Firewall Builder Architecture Overview

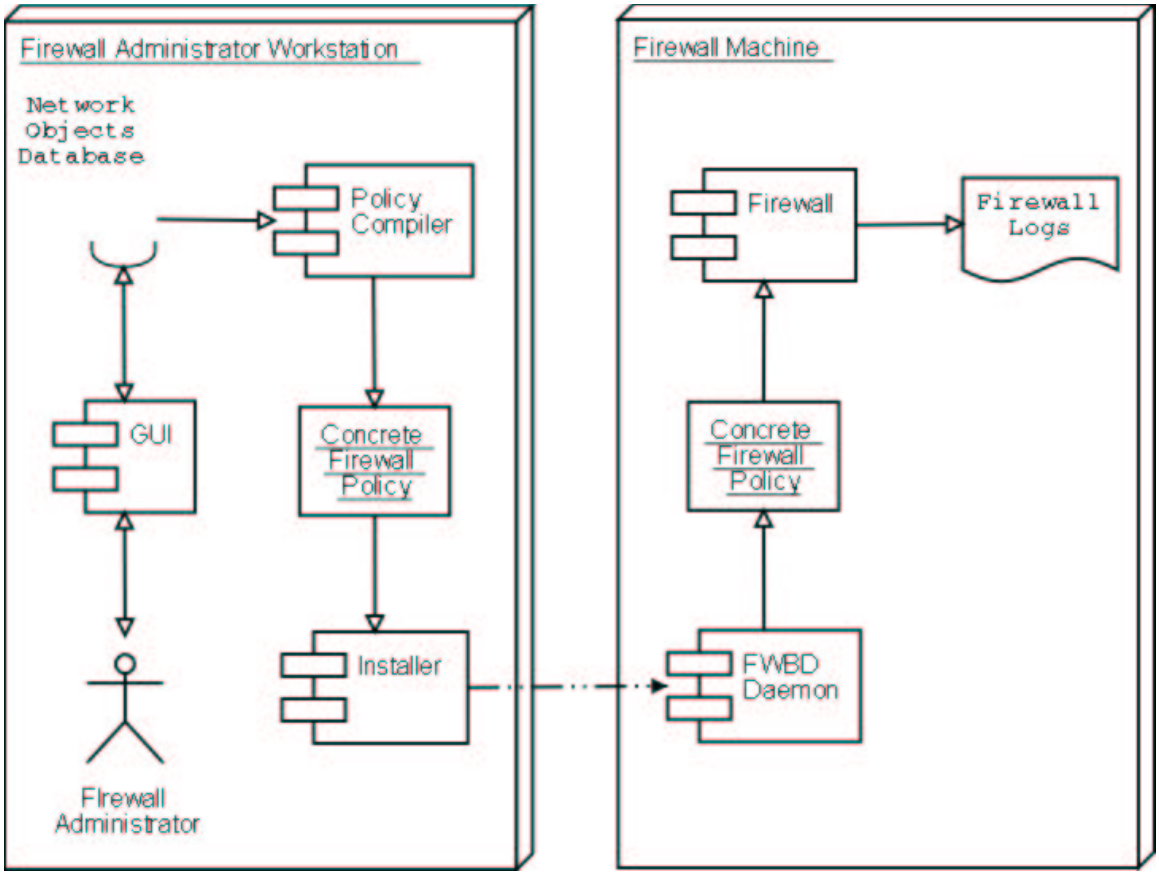
Vadim Zaliva <lord@crocodile.org>
Vadim Kurland <vadim@vk.crocodile.org>

Abstract

This document gives brief, high−level overview of existing Firewall Builder architecture.

Modules

Figure 1 shows current architecture:



The core of it is Network Object Database: centralized storage of information about network objects. It holds information about hosts, routers, networks, firewalls. For each object it stores information about its network−related properties. For example for a network these are address and netmask; for a host these are address and OS with its properties; a firewall is built on top of host and therefore includes all its properties, plus its Policy, NAT rules etc. Objects could be organized into groups. Currently Network

Object Database is stored in XML file according to Firewall Builder Extensible Markup Language (XML) Document Type Definition (DTD). In the future it is possible to map it to relational database schema.

