service object is represented in the screenshot (Figure 7-1). Firewall Builder provides standard object types for well-known and frequently used protocols such as ICMP (IP protocol number 1), TCP (IP protocol number 6), and UDP (IP protocol number 17). The IP service object helps describe other protocols which are neither ICMP nor TCP nor UDP. The protocol is defined by the 8-bit field in the IP packet header. The screenshot (Figure 7-1) represents the *'ESP'* object (Encapsulating Security Payload, part of the IPSEC protocol family) which uses the IP protocol number 50. Protocol numbers are assigned by IANA; one can look up the number for a particular protocol at the following URL: http://www.iana.org/assignments/protocol-numbers/

Besides the protocol number, the header of the IP packet also has a field called "options" which is a variable-length list of optional information for the packet. Not all firewalls can examine options, and those that can usually have certain limitations as to what options they can match against. Firewall Builder tries to provide controls for many popular options, supported by the most sophisticated firewalls. Not all options supported by Firewall Builder are supported by all target firewall platforms (Table 7-1).

Firewall Builder supports the following options:

`Source route options: LSRR, SSRR` — Normally IP routing is dynamic with each router making decisions about which next hop router to send the packet to. However another option exists, where the sender can choose the route. In the case of the Loose Source Route, the sender (host) can specify a list of routers the packet must traverse, but it may also pass through other routers between any two addresses in the list. The Strict Source Route works very much the same way, except the packet must traverse only through the specified addresses. Source routing can potentially be used to reach hosts behind the firewall even if these hosts use private IP addresses which normally are not reachable over the Internet.

`Record route option: RR` — This option causes every router that handles the packet on the way to add its IP address to a list in the options field. This option is used by the ping utility when it is called with the "-R" command line switch; it can potentially be exploited to discover the internal network addressing and layout behind the firewall. Although the risk is low, some firewall administrators prefer to block packets with this option set.

`Timestamp option:` — This option tells routers that handle the packet to record their timestamps and sometimes addresses (like in the case of the record route option). This option is seldom used, but can potentially be exploited to gather information about the protected network, so some firewall administrators prefer to block packets with this option set.

IP packets sometimes may become fragmented. This happens if the original datagram is larger than what a physical network layer can transmit. The IP packet header has special fields (called "Flags" and "Fragmentation Offset") which help detect fragmented packets and reassemble them. Many firewalls can check these bits as well. Certain combinations of flags and fragmentation offsets can never happen during a normal operation but were seen to be used by attackers. Firewall Builder provides two options for most commonly used cases related to packet fragmentation: the 'all fragments' option matches the second and further fragments, while the 'short' option is used to match packets that are too short to contain even a complete IP header.

Standard IP Service objects that come with Firewall Builder appear in the "Standard" tree, in the **Services/IP** branch.

**Table 7-1. Support for IP options and fragmentation on various firewall platforms**

| Firewall | lsrr | ssrr | rr | timestamp | all fragments | 'short' packets |
|---|---|---|---|---|---|---|

| Firewall | lsrr | ssrr | rr | timestamp | all fragments | 'short' packets |
|---|---|---|---|---|---|---|
| iptables | + | + | + | + | + | - |
| ipfilter | - | + | + | + | + | + |
| pf | - | - | - | - | + | - |
| Cisco PIX | - | - | - | - | - | - |

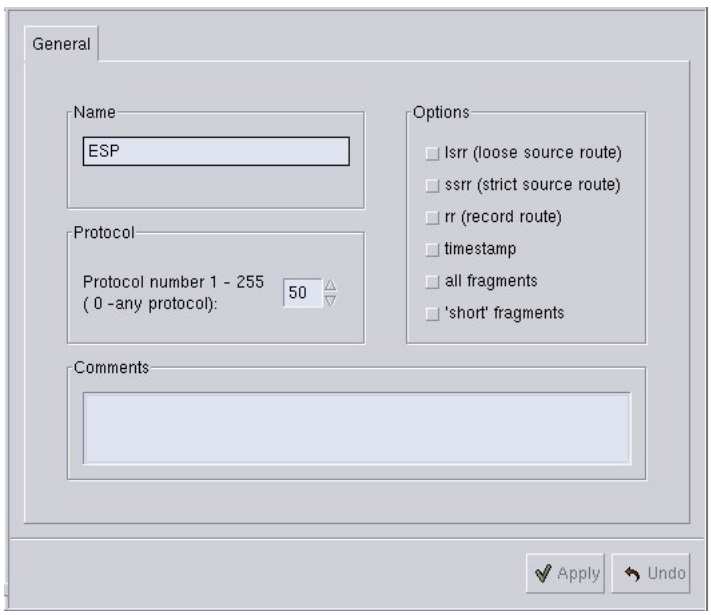



**Figure 7-1. Editing the IP Service Object**

The IP Service dialog provides the following controls:

- Name:
  This is the name of the object

- Protocol:
  This is the protocol number.

- Options:
  These flags represent 'Options' flags in the IP header:

  lsrr (loose source route)
  ssrr (strict source route)
  rr (record route)
  timestamp
  all fragments
  short packet

- Comments:

  This is a free-style text field used for comments.

## 7.2. ICMP Service

The ICMP Service object is a generalized representation of ICMP protocol. ICMP packets are often used to communicate error messages which are acted upon by either the IP layer or higher layer protocols (TCP or UDP). ICMP can also be used as a simple query protocol. The firewall administrator needs to understand the nature and purpose of ICMP in order to properly configure the firewall to block unwanted and permit useful ICMP messages. ICMP packets have two header fields that distinguish particular ICMP messages: the *"type"* and *"code"* fields. There can be 15 different ICMP types, each of which may use different values for its *"code"* field to further specify the condition. Firewall Builder provides a dialog with a built-in list of all possible ICMP types and corresponding codes which helps the administrator to the pick appropriate one without having to look up these numbers in the book (Figure 7-2). The standard ICMP Service objects that come with Firewall Builder appear in the "Standard" tree, in the **Services/ICMP** branch.

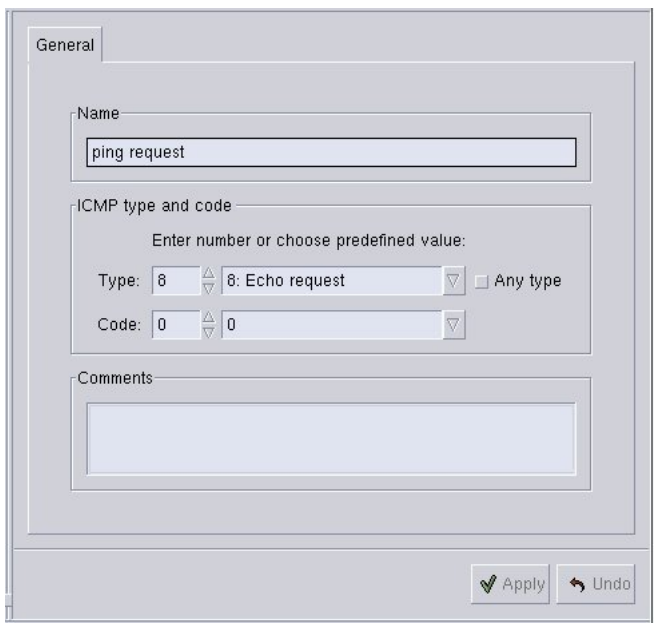



**Figure 7-2. Editing the ICMP Service Object**

The ICMP Service dialog provides the following controls:

- Name:

  This is the name of the object.

- ICMP Type and Code:
  - Type:

    The ICMP message type. This control consists of two widgets: the numeric field and the pull-down menu that represent a list of possible types and their names. You can choose a type either by number (using the numeric entry field) or by its description (using the pull-down menu).

- Code:

  The ICMP message code. Just like above, this control consists of two widgets: the numeric field and the pull-down menu that represent a list of possible codes and their names. This menu changes when you choose a different ICMP type since different types use different codes. You can choose a code either by number (using the numeric entry field) or by its description (using the pull-down menu).

- The **'Any Type'** checkbox

  Check this if you wish to create an object that would match any ICMP type and code.

- Comment:

  This is a free-style text field used for comments.

## 7.3. TCP Service

The TCP Service object is a generalization of TCP protocol which provides a connection-oriented reliable byte stream service. Many well-known frequently used application protocols use underlying TCP protocol: FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), HTTP (Hyper Text Transfer Protocol), and so on. The TCP header contains special fields called source and destination port numbers that are used to identify the sending and receiving application using the data stream. These two values, along with the source and destination IP addresses in the IP header, uniquely identify each connection.

Since port numbers are used to distinguish applications using the data stream provided by TCP protocol, each application should use a unique port number. To ensure interoperability, these numbers must be assigned by a central authority in a coordinated manner. Internet Assigned Numbers Authority (IANA) does just that; assigned TCP and UDP port numbers can be looked up at the following URL: http://www.iana.org/assignments/port-numbers . Most Unix systems also come with a /etc/services file which contains a list of assigned port numbers.

Firewall Builder comes with a collection of predefined TCP Service objects with port numbers already configured. You can simply use these objects in your policy rules and do not have to look up port numbers every time you need them.

The typical TCP Service object is represented in a screenshot (Figure 7-3.)

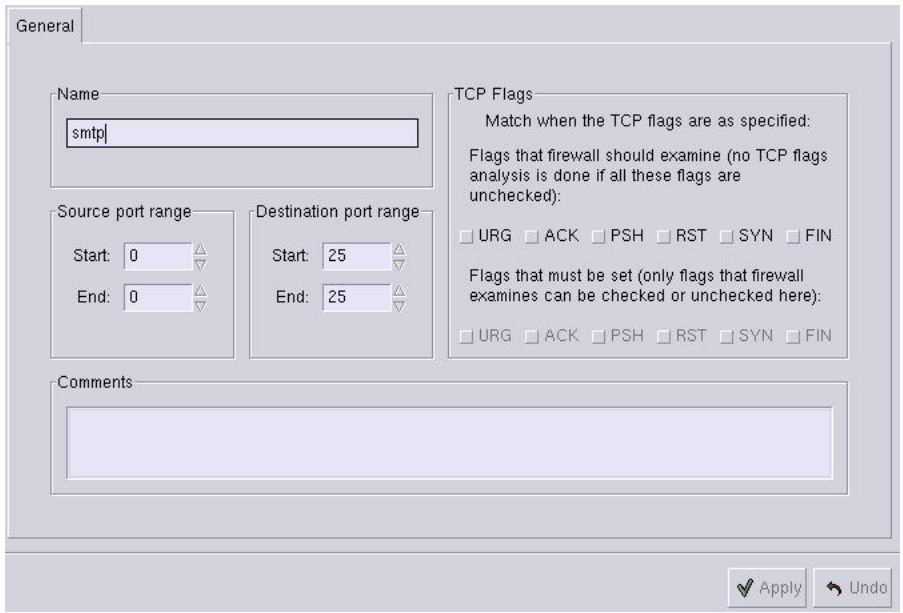



**Figure 7-3. Editing The TCP Service Object**

In Firewall Builder, the TCP Service object is a generalization of TCP protocol. The TCP header of a packet carries only one fixed value for the source port and one fixed value for the destination port. The TCP Service object allows a range of values to be used for the source and destination ports. This allows a single TCP Service object to describe either a family of protocols using consecutive port numbers, or a protocol that may use variable port numbers or simply many protocols that use port numbers from a certain range. For example, on Unix systems, TCP sessions opened by a priviliged process always have their source port number assigned from a range below 1024, while unpriviliged processes use source port numbers from a range starting from 1024 and above. The TCP Service object with a source port range defined as shown in the screenshot (Figure 7-4) describes all priviliged TCP sessions.

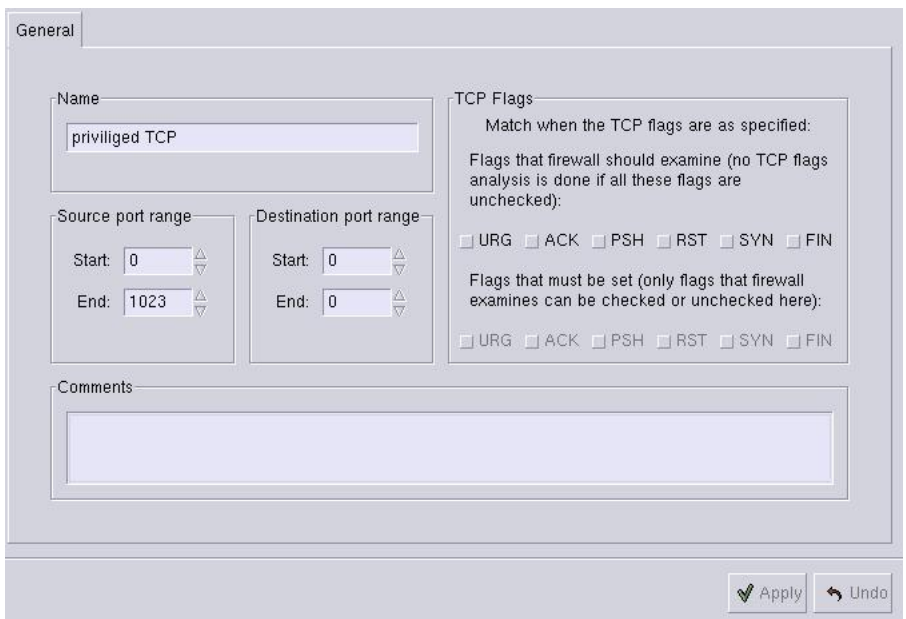



**Figure 7-4. The TCP Service Object: priviliged source ports**

Using '0' as both start and end values for a range means 'any value' for that range. The source port range of the 'priviliged TCP' object starts from 0 and ends at 1023 (port ranges are inclusive in Firewall Builder), while its destination port range's start and end are both set to zero, which means 'any destination port'. This object describes any TCP protocol with a source port in range 0-1023 and any destination port.

If all you need is to create an object to describe the TCP protocol with a particular destination port, just use the same port number for both the start and end of a destination port range (which effectively creates a range that consists of a single value). The example in Figure 7-3 shows such a service.

The TCP header also contains a collection of one-bit fields, or flags, that carry a variety of control information. For example, the SYN and ACK flags are used for connection establishment and the FIN flag is used for connection termination. Certain combinations of flags are not allowed during the normal protocol operation and may cause unpredicted reactions in some systems; because of this, the firewall administrator may want to block TCP packets with unwanted combination of flags.

There are six flags in the TCP header. We just briefly mention them here; more information can be found in *TCP/IP Illustrated, vol 1* by W. Richard Stevens, chapter 17.

| | |
|---|---|
| `URG` | The "urgent" pointer is valid |
| `ACK` | The acknowledgment number is valid |
| `PSH` | The reciever should pass this data to the application as soon as possible |
| `RST` | Reset the connection |
| `SYN` | Synchronize sequence numbers to initiate a connection. |
| `FIN` | The sender is finished sending data. |

Firewall Builder supports all six flags, although not all target firewall platforms can match all combinations of TCP flags or any flags at all (See Table 7-2).

**Table 7-2. Support for Matching against TCP flags**

| Firewall | Can inspect TCP header flags in the packet |
|---|---|
| iptables | Yes |
| ipfilter | Yes |
| pf | Yes |
| Cisco PIX | No |

Usually the firewall can not only match a combination of flags, but can also examine the only given subset of all TCP flags. Firewall builder provides two sets of checkboxes for TCP flags and flag masks (see Figure 7-5). Checkboxes in the first row control TCP flags that we want the firewall to examine and checkboxes in the second row tell it whether they should be set or cleared. Only flags whose checkboxes in the first row are set will be looked at by the firewall. To remind about this, Firewall Builder deactivates checkboxes in the second row for the flags that are to be ignored. The object in Figure 7-5 matches a TCP packet with any combination of port numbers, the TCP flag SYN set, and all other flags cleared. The firewall will examine all TCP flags.

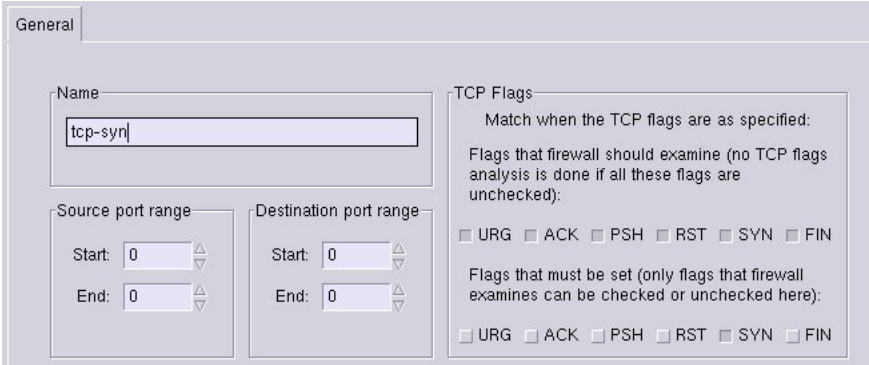



**Figure 7-5. TCP Service Object: any TCP packet with SYN flag set and all other flags cleared.**

A Combination of flags and a mask can be used in a rule that looks for some flags to be set or unset and ignores other ones regardless of their state. For example, we can create a rule that detects a so called "null scan" which is done using TCP packets with all flags cleared. For this rule we create a TCP Service object "tcp null scan" where all flag masks are set but all TCP flags are cleared. This means we examine all flags but match them if they are all cleared. This object is represented in Figure 7-6.

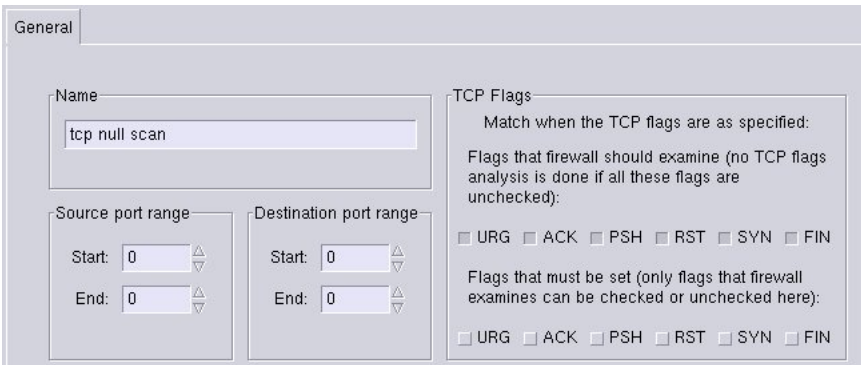



**Figure 7-6. The "null scan" TCP Service Object: the firewall examines all TCP flags and matches them if all are cleared.**

TCP Service dialog provides the following controls:

- Name:

  This is the name of the object

- Source port range:

  These two controls define the start and end of the source port range; they accept values between 0 and 65535.

- Destination port range:

  These two controls define the start and end of the destination port range; they accept values between 0 and 65535.

- TCP Flags:

TCP flags and masks, see above.

- Comments:

  This is a free-style text field used for comments.

## 7.4. UDP Service

The UDP Service object is a generalization of UDP protocol, which is a connectionless transport layer protocol. Many well known applicaitons use UDP as their transport, such as DNS (Domain Name System), DHCP (Dynamic Host Configuration Protocol), NTP (Network Time Protocol), SNMP (Simple Network Management Protocol), and so on.

Similar to that in the TCP, UDP uses port numbers to distinguish multiple applications running on one machine from one another. The UDP packet header carries two port numbers: the source port and the destination port. The UDP Service object in Firewall Builder allows for a definition of ranges for both the source and the destination ports. The meaning of values assigned to the start and end of the range is the same as in the TCP Service object: ranges are inclusive, that is, both start and end ports of the range are included. Using '0' for both the start and end of the range means 'any port'. These rules work for both the source and destination ranges. The screenshot (Figure 7-7) shows the 'dns' UDP Service object that represents the Domain Name System protocol which uses destination port 53.

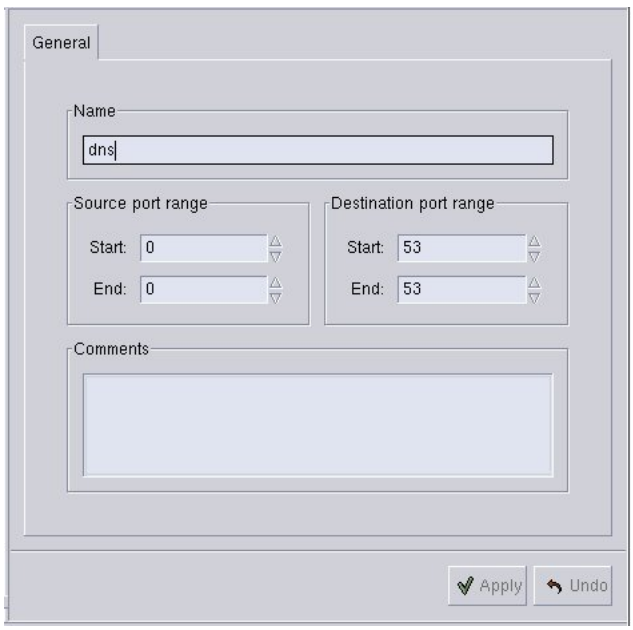



**Figure 7-7. Editing the UDP Service Object**

The UDP Service dialog provides the following controls:

- Name:

  This is the name of the object

- The Source port range:

  These two controls define the start and the end of the source port range; they accept values between 0 and 65535.

- The Destination port range:

  These two controls define the start and the end of the destination port range; they accept values between 0 and 65535.

- Comments:

  This is a free-style text field used for comments.

## 7.5. Custom Service

The Custom Service object can be used to inject arbitrary code into the generated firewall script. Any combination of options allowed in the target firewall command line language which does not fit in a strict model of standard service object types can be expressed using the Custom Service object. For example, iptables comes with a collection of patches called "Patch-o-matic" which adds an ability to match complex combinations of packet parameters or header fields that are not supported by a standard code. One of the patches adds an ability to detect a port scan condition, where the offending host tries to connect to UDP or TCP ports on the firewall or hosts behind it picking port numbers in turns trying to detect which ports are active and which are 'dead'. Normally the firewall would inspect each attempt to connect to a port separately from others and would generate many log records, one line per connection attempt. While this behavior is normal, it still fails to recognize a pattern in the actions of the offending host, which may be important in an investigation of malicious activity since port scanning is often a step in a preparation for a more focused attack.

The screenshot (Figure 7-8) represents a Custom Service object that uses the capabilities of 'PSD' module from patch-o-matic. This module can match port scan attempts that are spread in time and has a few parameters to fine tune the detection mechanism. The 'psd' Custom Object defines the command line options specific for 'psd' module in the 'Code' field.

> **Note:** The code specified in the Custom Service is used literally; no validation is done either by Firewall Builder GUI or policy compilers.

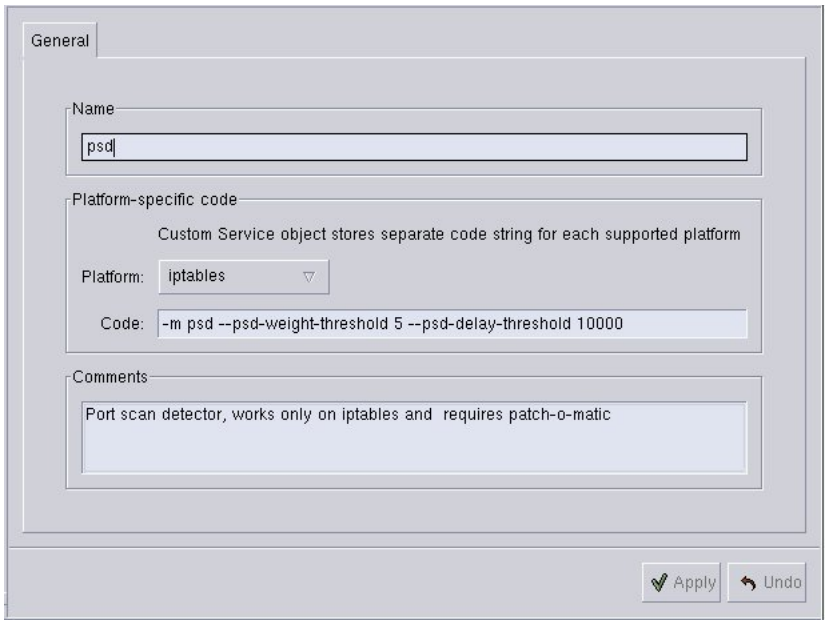



**Figure 7-8. Editing the Custom Service Object**

The Custom Service dialog provides the following controls:

- Name:

  This is the name of the object

- Platform:

  This is a pull-down menu that shows a list of all firewall platforms for which modules were installed and loaded when the GUI started.

- Code:

  This is a line of code in the target firewall language.

- Comments:

  This is a free-style text field used for comments.

The Custom Service Object created as shown in Figure 7-8 can be used in the rule just like any other service object. For example, the policy rule shown in Figure 7-9 blocks port scanning attempts launched against host *'server'*. The policy compiler translates this rule into an iptables command shown in Example 7-1:

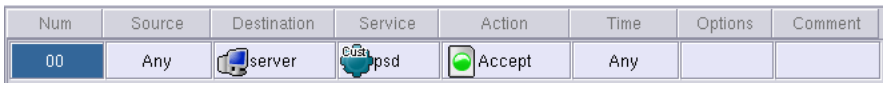



**Figure 7-9. Using The Custom Service Object in The Policy Rule**

**Example 7-1. Iptables commands generated for the policy rule Figure 7-9**

```
$IPTABLES -A OUTPUT  -d 10.3.14.100 -m psd --psd-weight-threshold 5 --psd-delay-threshold 10000
$IPTABLES -A FORWARD  -d 10.3.14.100 -m psd --psd-weight-threshold 5 --psd-delay-threshold 10000
```

In general, Custom service can store an arbitrary piece of code for each supported firewall platform. When this object is used in the firewall policy, the policy compiler picks the appropriate line of code depending on what target firewall platform the code is being compiled for.

# Chapter 8. Firewall Policies

## 8.1. Policies and Rules

Each firewall object has several sets of rules associated with it: Global Policy, Interface Policies and Network Address Translation (NAT) rules. Rules in the global policy and in interface policies control access to and from the firewall machine and machines behind it, while NAT rules describe address and port transformations that the firewall should make to the packets flowing through it.

Firewall software varies widely in the way it can process packets. For example, some firewalls require all rules to be associated with the interface, while others may not have such requirement. Some firewalls perform address and port transformations first and then apply policy rules, while some others do it the other way around. There are many other variations and features specific to particular implementations. In Firewall Builder though, the user works with an abstract firewall that looks uniform regardless of the target firewall platform it is used to generate code for. For example, Cisco PIX applies its Access List rules to the packet before it performs address and port transformations according to the NAT rules. This makes this firewall platform substantially different and inconvenient because a policy rule that controls access to the server behind the firewall doing NAT should be written using the firewall's object instead of the server's object. The meaning of such a rule is not obvious at a glance since one has to keep in mind all the NAT rules as well to understand that this policy rule controls access not to the firewall machine, but rather to the server behind it. Firewall Builder compensates for variations like this using smart algorithms to transform rules defined in the GUI to generate code in the language of the target firewall platform to achieve the desired effect.

Figure 8-1 represents the logical sequence in which rules defined in Firewall Builder affect the network packet. This diagram describes an abstract firewall which Firewall Builder represents for the user. In some cases the target firewall may work the same way, in some other cases it won't. Either way, the goal is for the administrator to build the rules assuming his firewall works like shown in Figure 8-1. The firewall may have multiple interfaces and multiple interface policies. Since the interface is one of the parameters of the rule, and rules in different Interface Policies are tied to different interfaces, they are orthogonal. Because of that, it does not matter in which order Interface Policies are applied. It is guaranteed however that Interface Policies are stacked on top of the Global Policy and inspect the packet before the Global Policy will.