Let us look closer at each component:

API

All access to Network Database is done via C++ API. API is packaged as independent package: '*libfwbuilder*' which could be used by third−party developers. All API classes are organized into several sub−modules, enclosed in separate namespaces:

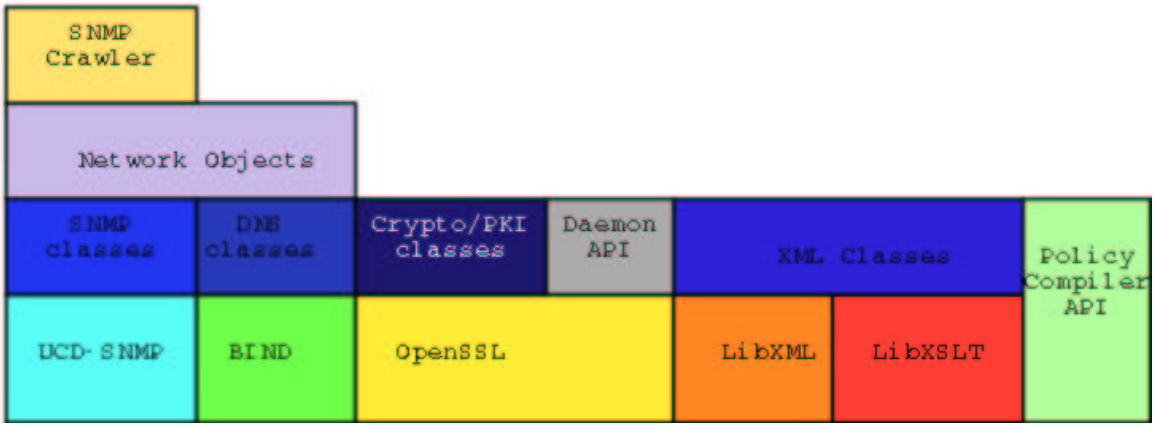
1. **fwbuilder** − Base API providing access to network database objects as well as some utility classes.
2. **fwcompiler** − Policy Compiler developer toolkit. Provides classes which allow to easily construct Policy Compilers for new platforms.
3. **fwbd** − Client library used to connect to Firewall Builder Daemon. This is C−language library, not C++.

There are several utility modules which are part of the library (currently in 'fwbuilder' module, but some might move to separate modules in the future):

1. **DNS** − This module is collection of classes which allows user to resolve hosts (single and bulk), transfer and parse zone. All operations are thread−safe. It makes use of BIND library (version 8 and 9).
2. **SNMP** − This module provides simple C++ client API. Special classes exists for several high level queries (like extracting interfaces information, ARP tables, routing tables). All queries return (when possible) information using network database objects. All operations are thread−safe. It makes use of NET−SNMP (formerly UCD−SNMP) library.
3. **SNMP Crawler** − Built on top of SNMP classes, this is sophisticated SNMP crawler. Given 'seed' host which responds to SNMP queries, it attempts to find as much hosts as possible and for each found host extracts all relevant network settings.
4. **PKI** −API contains classes for managing X.509 certificates and RSA keys. It is implemented on top of OpenSSL library.
5. **XML** − Library provides classes for loading and saving XML files. There is special facility for performing automatic data format upgrades using XSL Transformations. User normaly would not need to use it to access network database objects −we provide high−level C++ API for that. However it could be used to write other XML files, like GUI Preferences, etc. It completely hides all the details of particular XML parser interface, exporting outside DOM tree. If you are using this, you will not be affected if different parser is used in the future. Currently it is based on LIBXML and LIBXSLT libraries.

Here is diagram showing API classes organization:

Firewall Builder API



All library classes are commented and API documentation is built using **DOC++** tool.

GUI

Firewall Builder user interface is written using GTK+ toolkit. There are versions of this toolkit for most UNIX platforms (on top of Xlib) as well for Windows platforms (on top of Windows API). This allows us to run GUI on both platforms.

GUI allows to view, create, delete, edit, group network objects. Drag−and−drop technique is widely used. Objects on screen are organized into tree−like hierarchy in the left pane, with object type specific dialog appearing in the right part of the main window. User navigates objects using mouse, or keyboard shortcuts.

GUI represents firewall policy in the form of the list of rules, with one rule taking one line in the list. Each rule has few standard fields, or rule elements, such as ’Source’, ’Destination’, ’Service’, ’Action’. Rule elements ’Source’ and ’Destination’ hold references to network objects, such as hosts, networks, firewalls and groups of these, while rule element ’Service’ holds reference to a service. Each element can include more than one object. Policy compiler processing a rule like this is supposed to handle situation when multiple objects are included in the rule element and either split rule to a set of primitive rules with exactly one object in each element, or, if target firewall platform supports grouping in its native syntax, generate proper code with multiple references.

User can manipulate rules in the policy simply by dragging and dropping objects from the tree into rule elements. Standard Copy/Paste mechanism for objects is also supported. User can also add rules, delete rules, insert new rule above or below selected one, copy and paste rules in the same or between different policies.

Each rule can have some platform−specific options associated with it. Each rule that have non−default set of options is marked with a special icon in a rule element 'Options'. Such options can alter the behavior of particular rule using features which are available only on particular firewall platform. For example, one of the popular open source implementations called 'iptables' (available on Linux) supports custom logging prefix. This prefix is just a user−defined text string that shows up in the log record when firewall encounters a packet that matches a rule with this option. As far as we know, this feature is available only in iptables, therefore we did not want to include it in a generalized firewall configuration XML DTD. However we wanted to provide support for it as it proves to be very useful in debugging and automated log processing. In Firewall Builder this feature can be used via platform−specific policy rule options dialog.