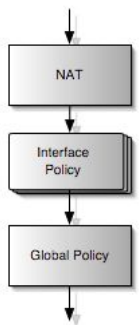



**Figure 8-1. The sequence in which NAT and Policy rules apply to the packet in Firewall Builder**

Among three major types of supported sets of rules, Global Policy rules and Interface Policy rules are similar. These rules represent access control because they define which packets will be permitted and which will be denied. Both Global and Interface Policy rules operate on the packet's source and destination addresses and protocol parameters. The difference is that Interface rules also take into account the firewall's interface which the packet is crossing in one or another direction. Global Policy rules work regardless of the interface. In the case of iptables, the Global Policy is roughly equivalent to the iptables commands that do not specify interface using the "-i" or "-o" option; however there might be no such direct correspondence in other firewall platforms. Interface Policy rules are associated with a particular interface of the firewall and have an option that describes the direction in which the packet should cross the interface in order for the rule to match it.

**Note:** Although this may be counter-intuitive for those who are very familiar with iptables (netfilter), Interface Policies are NOT directly converted into INPUT and OUTPUT chains. In fact, certain rules in the Interface Policy may still go into the FORWARD chain. Only rules where the firewall object or one of its interfaces is used in the destination generate iptables rules that go into the INPUT chain. Similarly, rules that have the firewall or one of its interfaces in the source generate iptables rules that go into the OUTPUT chain. Such rules may happen in both Global and Interface Policies. What makes Global and Interface policies different is a reference to the interface, which in iptables corresponds to "-i" and "-o" clauses with the corresponding interface name.

### 8.1.1. Actions

The policy rule action can be one of the following: *Acccept*, *Deny* or *Reject*. If a rule's action is Accept, then a packet with the source and destination addresses and service matching the rule is permitted through. If a rule's action is Deny, such packet is silently dropped. If a rule's action is Reject, then the packet is dropped and an appropriate ICMP message is sent back to the sender. For target platforms that support it, there might be an option to send a TCP RST packet back in response for TCP services. Figure 8-2 illustrates rule actions.

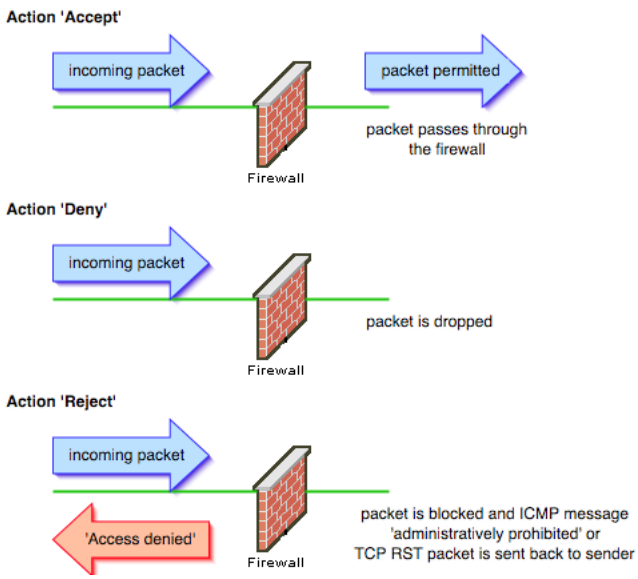



**Figure 8-2. Actions**

Some firewall platforms also support the *Accounting* action. A rule with this action is supposed to only count packets that match it, but it does not make any decision as to whether the packet should be permitted or denied. Rules with this action are an exception because even if the packet matches such a rule, the inspection process continues with other rules below it. The Figure 8-3, rule #0 counts packets that are part of *HTTP* sessions opened through the firewall. Rule #1 permits *HTTP* access to the server *server* from any source and rule #2 permits the internal network to connect to any destination. Since accounting rule #0 has *any* in its source and destination fields, it will count both accesses to the web server *server* and traffic created when users on the internal net connect to the web sites out on the Internet.

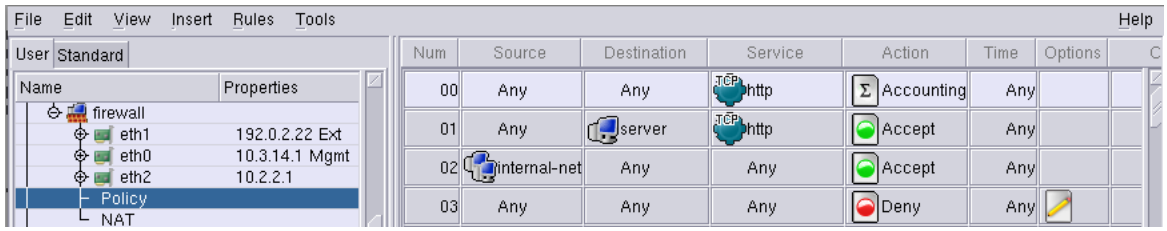



**Figure 8-3. Example of an Accounting Action**

## 8.1.2. Directions

The direction is being defined with respect to the firewall machine, not to the network behind it. For example, packets that leave the internal network through the firewall are considered "inbound" on firewall's internal interface and "outbound" on its external interface. Likewise, packets that come from the Internet are "inbound" on the firewall's external interface and "outbound" on its internal interface. Figure 8-4 illustrates the definition of the direction for packets entering or exiting the firewall interface.

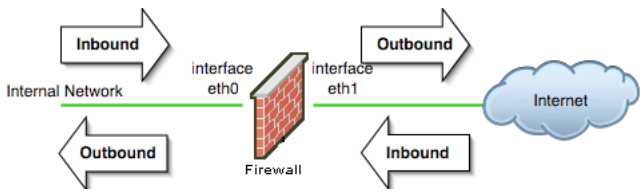



**Figure 8-4. Directions**

### 8.1.3. Global Firewall Policy

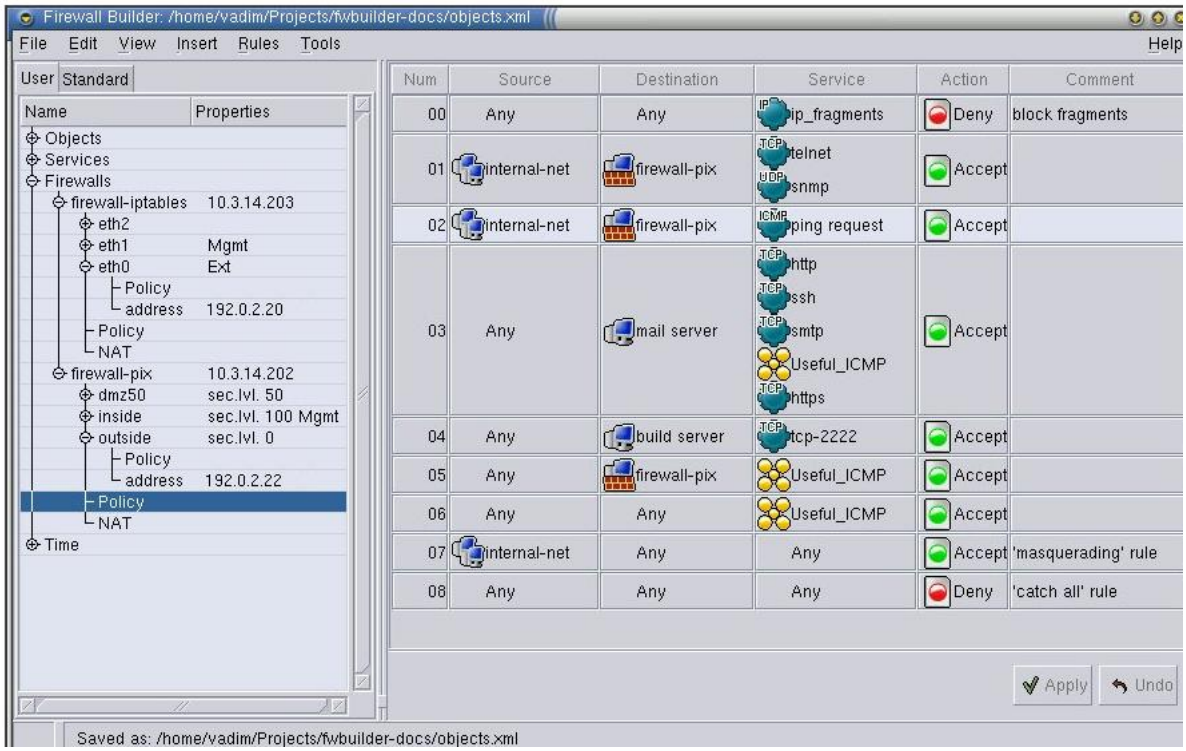



**Figure 8-5. Global Policy**

The Firewall policy consists of a set of rules. Packets are analysed by comparing their parameters with rules in the policy; inspecting rules in an orderly manner from top to bottom. The first rule that matches the packet is used. Each rule includes a standard set of parameters, or *Rule Elements*, which are used to compare the data packet with, such as "Source", "Destination", "Service", "Direction". If a rule is a part of the Interface Policy (Section 8.1.4), then the interface it belongs to acts as yet another parameter. For example, if a packet entering the firewall has a source address which matches the object in the "Source" field of the rule, its destination address matches the object in "Destination", its protocol and port numbers match the object in the field "Service", and its direction matches that specified in the field "Direction", then the firewall will take action, specified in the "Action" field of the matching rule. In the case of Interface rule, the firewall will also take into account the interface through which packet had entered or through which it is about to leave the firewall.

For example on Figure 8-5, the rule #0 says that all fragmented packets should be dropped (the rule uses service object *ip_fragments* and the *Deny* action). Rule #1 says that connections from the internal network (object *internal-net*) to the firewall itself (object *firewall-pix*) using protocols *telnet* and *snmp* are allowed (action *Accept*). The "Catch all" rule #8 denies all packets that have not been matched by any other rule above it. The global policy on Figure 8-5 is built to allow only specific services and deny everything else, which is a good practice.

Section 8.4 lists rule elements available in the policy, depending on the chosen target firewall platform.

### 8.1.4. Interface Policy




**Figure 8-6. Interface Policy for iptables firewall**




**Figure 8-7. Interface Policy for Cisco PIX firewall**

Each firewall's interface may have a security policy associated with it. Different firewalls require different commands to attach policy rules to the firewall, but policy compilers take care of that and generate the appropriate code. The Interface Policy consists of a set of rules; each rule has the same rule elements as a Global Policy rule and adds some additional ones.

In Figure 8-6 rule #0 (the so called Anti-spoofing rule) says that all packets which enter the firewall's interface eth0 (direction is *"Inbound"*) and have their source address match either the internal network or dmz network or any address that belongs to the firewall itself, should be dropped and a record should be made in the log (since there is a "notebook and pencil" icon in the *"Options"* rule element).

## 8.2. Network Address Translation Rules

### 8.2.1. Basic NAT Rules

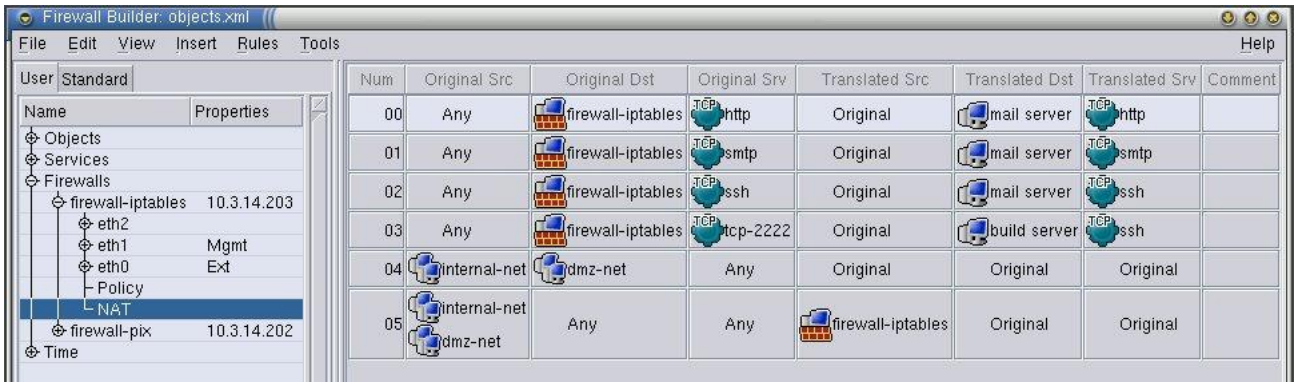



**Figure 8-8. Network Address Translation Rules**

Just like the firewall policy, the Network Address Translation (NAT) rules are inspected by the firewall in the order they appear in the NAT policy. Each NAT rule consists of the following rule elements:

- Original Source

  This is the object which the source address of original packet should match for it to be translated by this rule

- Original Destination

  This is the object which the destination address of original packet should match for it to be translated by this rule

- Original Service

  This is the service which the original packet should match to be translated by this rule

- Translated Source

  This object defines the address which should be used to substitute the original packet's source address

- Translated Destination

  This object defines the address which should be used to substitute the destination address in the original packet.

- Translated Service

  This object defines the protocol and port numbers which should be used to substitute those parameters in the original packet

- Comment

Here is how it works:

The original packet is compared with NAT rules, one at a time, starting with the topmost rule. Once the rule that matches packet's source and destination addresses and its service is found, the firewall takes parameters from the second half of that rule and makes substitutions for the source and destination addresses and port numbers. Some rule elements in the first half of the rule may be left empty (they show "Any" in this case); these elements match any value of a corresponding parameter of the packet. Some rule elements in the second half of the rule may also be left empty (in which case the GUI shows the word "Original" in the rule element). This tells the firewall that no substitution should happen for those elements.

In fact the firewall not only substitutes addresses and port numbers, it also makes a record in its internal table to remember the original packet and translated packet it just produced. This way the firewall can perform a reverse translation when the reply packet comes back.

The NAT rules in the screenshot (Figure 8-8) tell the firewall to do the following:

- Rule #0:

  If the original packet has a destination address which matches the addresses of an external interface of the firewall object "firewall-iptables" and protocol TCP and port 80 (matches service "http", protocol used by web servers), then its destination address should be changed to that of the object "mail server" and the service should be left the same. Note that this rule would have had the same effect if the "Translated Service" was left "Original".

- Rule #1:

  This rule is very much the same as #0, except it works for packets headed for TCP port 25 (service "smtp" which describes protocol used to send and receive Internet email)

- Rule #2:

  This rule is like #0 and #1, except it works for the service "ssh"

- Rule #3:

  This rule follows the same pattern as #0-2, but translates packets headed for the external firewall's address, TCP port 2222, and sends them to the standard TCP port that belongs to the service "ssh" on the internal machine "build server". This rule illustrates that you can simultaneously translate both addresses and services: it makes changes to the destination address of the packet and the service.

- Rule #4:

  This rule matches packets that originated in the internal network (object "internal-net") and headed for DMZ net (object "dmz-net"). Since all rule elements in the second half of the rule are left blank, this rule tells the firewall that no translation is needed for these packets.

- Rule #5:

  This rule defines another type of translation: all packets originated in either the internal network or in DMZ net, going to any destination, should have their source address replaced with the address of the firewall's external interface. This type of translation rule is sometimes called the "Masquerading".

Some firewall platforms support negation in NAT rules. If it is supported, this feature can be activated through the pop-up menu ( Figure 8-18 ) that appears when you click the right mouse button on the rule element in the NAT rule. Section 8.4 shows what firewall platforms support negation in NAT.

You can create NAT rules and edit them using the same methods as described in Section 8.3

### 8.2.2. Source Address Translation

Suppose your internal network is configured using so-called private IP addresses as defined in RFC 1918. These addresses are:

- 10.0.0.0 / 255.0.0.0
- 172.16.0.0 / 255.240.0.0
- 192.168.0.0 / 255.255.0.0

If your network uses addresses from one of these blocks, and you want to provide Internet access for machines on this net, then you may need Network Address Translation (NAT) configured on the firewall. This particular type of NAT configuration is often called Masquerading. When configured this way, the firewall rewrites source IP address of each packet sent by internal machines to the Internet, replacing private IP address with the address of its external interface.

In Firewall Builder this type of the NAT rule is composed as shown on Figure 8-9

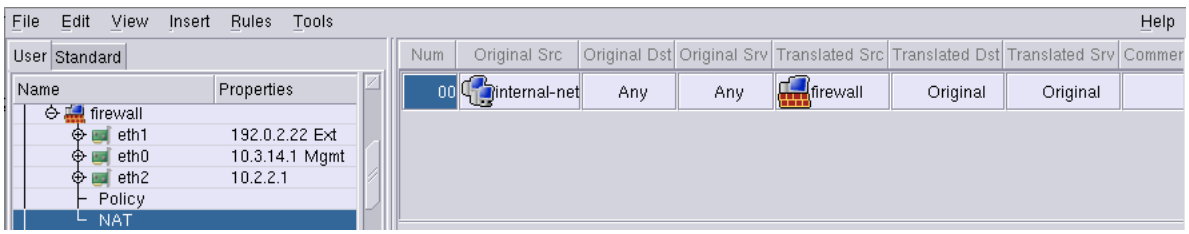



**Figure 8-9. Source Address Translation Rule**

In this rule object representing internal network is placed in "Original Source" and firewall's object is placed in "Translated Source", indicating that we want source address of the packets to be translated. As before, we do not need to worry about reply packets because underlying firewall software keeps track of translations done for all the connections opened through the firewall and rewrites addresses in all reply packets automatically.

Figure 8-9 uses firewall object in Translated Source, which means source address of the packet will be substituted with the addresses of firewall's interfaces. Since the firewall has more than one interface, this rule will use all of them in a such fasion that packets exiting the firewall through interface *"eth1"* will have their source address translated into address of that interface, packets exiting through *"eth2"* will have its interface used for their source address and so on. Decision on which interface given packet should exit the firewall through is made by firewall's TCP/IP stack using its routing table.

One of the consequences of this design is that rule #0 on Figure 8-9 provides translation for packets coming from *internal_net* and going out to the Internet, as well as those coming from *internal_net* and going back to the same net. The latter case may sound strange and useless, but we'll see later that it may have some very practical applications (see ... ).

But what if the firewall has three or more interfaces ? This happens if it has multiple connections to the Internet or other insecure subnets like DMZ. Following the same logic, the translation will be performed using all interfaces of the firewall. Figure 8-12 (A) illistrates this.

There may be situations when we would not want to automatically use all interfaces of the firewall for translation. Suppose firewall "firewall" has two insecure interfaces, one of which connects it to the Internet, while another one is connected to DMZ subnet which is using private IP addresses on the net 10.2.1.0. We want to translate the source address of packets going out to the Internet through external interface*eth1* and do not want any other translation to happen. Rule #0 in Figure 8-10 does just that.

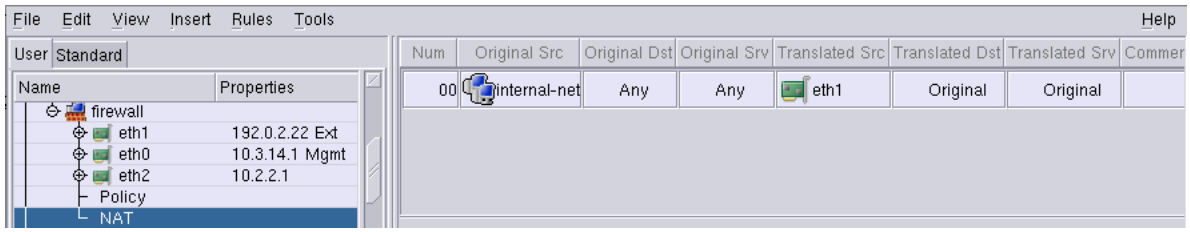



**Figure 8-10. Source Address Translation Rule Using Interface of the Firewall**

Rule #0 in Figure 8-10 uses firewall's interface "eth1" for translation. This means that only packets exiting through this interface will be translated; packets exiting through *"eth2"* will keep their original source address on the internal net. Figure 8-12 (B) illistrates this.

> **Note:** Interface object can be used in the NAT rules even if the address of this interface is obtained dynamically and is not known beforehand.

Another possibility is when neither firewall itself, nor one of its interfaces is used in Translated Source in the rule. The translation is similar to the one in Figure 8-12 (A) except the same address is used regardless of the interface the packet exits firewall through. This case is illustrated in Figure 8-11 and Figure 8-12 (C).

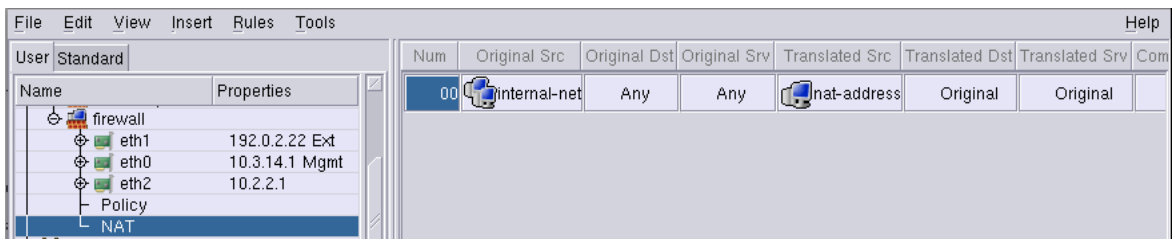



**Figure 8-11. Source Address Translation Rule Using Host Object**

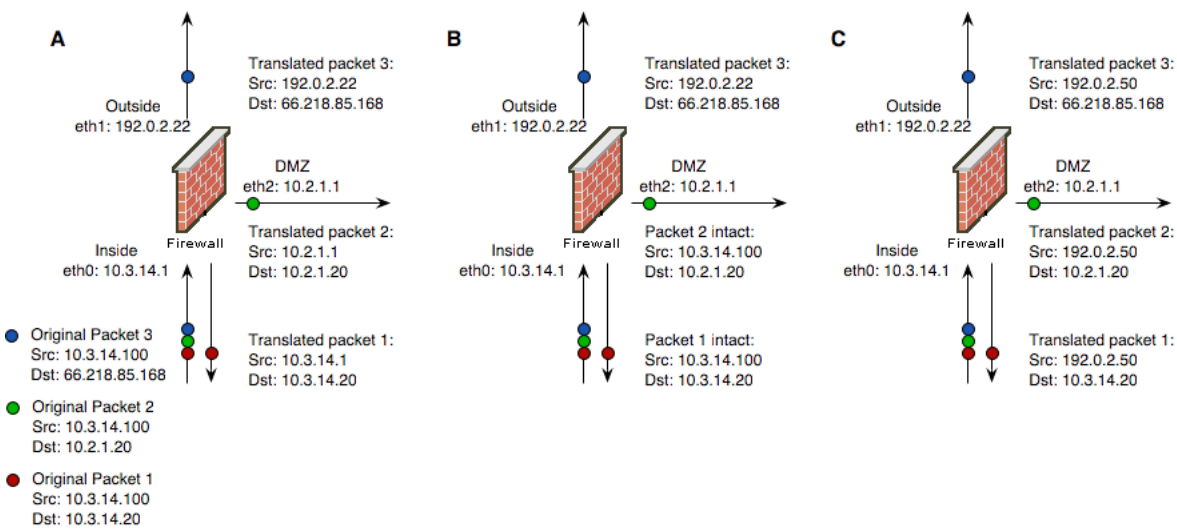



**Figure 8-12. Translations done to packets going in different directions: (A) when firewall object is used in**

**TSrc in the NAT rule; (B) when interface eth1 is used in TSrc in the NAT rule; (C) when host object with address 192.0.2.50 is used in TSrc in the NAT rule**

### 8.2.3. Destination Address Translation

Suppose we have a network using private IP addresses behind the firewall, and a server on it. We need to provide access to this server from the Internet in a such way that connections will be established to the address of the firewall. In this case we need destination address of packets to be rewritten so packets would reach the server on internal network. The simplest rule that translates destination address of incoming packets looks like the one on Figure 8-13:

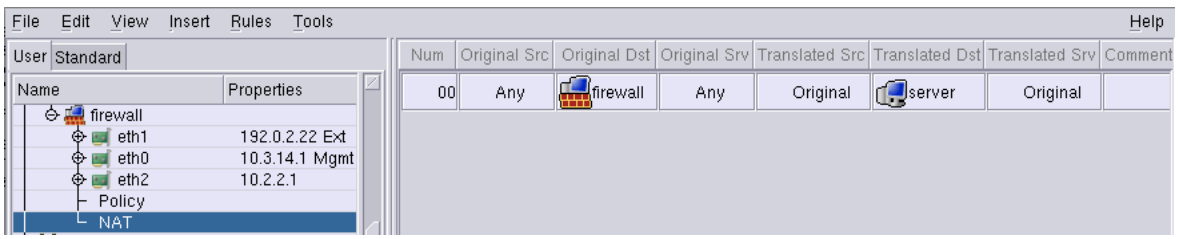



**Figure 8-13. Simple Destination Address Translation Rule**

Basically this rule says "if destination address of the packet matches one of the addresses of the firewall, replace it with the address defined by the object "server". Note that since we used firewall object in Original Destination, all addresses of all interfaces of the firewall are used for translation. Figure 8-16 (A) illustrates this. The red, green and blue packets come to the firewall from different subnets and all have destination addresses that match address of the corresponding interface. If it were not for our NAT rule, packets like that would have been accepted by the firewall and sent to a process expecting them. However NAT rule comes to play and changes destination address of all three packets to 10.3.14.100 (the address of server). Packets with this address do not match any address belonging to the firewall and therefore get sent out of the firewall according to the rules of routing.

The rule in Figure 8-13 does not specify any service for the translation and therefore translates addresses in packets of all protocols. This actually makes this rule rather impractical because it will translate and bounce any packets that are headed for the firewall, making it impossible to connect to the firewall itself using telnet or any other protocol. This is especially inconvenient since as we saw earlier translation happens for packets coming from all directions; this means that you won't be able to connect to the firewall even from inside of your network. To alleviate this problem we just add an appropriate service object to the rule as shown in Figure 8-14:

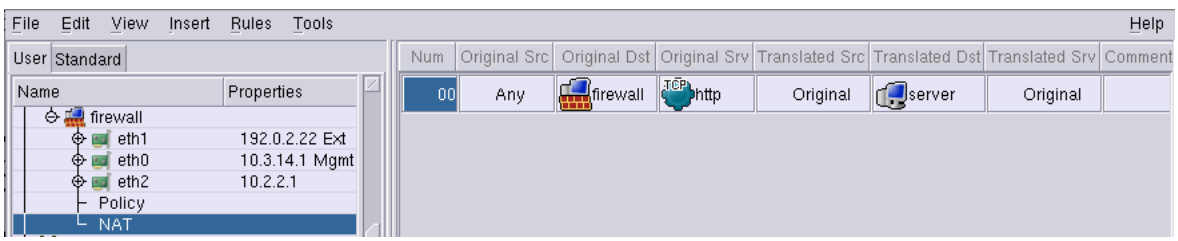



**Figure 8-14. Translation limited to packets of HTTP protocol**

Rule #0 in Figure 8-14 has limited scope because of the service object "http" in Original Service; it matches and performs address translation only for packets of HTTP protocol, while other packets are processed by TCP/IP stack on the firewall as usual. Very often we only want to translate address for packets coming from particular side of the firewall, typically from the Internet, and do not change other packets. Rule #0 on Figure 8-15 achieves this goal by using firewall's interface object in Original Destination. Only packets with destination address the same as that of interface eth1 of the firewall match this rule and get their address translated. Packets coming from other directions will have different destination address and won't match the rule (see Figure 8-16 (B) ).