GUI supports two types of firewall policy: global policy and interface policies. The former represents set of policy rules which are not associated with any interface. Interface policy is always associated with one of the firewall's interfaces. There are firewall platforms which require rules to be always associated with interfaces, these are for example Cisco routers ACLs, Cisco PIX. Some other platforms can handle rules within or without interfaces, these are iptables, OpenBSD pf and some others both commercial and free firewalls. We believe that designing a policy regardless of firewall interfaces makes it more readable and easier to understand, however there are certain cases when information about interface must be available. One such case is anti−spoofing rules, where firewall is supposed to block packets that have source address belonging to internal network behind the firewall but that are coming from the Internet (that is, moving in inbound direction on external interface). This rule can not be built without knowledge about interface and direction in which packet is coming, therefore this rule must be placed in policy associated with particular interface of the firewall.

Interface policies may become the only policies used if the target platform requires information about interface (Cisco routers ACLs for example), however this really depends on how clever policy compiler is. In certain cases policy compiler can either guess what interface particular rule should be placed on, or simply place it on all interfaces.

Network Address Translation (NAT) rules are organized in the similar way in yet another list called NAT. NAT rules have different set of rule elements though, these are: 'Original Source', 'Original Destination' , 'Original Service', 'Translated Source', 'Translated Destination', 'Translated Service'. First three represent parameters of IP packet before translation, which other three represent its parameters after translation. All the same operations apply to the NAT rules: user can manipulate rule elements using drag and drop or Copy/Paste mechanisms and rules can be added, deleted or rearranged.

GUI provides several interactive wizards (we call them 'Druids') to simplify certain operations. For example, such druid can be used to build initial firewall policy. This provides for a powerful combination of flexibility of policy representation in the form of a list of rules and a simplicity of druid, which asks few questions and builds a skeleton policy for standard network configurations.

GUI uses classes provided by API to call policy compiler and installer programs as a

background operations. This allows for relatively simple extension mechanism, where user could write their own policy compiler or policy installer script and use it with our GUI not having to make any changes to it.

Another important GUI function is printing. User can print either particular firewall policy or all network objects database. Printing format is customized via XSLT printing transformations. We currently provide several transformations for printing to HTML and plain text format. User can add new transformations and customize existing ones.

Data Import/Discovery

If you are planning to manage network consisting of more than pair computers using Firewall Builder, then typing in all hosts and firewalls objects along with their configuration information is tedious and error prone task. We recognize this problem and provide several tools in Firewall Builder which will allow you quickly import information about your network into Firewall Builder network objects database.

Simplest function you can use is to import */etc/hosts* file.

More sophisticated is DNS zone import. This function asks you for domain name, then queries DNS looking for primary name servers responsible for this domain. The program can either use one of these servers, or some other alternative one to pull DNZ zone information from. If zone transfer has been successful, all hots names along with IP addresses in this zone will be added to our network objects database.

Last, and by far the most sophisticated method of discovering objects is SNMP crawler. It starts from some ’seed’ host which should respond to SNMP queries. This host is queried and information about it configuration, list of interfaces, routing and ARP tables is retrieved. Once appropriate network object is created for this host, crawler proceeds to recursively query other hosts mentioned in this host ARP and routing tables. Advantage of this import mechanism is that it gathers much more information about hosts than DNS zone or hosts file import. For example it discover not only hosts, firewalls and routers, but also IP networks.

User is given a chance to review all objects found by any of import mechanisms and decide which of them he wants to import. For any discovered object, user is given a chance to specify how it have to be imported (as host, as firewall or as router).

Daemon

Once you defined and compiled your firewalls’ policy, you need to deploy it on the actual firewall. Firewall Builder provides several ways to do this. The simplest one is user defined installation script. This script could be invoked from "Install" menu item in GUI and does whatever user programmed it to do to install policy files. Typical usage is to invoke secure shell ’*scp*’ command to copy firewall script to the firewall machine, then use ’ssh’ to execute it.

We also provide more advanced method of installation, using special daemon (written by us). You can install this daemon on the firewall and connect to it from the GUI to install the policy.

Connection makes use of Transport Level Security (TLS) protocol implemented in OpenSSL library. This is the same protocol as used in HTTPS. Authentication to the daemon is done using public keys and X509 certificates. User can generate himself several certificates (or import ones generated by 3rd party Certification Authority). For each firewall he is managing he can specify which certificate has to be used. Firewall administrator puts user public key into the list of keys authorized to access this firewall. This provides secure, authenticated mechanism for remote deployment of firewall policy from Firewall Builder GUI.

Links

1. Firewall Builder home page
2. Managing XML Documents Versions and Upgrades with XSLT