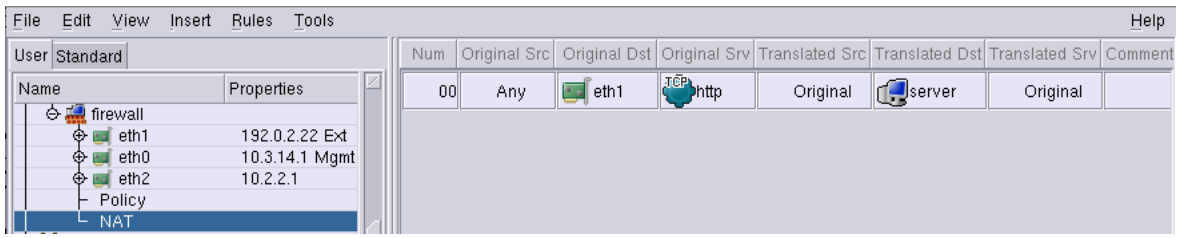



**Figure 8-15. Destination Address Translation Rule Using Interface of the Firewall**

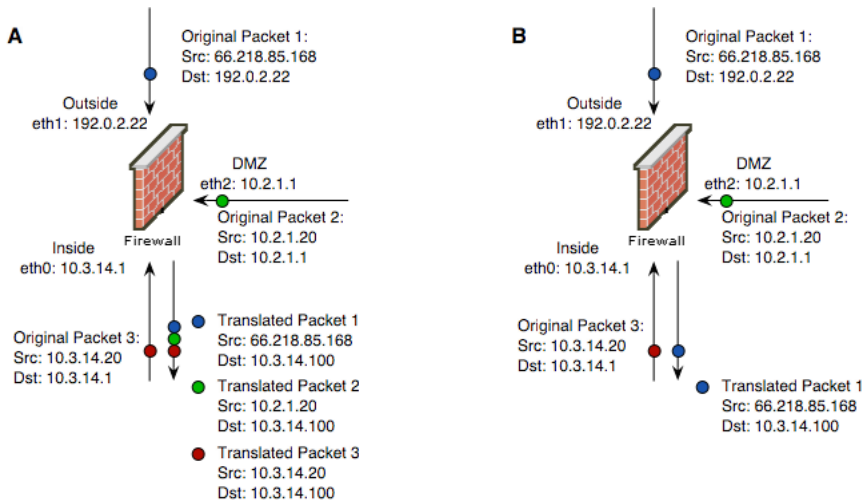



**Figure 8-16. Translations done to packets going in different directions: (A) when firewall object is used in ODst in the NAT rule and (B) when interface eth1 is used in ODst in the NAT rule**

## 8.3. Editing firewall Policy and NAT Rules

### 8.3.1. Adding and removing rules

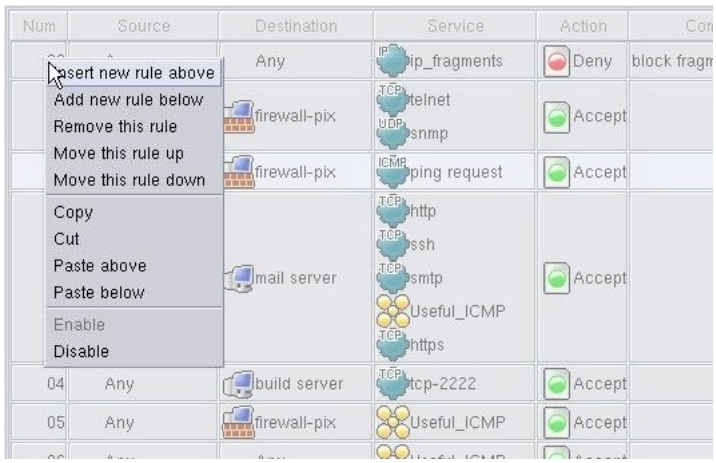



**Figure 8-17. Modifying Policy rules**

Rules can be added, removed or moved around in the policy using commands in a popup menu which appears if you click the right mouse button on the rule element "Num" (the very first rule element, counting from the left). Screenshot Figure 8-17 illustrates this menu.

This menu allows for adding new rules above or below of the currently selected rule in the policy, removing rules, moving the current rule up and down, plus the usual **"Copy"/"Paste"** operations done on policy rules.

Functions of this pop-up menu are also duplicated in the main menu **"Rules"**

You can also use the drag and drop operation to move rules up and down in the policy. Just drag the rule with the mouse by its "number" rule element.

### 8.3.2. Adding, removing and modifying objects in the policy and NAT rules

To add objects to the policy or NAT rule you can either drag them from the tree and drop them into the corresponding rule element, or use the **"Copy"/"Paste"** operation. Objects can be copied into clipboard both from the tree and from another policy rule; in both cases use the pop-up menu or main menu **"Edit"**.

Clicking the right mouse button when the cursor is over rule elements "Source","Destination" or "Service" brings a context-sensitive pop-up menu ( Figure 8-18 ). The same pop-up menu appears when you do that over the "Original Source", "Original Destination", "Original Service", "Translated Source", "Translated Destination" and "Translated Service" rule elements in the NAT rule.

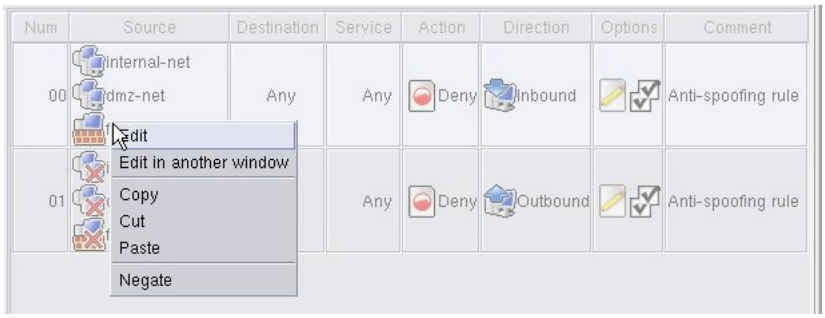



**Figure 8-18. Modifying Object in the Policy Rule**

This menu provides items for the following functions:

- Edit:

  This menu item opens the currently selected object in the dialog

- Edit in a new window:

  This menu item creates a new main window and opens the currently selected object in it. If more than one main window currently exists, the object will be opened in one of the existing windows.

- Copy:

  The object is copied into clipboard.

- Cut:

  The object is copied into clipboard and removed from the rule.

- Paste:

  The object is copied from the clipboard to the rule.

- Negate:

  All objects in the selected rule element are negated. The rule element "Source" is negated in rule #1 in screenshot Figure 8-18

### 8.3.3. Changing rule action

Clicking the right mouse button when the cursor is over the rule element "Action" brings a context-sensitive pop-up menu ( Figure 8-19 ).

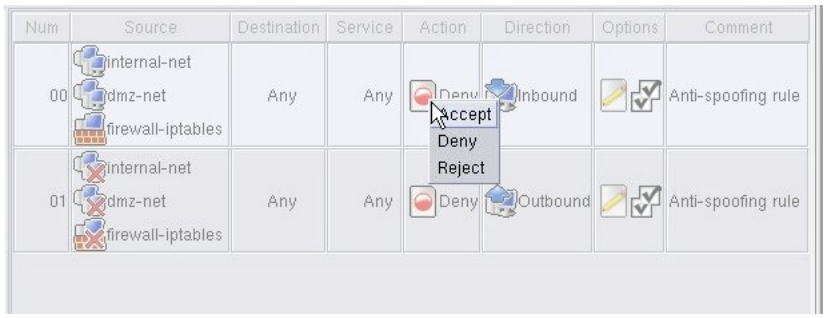



**Figure 8-19. Modifying the Action of the Policy Rule**

This menu provides items for the following functions:

- Accept:

  The rule will accept the packet matching it

- Deny:

  The rule will drop the packet matching it

- Reject:

  The rule will drop the packet matching it, and will also send appropriate ICMP responce message back to source.

### 8.3.4. Changing rule direction

Clicking the right mouse button when the cursor is over the rule element "Direction" brings a context-sensitive pop-up menu ( Figure 8-20 ).




**Figure 8-20. Modifying the Direction of the Policy Rule**

This menu provides items for the following functions:

- Inbound:

  The rule matches packets entering the firewall

- Outbound:

The rule matches packets exiting the firewall

- Both:

  The rule matches both entering and exiting packets

### 8.3.5. Changing rule options and logging

clicking the right mouse button when the cursor is over the rule element "Options" brings up a context-sensitive pop-up menu ( Figure 8-21. )




**Figure 8-21. Modifying Options of the Policy Rule**

This menu provides items for the following functions:

- Modify Options:

  This menu item brings up a dialogue which allows you to modify certain options specific to the target firewall platform and associated only with the current rule.

- Turn logging ON:

  Turns logging on for all packets matching this rule. If the target firewall platform does not support selective logging of packets, this menu item is disabled.

- Turn logging OFF:

  Turn logging off for packets matching this rule. If the target firewall platform does not support selective logging of packets, this menu item is disabled.

## 8.4. Support for Rule Elements and Features on Various Firewalls

Certain fields in the rules are only available if the target firewall platform supports them. For example, the iptables firewall provides controls for logging of matched packets, while Cisco PIX does not; PIX always logs every packet it drops. Where possible, the policy compilers are trying to emulate the missing feature. For example, OpenBSD PF does not support negation natively, but policy compiler *"fwb_pf"* provides a workaround and tries to emulate this feature for PF. Another example is policy rules with "Outbound" direction. Cisco PIX supports only inbound Access Lists, so the policy compiler emulates outbound Access Lists while generating configuration for PIX. Table 8-1 represents a list of fields in the rules and which firewall platforms support

them. Information about these fields and features is available for fwbuilder GUI that disables corresponding menu items and hides associated policy elements when they are not supported.

**Table 8-1.**

| Firewall Platform | Source | Destination | Service | Time Interval | Direction | Action | Logging/ Options | Comment | Negation in Policy rules | Negation in NAT rules |
|---|---|---|---|---|---|---|---|---|---|---|
| iptables | + | + | + | + | + | + | + | + | + | + |
| ipfilter | + | + | + | - | + | + | + | + | + | - |
| pf | + | + | + | - | + | + | + | + | + | + |
| Cisco PIX | + | + | + | - | + | + | - | + | - | - |

## 8.5. Using Objects With Multiple Addresses in the Policy and NAT rules

One of the great advantages of the object-oriented approach to building firewall's policy is that it helps simplify the representation of the policy. Host and Firewall objects can have child Interface objects, which in turn have child Address and Physical Address objects. Any object in the hierarchy Host - Interface - Address/Physical Address can be used in the policy and compiler uses all objects in the hierarchy down from the one used in the policy to build the rule.

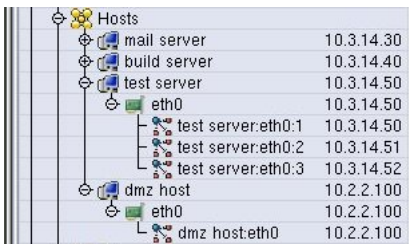



**Figure 8-22. Host object with an interface that has multiple addresses**




**Figure 8-23. Using objects with multiple addresses in the policy rules**

Consider example Figure 8-22, Figure 8-23. Here interface *eth0* of the Host object *"test server"* has three IP addresses (named *"test server:eth0:1"* through *"test server:eth0:3"*) and interface *eth0* of the Host object *"dmz host"* has only one IP address *"dmz host:eth0"*. Policy rule #4 says that *"dmz host"* can talk to *"test server"* using any protocol. Since *"test server"* has three different addresses, we need to generate policy rule that will match any of them (obviously not on all three at once, that's impossible, so we have to use logical operation "OR", not "AND" here.) Basically, rule #4 is equivalent to three separate rules, each of them using one address of *"test server"* in turn. These three rules are represented in Figure 8-24 (original rule #4 also shown there, but it is disabled.)



| Num | Source | Destination | Service | Action |
|---|---|---|---|---|
| 04 | dmz host | test server | Any | Accept |
| 05 | dmz host:eth0 | test server:eth0:1 | Any | Accept |
| 06 | dmz host:eth0 | test server:eth0:2 | Any | Accept |
| 07 | dmz host:eth0 | test server:eth0:3 | Any | Accept |

**Figure 8-24. Equivalent rules**

Firewall Builder takes care of this situation automatically and generates firewall policy described in Figure 8-23 as if a user have built a policy in the GUI using the three rules as shown in Figure 8-24.

In fact, the algorithm it uses is even more general. In the example Figure 8-23 host *"test server"* has single interface with multiple addresses which compiler used to generate target firewall code. The policy compiler works in a similar way even if host or firewall object used in the rule has multiple interfaces and each interface, in turn, has multiple addresses. If host (or firewall) object is used in the rule, then the compiler scans all its interfaces, finds all corresponding addresses and uses them to generate firewall configuration. If interface object is used in the rule, then the compiler uses all its addresses. And finally, if Address or Physical Address object is used in the rule, then compiler uses only this parameter to generate the target firewall configuration. In other words, compiler always traverses the tree, starting from the object found in the policy rule and uses parameters of all Address and Physical Address objects it finds. Since Address and Physical Address objects are the leaf nodes in the tree and have no other objects beneath them, the compiler uses parameters of the objects of these types to generate the target code.

**Note:** There is an exception to this algorithm, see Section 5.5.1

# Chapter 9. Examples of Policy Rules

## 9.1. Letting certain protocols through, while blocking everything else

This is one of the simplest, most basic tasks you may want your firewall to do - block all the traffic while letting certain protocols through. Let's assume that we have a network consisting of just the firewall "firewall1" and a few hosts behind it. We want to let protocols SMTP and FTP through to the server "hostA" from the Internet, and block everything else. All we need to do is put the following rules in the Global Policy:




**Figure 9-1. Example of a rule permitting only certain protocols to the server and blocking everything else.**

Rule #0 allows SMTP and FTP through to the server, while rule #1 blocks and logs everything else. It is worth mentioning that this policy also blocks all the access to firewall itself, including access to it from internal hosts.

We do not need any additional rules to take care of "reply" packets coming back from the server to clients because our underlying firewall software supports stateful inspection and "understands" that such packets should be let through.

## 9.2. Interchangeable and non-interchangeable objects

In the previous example we put object "hostA" into the Destination field of the policy rule #0 because our goal was to permit protocols SMTP and FTP to that host and not to any other one. This actually reflects general principle Firewall Builder is based on: put the object you want to control access for in the Source or Destination field of the policy rule. Two different objects with the same address may or may not be interchangeable, depending on their type and other parameters. One of the frequent mistakes is to create Host object with IP address of the firewall, then use it in the policy and expect Firewall Builder to build policy controlling access to the firewall. Unfortunately it does not always work that way. If you wish to control access to or from the firewall machine, then put Firewall object into the policy rule.

Another example of two objects which may on the first glance represent the same thing, but in fact are not interchangeable, is IP service object with protocol number set to 1 and ICMP service object with type and code set to "any". Both objects might represent the same type of service, namely "Any ICMP message". IP protocol 1 is in fact ICMP, so one would expect the behaviour of the firewall to be the same regardless of what type of service object is used. However, target firewall software typically uses special syntax for indication of different protocols, so using proper syntax for ICMP protocol turns certain features on, like for example session state tracking. Using just IP with protocol number 1 will most likely not turn these features on and therefore will lead to unexpected results.

On the other hand, two objects of the same type with different names but the same values of all other parameters are perfectly interchangeable. Using different objects to describe the same object may be confusing, but the final firewall policy will be correct. Firewall Builder leaves design of the objects up to the firewall administrator.

## 9.3. Using groups

Sometimes we need to define a lot of very similar rules for multiple hosts or networks. For example, there may be a need to permit the same service to 10 different hosts on the network, while still blocking it to all others. The simplest way to accomplish this is to add 10 rules with the same source and service fields and just different destinations. Obviously this can make firewall policy quite cluttered. To avoid this and make policy

readable we can use groups. Group is just a container which includes multiple objects of the same or similar type. Firewall Builder supports groups of objects and groups of services. One can put "Host", "Network" and "Firewall" objects in the group of objects; service objects can not be put in a such group. Similarly group of services can contain "IP Service", "TCP Service", "UDP Service" and "ICMP Service" objects and can not contain hosts or networks. Groups can contain other groups of the same type as well. Figure 5-17 represents an example of the object group.

To add objects to a group simply drag them from the tree on the left into group view on the right, or use Copy/Paste functions available via menus.

Once appropriate group has been created, it can be used for the policy and NAT rules just like any other object.




**Figure 9-2. Example of a rule using object group.**

## 9.4. Firewall talking to itself

Many services running on the firewall machine need to be able to establish connections to the same machine. X11, RPC, DNS are services like that, to name a few. Blocking these services on the firewall can cause various problems, depending on what protocol is being blocked. If it is DNS, then it may take a lot longer than usual to get to a command line prompt when logging in to the machine using telnet or ssh. Once logged in, you won't be able to resolve any host names into addresses. If X11 is blocked, then X server and any graphic environment using it (KDE, Gnome etc.) won't start. In any case though the problem can easily be solved by adding a simple rule to the loopback interface of the firewall to permit all sorts of commincations. As shown on Figure 9-3, this rule should use firewall object in both source and destination, have action *Accept* and direction *Both*.

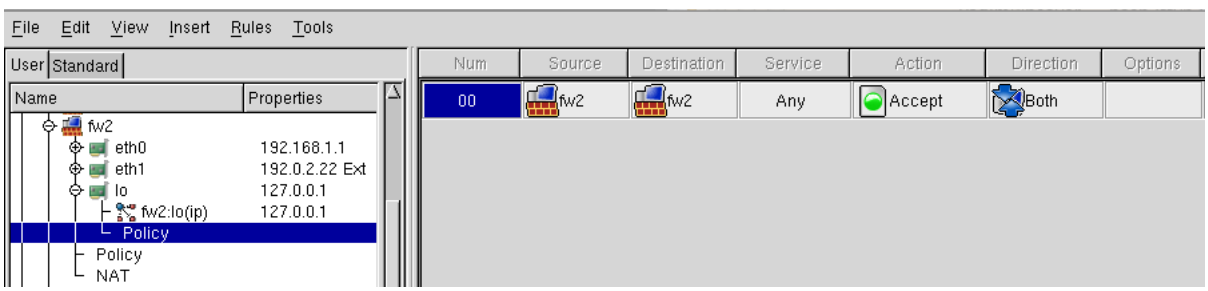



**Figure 9-3. Rule permitting everything on the loopback interface**

**Note:** Running X11 and other complex services on the dedicated firewall machine should be discouraged, however recommendations given in this chapter do apply to the firewall protecting a server or a workstation where X11, RPC and other services are perfectly normal.

## 9.5. Blocking unwanted types of packets

Fragmented IP packets, although useful in certain situations, are often used as a tool to probe and penetrate simple packet filters. Particular kind of fragmented packets, namely those with incorrect length specification, are especially bad because they can cause some operating systems to crash (for example Windows NT was known to crash before fix was developed and published by Microsoft). These packets therefore are considered potentially harmful and should be blocked on the perimeter of your network. Many firewall platforms provide ways to deal with such packets.

In Firewall Builder, we provide a way to set flags or options in the IP service object. Two options deal with fragments: one is called "all fragments" and another "short fragments". Figure 9-4 shows how does object "ip_fragments" look like with both options turned on. Policy compilers recognize this object and generate correct code for underlying firewall software platform.

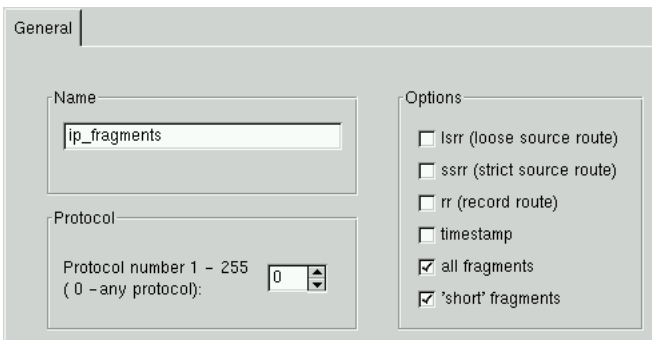



**Figure 9-4. IP Service object which represents fragmented packets.**

Object "ip_fragments" is included in the section "Services/IP" of the standard objects database coming with Firewall Builder.

Another potentially harmful type of packets is so called "Christmas tree" packet. This one is just a TCP packet with impossible combination of TCP flags or even all TCP flags turned on at once (for example SYN, ACK, FIN, RST, PSH). This combination is never used in real communications, so if a packet like that appears at the boundary of your network, it should be considered illegal and blocked. Object "tcp-xmas" is included in the section "Services/TCP" of the standard objects database coming with Firewall Builder.

Some platforms provide a mechanism to turn on and off stateful inspection on individual rules. Turning it off on those rules which do not require it may improve performance of the firewall. Obviously we do not need stateful inspection while analysing fragmented packets as we do not really want any session to be established, so we can safely use this option on this rule. One example of firewall platform which supports stateful inspection but provides a way to turn it on and off is iptables. In Firewall Builder, this can be done in the rule options dialog (which is platform-sensitive and shows different options for different platforms). Figure 9-5 shows rule options dialog for iptables firewall:

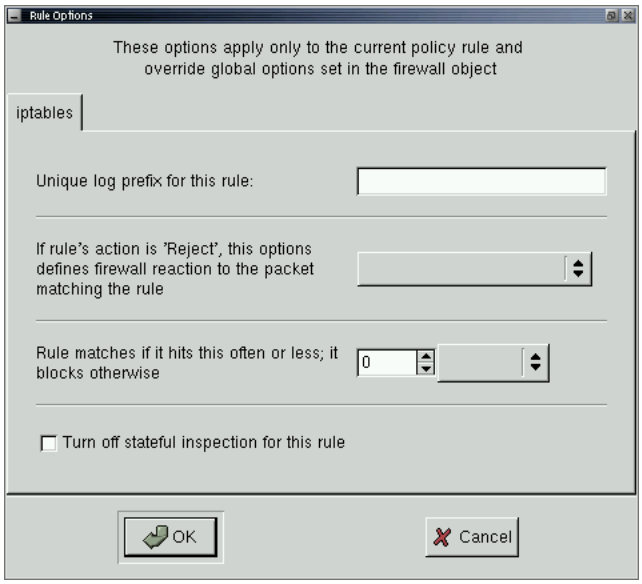



**Figure 9-5. Rule options dialog for iptables firewall**

Here is an example of the policy rule which is intended to block all fragmented and TCP "Christmas tree" packets. Note an icon in the "Options" column, it indicates that this rule has non-default options set.

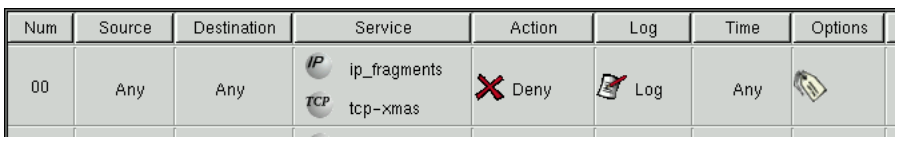



**Figure 9-6. Rule blocking all fragmented packets and TCP "christmas tree" packets**

This rule is part of the Global Policy and therefore applies to all packets crossing the firewall regardless of their origin. This means that it will block such packets originating in your network, too. If by some reason you might want to be able to send this kind of packets out, then put this rule in the Interface Policy on your external interface, with Direction "Inbound".

An interactive Druid available via menu item "Rules/Help me build firewall policy" can create a rule to protect against fragmented packets.

## 9.6. Anti-spoofing rules

Generally speaking, IP spoofing is a technique of generating IP packets with source address that belongs to someone else. Spoofing creates danger when hosts on the LAN permit access to their resources and services to trusted hosts by checking source IP of the packets. Using spoofing, intruder can fake source address of his packets and make them look like those originated on the trusted hosts. Basic idea of anti-spoofing protection is to create a firewall rule assigned to the external interface of the firewall to examine source address of all packets crossing that interface coming from outside. If the address belongs to internal network or firewall itself, packet should be dropped. Since rule must take into account both the interface packet is coming through and the direction in which it is coming, it must be put into Interface Policy. This is one of the rare cases when the rule can not be put into Global Policy.

To create anti-spoofing rule in Firewall Builder we need objects for all subnets inside the firewall perimeter and obviously an object for firewall itself. Objects for internal subnets can be put in the single group, to make

rule compact. Then we create a rule in the Interface Policy of firewall's external interface (interface eth1 on Figure 9-7) and put all these objects, or a group which contains them, plus the firewall object in the "source" rule element. "Destination" and "service" must be set to "any" and Direction must be set to "inbound". Figure 9-7 shows the anti-spoofing rule.

Section 5.9.1 explains how firewall object and its interfaces can be created. Section 5.12 has more details on the firewall's interfaces, their types and other properties. Section 8.1.2 explains concept of the direction.

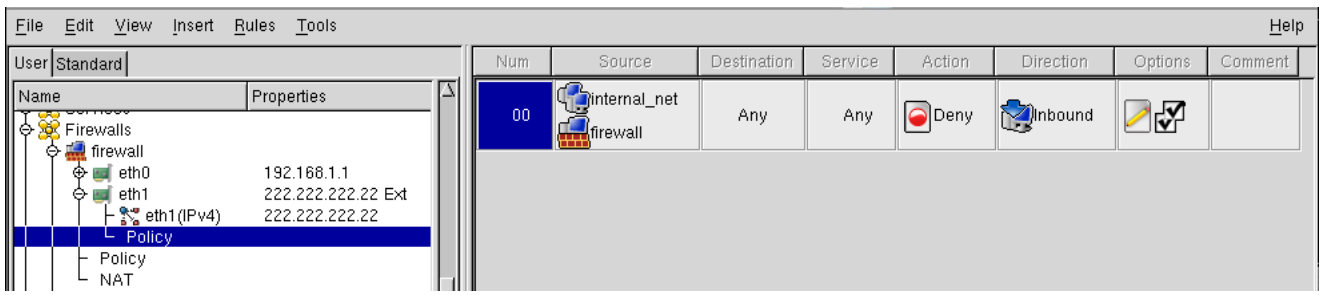



**Figure 9-7. Anti-spoofing rule**

Just like in the previous example, it may be advantageous to turn stateful inspection off on anti-spoofing rules. Since we want to prevent these packets from entering our network, there is going to be no session opened and we do not want to keep state at all. You can use "Options" element to do it.

An interactive Druid available via menu item "Rules/Help me build firewall policy" can create anti-spoofing rules.

## 9.7. Protecting local host

This is the kind of policy you would want if you are using firewall software running on the host itself. Sometimes your host may obtain its IP address dynamically using DHCP or PPPoE protocol; Firewall Builder can handle this situation as well.

- create firewall object, do not forget to choose firewall platform and OS it is running on and add its interfaces, including loopback interface. If host gets its IP address dynamically, mark interface as 'dynamic' using appropriate checkbox. See Section 5.9.1.
- call Druid, choose "Firewall protects local host" and then pick rules you want.

See what Druid have created for you. You can edit these and add more rules now.

## 9.8. Using Action 'Reject': blocking Ident protocol

Suppose we want to block connections to certain ports on the server behind the firewall, but want to do it in a "polite" manner, that is somehow let sender host know right away that the connection attempt was blocked so our server would look like nothing is listening on that port at all. One of the practical applications of this setup would be blocking Ident connections to a mail relay or a mail server. Sendmail and many other MTA's (Mail Transport Agents) attempt to connect to Ident port (TCP port 113) on the mail relay every time they accept email from that relay. Many believe Ident protocol is practically useless and does not really serve as a protection against SPAM or for any other useful purpose. Unfortunately silent blocking of ident connections on the firewall using a rule with action *"Deny"* adds a delay in the email delivery. This happens because when sender host tries to establish Ident connection to the recipient, it sends TCP SYN packet to it (the first packet in three-way TCP handshake) and then waits for TCP ACK packet in responce. However, it never sees it because recipient's firewall blocked its first TCP SYN packet. In the situation like this sender host assumes the reply

packet got lost and tries to send TCP SYN packet again. It repeats this for a few seconds (usually 30 sec) before it gives up. This adds 30 sec delay to email delivery. Our intent is to show how one can construct a policy rule to block Ident without causing this delay.

The simplest way to block any protocol is to use "Deny" action in the policy rule. Since "Deny" causes firewall to silently drop the packet, sender never knows what happened to it and keeps waiting for responce. To avoid this delay we will set rule Action to "Reject". Normally "Reject" makes firewall to send ICMP "unreachable" message back to sender, thus indicating that access to requested port is denied by the firewall. This may be insufficient in some cases because host trying to connect to our Ident port won't understand this type of ICMP message and will keep trying. In fact, most OS do not recognize ICMP "administratively prohibited" message and do keep trying. To make host on the other side stop its attempts right away we need to send TCP RST packet back instead of ICMP message. This can be done by adding appropriate rule option (see Figure 9-9) It is also safe to turn stateful inspection off on this rule since we do not want connection to be established and therefore do not need to keep track of it.




**Figure 9-8. Using action "Reject" with rule option**

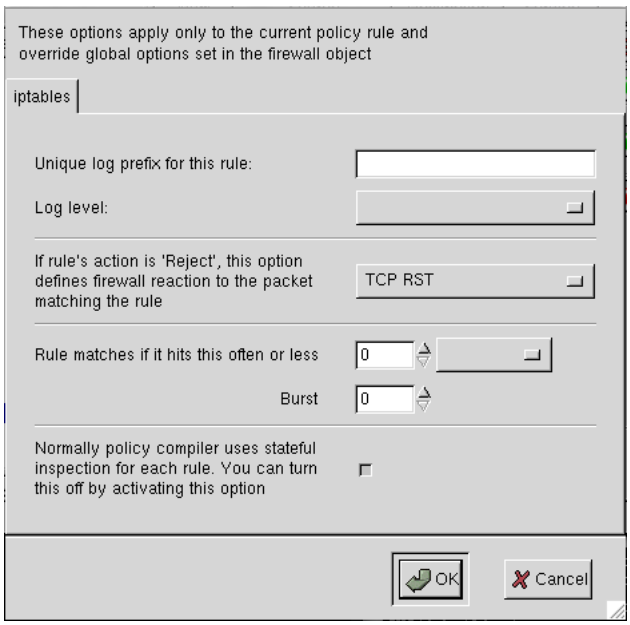



**Figure 9-9. Adding rule option to make send TCP RST packet**

## 9.9. Using negation in policy rules

Suppose we want to set up a rule to permit access from the host on DMZ net "mail_relay_1" to hosts on the Internet, but do not want to open access from it to machines on our internal network represented by the object "internal_net". Since we want it to connect to hosts on the Internet and can not predict their addresses, we have to use "any" as a destination in the policy rule. Unfortunately "any" includes our internal net as well, which is going to open undesired hole in the firewall.

There are two solutions to this problem. First, we can use two rules: first will deny access from "mail_relay_1" to "internal_net" and the second will permit access from "mail_relay_1" to "any". Since rules are consulted in the order they are specified in the policy, access to internal net will be blocked by the first rule since the packet would hit it first. These two rules are represented on Figure 9-10




**Figure 9-10. Using two rules to block access from DMZ to internal net and permit access to the Internet**

Another solution uses negation. We can specify destination in the rule as "not internal_net", thus permitting access to anything but "internal_net". Negation can be enabled and disabled in the pop-up menu which you call by clicking right mouse button on the corresponding rule field. This rule depends on the rules below it to block access from "mail_relay1" to the "internal_net". If the policy was built using general principle of blocking everything and then enabling only types of connections that must be permitted, then it usually has a "catch-all" rule at the bottom that blocks everything. This last rule is going to deny connections from the "mail_relay1" to "internal_net".




**Figure 9-11. Using rule with negation to block access from DMZ to internal net and permit access to the Internet**

Negation can be used in NAT rules in a similar way.

# Chapter 10. Examples of Network Address Translation Rules

## 10.1. Providing Internet connection for workstations behind the firewall

Suppose your internal network is configured using so-called private IP addresses as defined in RFC 1918. These addresses are:

- 10.0.0.0 / 255.0.0.0
- 172.16.0.0 / 255.240.0.0
- 192.168.0.0 / 255.255.0.0

If your network uses addresses from one of these blocks, and you want to provide Internet access for machines on this net, then you may need Network Address Translation (NAT) configured on the firewall. This particular type of NAT configuration is often called Masquerading. When configured this way, the firewall rewrites source IP address of each packet sent by internal machines to the Internet, replacing private IP address with the address of its external interface.

In Firewall Builder this type of the NAT rule is composed as shown on Figure 10-1




**Figure 10-1. Masquerading rule (this rule is part of NAT ruleset)**

Here, object representing internal network is placed in "Original Source" and firewall's object is placed in "Translated Source", thus indicating that we want source address of the packets to be translated. As before, we do not need to worry about reply packets because underlying firewall software keeps track of translations done for all the connections opened through the firewall and rewrites addresses in all reply packets automatically.

Figure 10-1 uses firewall object in Translated Source, which means firewall will substitute a source address in the packet for the address of its external interface. But what if the firewall has two or more external interfaces ? This happens if it has multiple connections to the Internet or other insecure subnets like DMZ. Generally speaking, Masquerading rule translates source address of packets that go from the more secure subnet to the less secure one. See Section 5.12 for detailed explanation of Security Levels of firewall's interfaces. If the firewall has multiple interfaces and firewall object is used in the Translated Source of the NAT rule like in Figure 10-1, then the firewall will translate source address into address of the interface through which the packet is going to exit (which is dictated by the routing table). Firewall *"fw2"* on Figure 10-2 has two insecure interfaces: *"eth1"* and *"eth2"*, both use routable IP addresses. NAT rule #0 will automatically translate source address of outgoing packets into the address of either eth1 or eth2, depending on the routing which dictates through which interface the packet is going to exit.

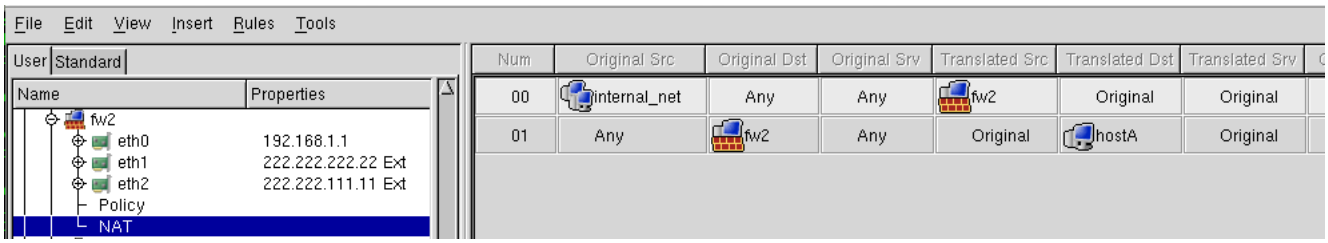



**Figure 10-2. NAT rule for the firewall with two external interfaces**

There may be situations when we would not want to automatically use all external interfaces of the firewall for translation. Suppose firewall *"fw2"* has two insecure interfaces, one of which connects it to the Internet, while another one is connected to DMZ subnet which is using private IP addresses on the net 192.168.2.0. We want to do the same masquerading translation for all packets going from internal network to the Internet as before, but do not need to translate packets going from the internal net to DMZ. Figure 10-3 illustrates this configuration.

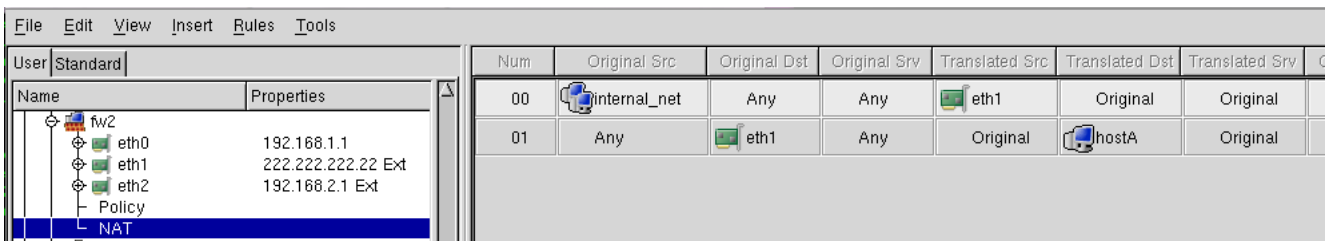



**Figure 10-3. Translation using only one interface of the firewall**

Rule #0 in Figure 10-3 uses firewall's interface *"eth1"* for translation. This means that only packets exiting through this interface will be translated; packets exiting through *"eth2"* will keep their original source address on the internal net.

> **Note:** Interface object can be used in the NAT rules even if the address of this interface is obtained dynamically and is not known beforehand.

The rules on Figure 10-1 , Figure 10-2 and Figure 10-3 provide address translation for packets, but to permit connections to the Internet a special Global Policy rule is needed. This rule simply permits all connections from internal network to "any" and should be placed somewhere close to the end of the policy, for example right on top of "catch all and log" rule.




**Figure 10-4. Rule #3 permits connections from internal network to the Internet (this rule is part of Global Policy)**

## 10.2. Server behind the firewall using address of the firewall for access

Suppose we have a network using private IP addresses behind the firewall, and a web server on it. We need to provide access to this server from the Internet in a such way that connections will be established to the address of the firewall. In this case we need destination address of packets to be rewritten so packets would reach the server on internal network. Just like in the previous case we need one NAT rule to provide translation and one Global Policy rule to permit connecitons. Figure 10-5 shows NAT rule we need:

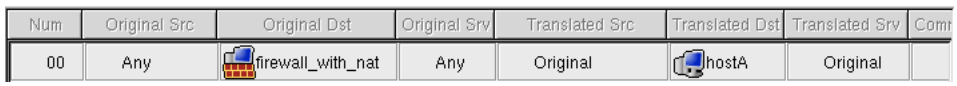



**Figure 10-5. NAT rule for the server behind the firewall**

Since we wanted to use address of the firewall's external interface to access the server, firewall's object is placed in "Original Destination" part of the rule. Server's object was placed in "Translated Destination" part of the rule so that original destination address of the packet will be replaced with server's address.

Just like with masquerading rules, firewall with multiple external (or insecure) interfaces automatically uses all of them for this type of the NAT rule. Which one will actually be used for a particular packet depends on the routing that dictates which interface the packet enters the firewall through. Firewall *"fw2"* on Figure 10-6 has two external interfaces: *"eth1"* and *"eth2"*, both use routable IP addresses. NAT rule #1 provides translation for packets entering the firewall through either eth1 or eth2; in both cases destination address of the packet is going to be rewritten and packet will be sent to the server *"hostA"*.

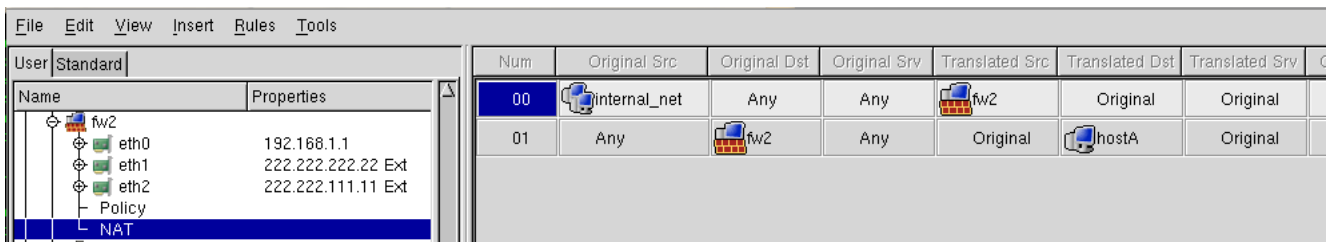



**Figure 10-6. NAT rule for the server behind the firewall which has multiple insecure interfaces**

This may not have the desired effect in certain configurations though. One such case is when the second insecure interface*"eth2"* of the firewall is connected to a DMZ subnet using private addresses. The translation is not necessary for the packets going from DMZ to internal net and we are avoiding it by using firewall's external interface *"eth1"* instead of the firewall's object in the NAT rule #1 as shown in Figure 10-7.

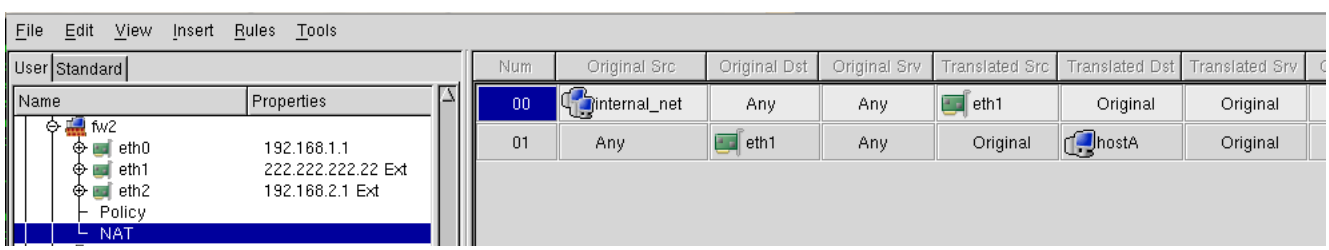



**Figure 10-7. NAT rule for the server behind the firewall, using interface object in the NAT rule**

As before, we also need a Global Policy rule to permit connections to the server. This rule is shown on Figure 10-8




**Figure 10-8. rule #0 permits access to the server (This rule is part of Global Policy)**

Figure 10-7 also illustrates how configuration Figure 10-5 could be combined with configuration Figure 10-1. Such combination allows machines on the local network to go out to the Internet, and at the same time users on the Internet can access server located on the internal network - all using single IP address of the firewall's external interface! This configuration can be useful for those with cable or DSL connection and single IP address.

## 10.3. Server behind the firewall using virtual address for access

Those who have more than one IP address allocated by their ISP may want to configure the firewall to map extra addresses to different servers inside their network. This can be done using slightly different NAT rule.

First of all, we add virtual addresses to appropriate interface of the firewall. See Section 4.2 and Section 5.4 for an explanation on how address can be added to the interface. Since our goal is to use these additional IP addresses for NAT, we will add them to the external interface of the firewall. Figure 10-9 shows firewall *fw2* with external interface *eth1* that has two IP addresses: 192.0.2.22 and 192.2.0.44. These two addresses are represented by corresponding Address objects with names *fw2:eth1(ip1)* and *fw2:eth1(ip2)*. The choice of the name for the Address object is completely up to administrator, although we usually recommend our users choose descriptive names which can be easily recognized when objects are used in the policy and NAT rules.

> **Note:** Firewall script created by Firewall Builder adds virtual addresses created in the GUI to the real interfaces of the firewall machine. This means that you do not need to configure these virtual addresses in two places - in the configuration of the firewall machine and in the GUI. Adding them in the GUI should be sufficient. Should you need to remove a virtual address, you just remove it in the GUI. It will be removed from the interface on the firewall machine when you run regenerated firewall script.
>
> Firewall Builder uses appopriate command line utilities to add virtual addresses to interfaces depending on the OS the firewall runs on. For example, for the firewall running on Linux it uses */sbin/ip* tool, while for the firewall running on FreeBSD or OpenBSD it uses *ifconfig*. Policy compilers that generate firewall script are aware not only of the names of tools that are available on the target OS, but also of the differences in the command line syntax specific to that OS.

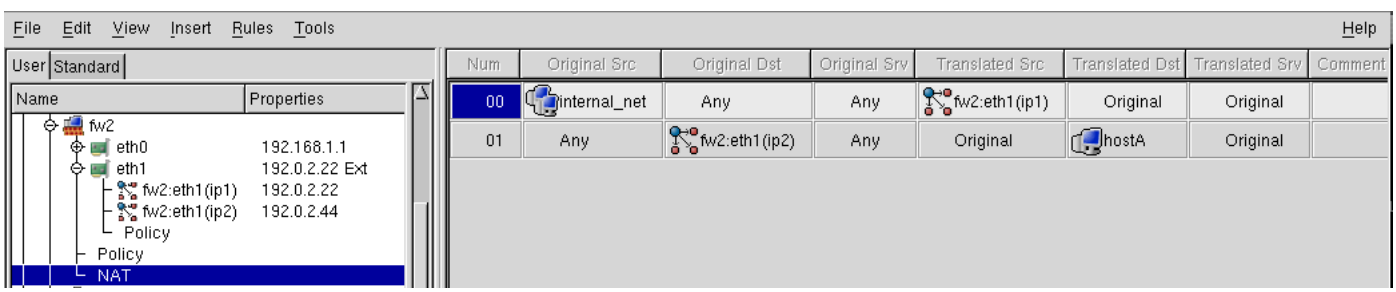



**Figure 10-9. Firewall has two IP addresses on its external interface eth1**

Figure 10-9 also shows how different addresses assigned to the firewall's external interface can be used for different purposes. NAT rule #0 uses address 192.0.2.22 represented by the object *fw2:eth1(ip1)* for masquerading translation for machines on internal network trying to access services on the Internet. NAT rule #1 uses address 192.0.2.44 represented by the object *fw2:eth1(ip2)* to provide access to the server *hostA*. These two NAT rules are just like those explained in the previous two chapters, except they use Address objects instead of the Firewall or Interface objects.

Finally we need a Global Policy rule to permit connections to the server. This rule looks exactly like that in the previous case - see Figure 10-8

## 10.4. Server behind the firewall with port mapping

This case is similar to two previous ones, except it illustrates how NAT rule can help change a port number besides of the address. This can be useful if server behind the firewall runs on some alternative port, yet we want to make it so that users would access it using another port when connect to it. For example, one could use it to provide access to the web server running on non-standard port #8080 by setting up translation of TCP ports to the standard one #80. This configuration can be combined with either one of the two above, that is translation can be done using either firewall's external interface address or some different address.

This kind of translation works for both TCP and UDP services.

First of all, additional Service object needs to be created. Since our goal is to provide access to the web server running on TCP port #8080, we need to create a TCP Object with destination port equal to 8080. The source port number does not matter in this case, therefore we leave source port setting at zero in our new object. We also do not need to set any TCP flags since we are not going to build rules matching those. Basically, all we need is an object that defines destination port 8080. See Section 4.2 and Section 7.3 for an explanation on how create and edit TCP Service object. Figure 10-10 illustrates this object's setup.

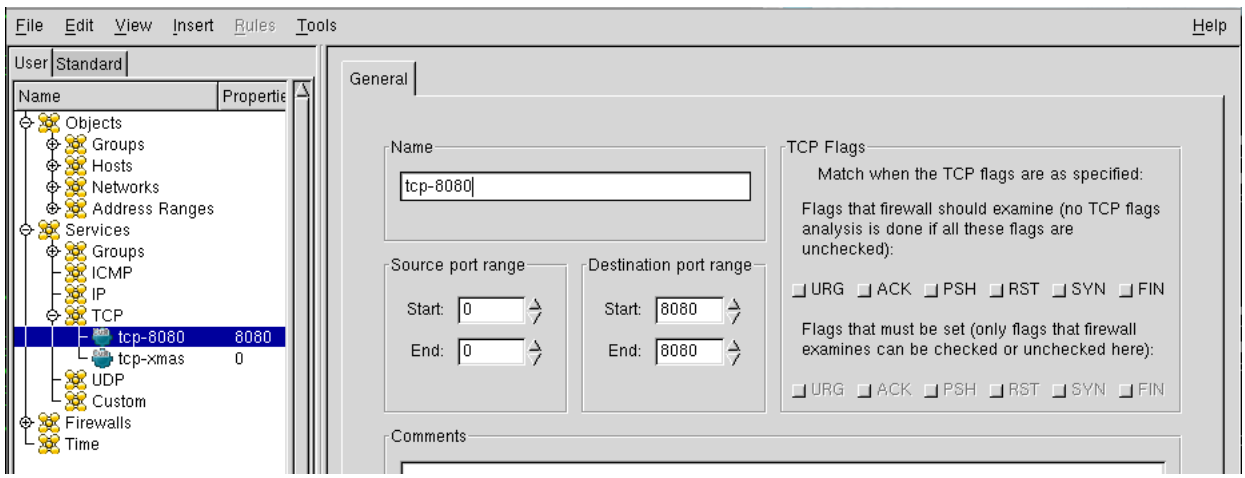



**Figure 10-10. TCP Service object for port tranlsation rule**

Now we can create the rule, it would look just like previous translation rules except we put appropriate service objects in "Original Service" and "Translated Service" columns. see Figure 10-11

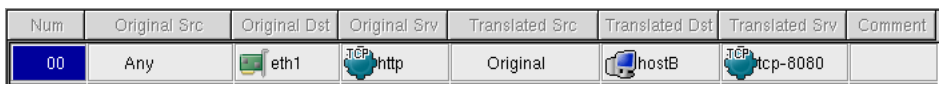



**Figure 10-11. NAT rule with port translation**

Of course we need Global Policy rule as well to permit connections to our server. As described in Section 8.1, Firewall Builder operates assuming policy rules are consulted after NAT. This means policy rule should use TCP Service object "tcp-8080" instead of "http". See Figure 10-12

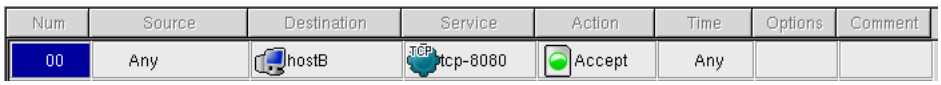



**Figure 10-12. Global Policy rule which permits access to the server**

## 10.5. Destination NAT Onto the Same Network

This situation is described in iptables HOWTO http://www.netfilter.org/documentation/HOWTO//NAT-HOWTO.html

This problem occurs when machines on internal LAN try to access a server (let's say a web server) which is actually located on the same LAN and NAT'ed through the firewall for external access. If internal users access it by its external NAT'ed address, then they send their TCP packets through the firewall, which translates them and sends them to the server on LAN. Server, however, replies back to the clients directly, because they are on the same network. Since the reply has server's real address in the source, clients do not recognize it and connection can not be established.

To resolve this problem you need to make NAT rule replace source address of the packet with address of firewall's internal interface. This should happen in addition to the translation of the destination address described in the previous chapters. If the source address of the packet that hits the server belongs to the firewall, the server replies to it; the firewall then translates again before sending the packet back to the client. Client sees the address it expects and connection gets established.

Fortunately Firewall Builder supports this kind of a dual translation NAT rule. The rule #0 in Figure 10-13 does just that: it translates both source and destination address of the packet.

Firewall's interface *eth0* is internal and is connected to the same subnet the server *web server* belongs to. For any packet headed for any address of the firewall, TCP port 80, the rule #0 substitutes its source address with the address of interface *eth0* and its destination address with the address of *web server*. The packet reaches the server because its destination address has been changed. This also makes the server reply back to the firewall, which in turn provides reverse translation before it sends these reply packets back to client hosts.




**Figure 10-13. DNAT back to the same LAN**

Rule in Figure 10-13 replaces source address of all packets regardless of their origin. Because of this, the web server sees all connections as if they were coming from the firewall rather than from the real clients. If having real client addresses in the web server log is necessary, the scope of this rule can be narrowed by placing object representing internal network in the *Original Source*. Since the source address needs to be translated only in the connections coming from the internal net, dual translation rule should only be needed for these connections. Connections coming from the Internet can be translated as usual. Combination of rules that implement this

configuration is shown in Figure 10-14. Rule #0 does dual translation, while rule #1 does a simple destination address translation. Dual translation rule must be the first in the pair because if it weren't, another one would match connections coming from the internal net and translate destination address without chaning the source address.

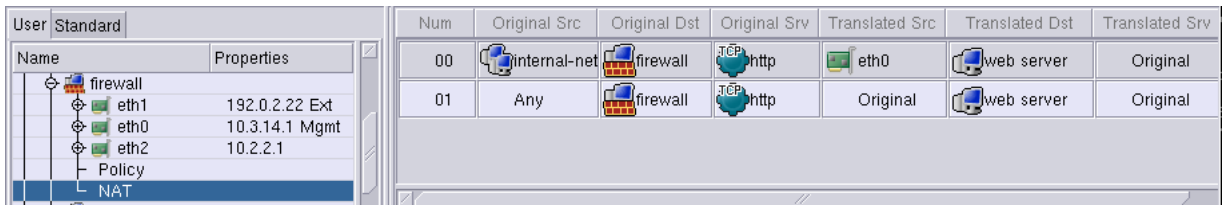



**Figure 10-14. Using dual translation only for connections coming from internal network**

**Note:** Not all firewall platforms provide features Firewall Builder needs to implement dual translation rules. Currently dual translation rules are supported only with iptables and OpenBSD PF.

## 10.6. "No NAT" rules

Sometimes we need to instruct the firewall to do NAT for certain connections, but not for others. In other words, we need to set up a rule to define an exceptions for another NAT rule. For example, suppose firewall *fw3* has three interfaces: an external interface *eth1*, internal interface *eth0* and interface *eth2* connected to a DMZ subnet. Both internal net and DMZ use private IP addresses. Our goal is to provide NAT for both internal net and DMZ to be able to establish conenctions to hosts out on the Internet, but connections between internal net and DMZ should not be translated. This can be achieved using combination of rules shown in Figure 10-15

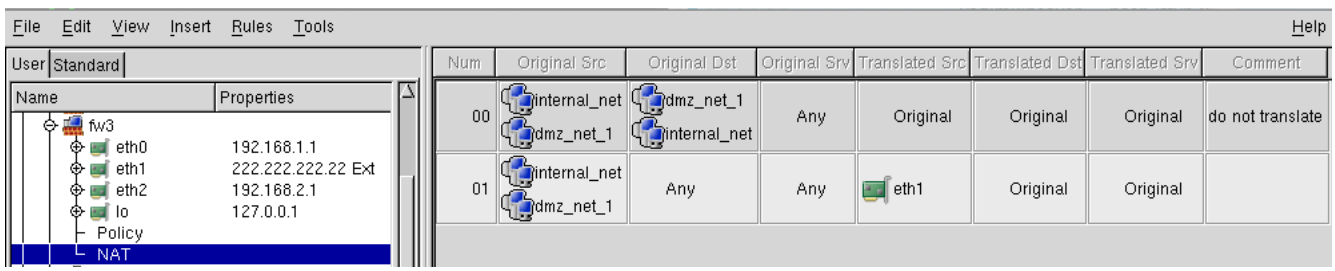



**Figure 10-15. Example of no NAT rule**

Since both *Translated Source* and *Translated Destination* in the rule #0 are left *Original*, this rule says that connections from *internal_net* to *dmz_net_1* and from *dmz_net_1* to *internal_net* should not be translated. The rule #1 is a usual masquerading type rule as described in previous chapters. The order of these two rules is important: the *Original destination* in the rule #1 is *Any*, which means connections from both *internal_net* and *dmz_net_1* to any host on any network should be translated. In order to exclude connections between *internal_net* and *dmz_net_1* from the rule #1, "no nat" rule #0 must be located above rule #1 so it would hit first.

**Note:** The same effect can be achieved by using negation in the *Original Destination*

## 10.7. Redirecting the traffic

This is a special case of NAT where packet's destination address gets changed to that of the firewall itself. This may be useful if you, for example, run transparent web proxy on the firewall box. The rule goes to NAT ruleset and should have firewall object in "Translated desination". Redirection rule may be combined with port mapping, so redirected packet will hit firewall on a different port.

In a real policy this rule would have a specific port number in the "Original Service" to limit its scope to the protocol supported by the proxy server and will also be placed on top of other NAT rules dealing with the same source addresses so it would match first. Figure 10-16 shows redirection rule which bounces all HTTP connections opened by users on "internal_net" to the Squid proxy server running on the firewall box on port 3128.

> **Note:** Squid needs to be configured as a transparent proxy or it won't work. Please see Squid documentation and FAQ for instructions.

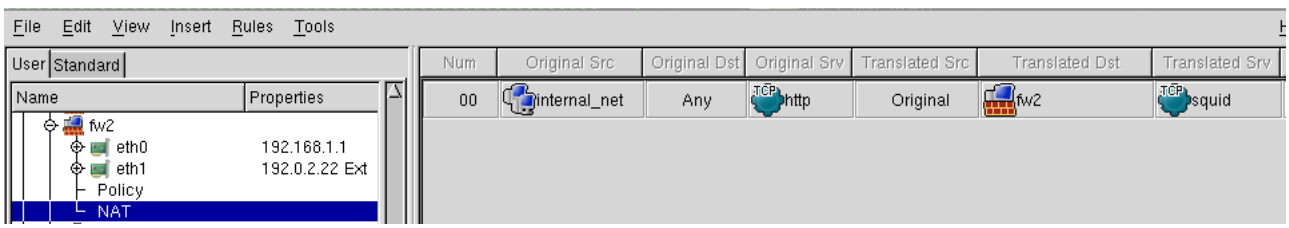



**Figure 10-16. Redirect rule**

Note that for this setup to work a Global Policy rule is needed to permit HTTP requests made by Squid. These requests will originate from the firewall itself.

## Notes

1. http://www.netfilter.org/documentation/HOWTO//NAT-HOWTO.html