A Tool for Building Firewall-Router Configurations

Christopher J. Calabrese
Novell Information Services
and Technology

ABSTRACT: Several commercial firewall solutions are currently available, but they may not be appropriate for all situations because of their expense, lack of flexibility, or lack of scalability. Instead, many firewalls are built using packet-filtering routers.

One big impediment to building router-based firewalls is how poorly the configuration languages used to describe the proper operation of the routers are suited to the job of building and maintaining robust firewalls. These languages are overly terse and do not support software-engineering techniques such as code-reuse and datahiding.

This paper describes a tool that overcomes these limitations by generating a configuration in the router's native configuration language from a high-level description of the firewall that is embedded in KORNSHELL, a popular computer language supporting code-reuse and data-hiding.

1. Introduction

This paper describes `router-config`, a software tool that translates a high-level description of a router's functions into the native configuration-language of that router. The input is a language that provides software-engineering capabilities such as code-reuse and information-hiding. These capabilities allow for significant time-savings in router configuration and maintenance, especially when used in firewall applications. Actually, the input to `router-config` is a KORNSHELL (`ksh`) program that sets variables defining the parameters of the router. This allows network administrators to describe complex configurations using a powerful and flexible language they're already familiar with. `router-config` itself is built from KORNSHELL scripts.

Using variable-settings controlled by a scripting language may seem a strange way to to control something as complex as a router, but this idea is grounded in the way the KORNSHELL itself is configured (and its predecessor the Bourne Shell), and has also been used in other successful software systems, such as EMACS [Bolsky & Korn 1989, Bourne 1994, Stallman 1995]. `router-config` can generate router configurations for the following situations:

- A non-firewall router connecting multiple *inside* networks.
- A simple firewall-router connecting one or more *inside* networks to one or more *outside* networks.
- A more complex firewall-router with *inside* network connections, *outside* network connections, and one or more screened-subnets, or DMZ's.[1]
- A two-router screened-subnet firewall architecture, where the outer router has one or more *outside* network connections, the inner router has one or more *inside* network connections, and the two routers are connected by one or more DMZ's.

1. DMZ is short for De-Militarized Zone, a term originating from the U.N.-controlled buffer between North Korea and South Korea. In this case, it refers to a screened-subnet where semi-trusted net-visible servers are placed.

The current software can generate configurations only for Cisco routers, but could be extended to handle other router types.

2. *Background and Motivation*

Back in the days when the Internet was a small, safe place, the people in Novell responsible for building firewalls "hand-built" each of our few firewalls using a packet-filtering-router architecture. Designs were based on past experiences and information available from places like Brent Chapman's `firewalls` mailing list. By mid-1994, however, several factors caused us to reconsider our approach:

- The number of IP-aware computers (especially servers) in the company was mushrooming. Originally, most non-UNIX computers in Novell used only IPX-based network protocols, but today almost all computers in Novell use both IPX and IP.
- The number of Internet and partner-net[2] connections to the company network was also mushrooming. At that time, we had three Internet connections and about five partner-net connections. Today we have six Internet connections and over a dozen partner-nets.
- The number of Internet break-ins being reported was also mushrooming.
- Finally, Cheswick and Bellovin published *Firewalls and Internet Security* [Cheswick & Bellovin 1994], allowing us to systematically compare our firewalls to the ones described in their book.

In response, we began a major project that eventually changed the whole way we approached firewall-construction. It started with re-thinking the underlying reasons for our firewalls, leading to a policy document specifying the exact level of security we expected when connecting our company network to any foreign network, be it the Internet, a partner-net, or the telephone network. Next we published an interpretation of how the policy applied to the details of constructing IP-firewalls. Some of the things specified in the interpretation document are:

- The type of firewall architecture to use (a one- or two-router DMZ configuration).
- What services can be accessed securely from inside the firewall.

2. A partner-net is a private network connection to another organization, usually a business partner.

- How to proxy inbound data (such as e-mail and net news).
- How to setup packet-filters to meet our requirements.

The next logical step was to encode these policy decisions into software so that engineers constructing firewalls could concentrate on the situation at hand rather than re-inventing the wheel each time. Wheel re-invention was a common problem because the router configuration languages didn't allow for code-reuse.

3. *Software Implementation*

From a language-centric standpoint, this system takes a simple input language wrapped in a powerful macro-language (KORNSHELL) and compiles it into an equally simple output language. Since the KORNSHELL software is doing the parsing and lexing, the compiler has no true front-end and operates directly from internal data representations of the language (the values in the significant shell variables). Since both the input and output languages are entirely non-procedural, the compiler also does not have many of the traditional back-end compiler features like a block-optimizer or register-allocator. Instead, it operates as follows:

- A data-driven front-end (the `router-config` script) takes the input language (which is similar in concept to the data-structures used in the back-end of a traditional compiler) and makes calls into a language-independent code-generator (`router.ksh.lib`).
- The language-independent code-generator, in turn, makes calls into a lower-level device-dependent code-generator (`cisco.ksh.lib`).
- The low-level device-dependent code-generator finally outputs the actual configuration information in the target device's own configuration language.

If you don't like the compiler analogy, `router-config` can be thought of as a program driven by a simple input language (like `dc`), `router.ksh.lib` as the C libraries and system calls used by that high-level program, and `cisco.ksh.lib` as the underlying operating-system and device-drivers.

4. *Description of the Input Language*

As stated earlier `router-config`'s input language lives in the values of a set of KORNSHELL variables. This section describes these variables and how they behave.

4.1. *Variables Describing the Physical Router Configuration*

- `routerType` The type of router this is. This determines which library is used for device-dependent code-generation. The only value currently supported is `cisco`.
- `routerHardwareModel` The version of the router hardware (e.g., `2514`, `7500`).
- `routerSoftwareVersion` The version of the software running on the router. The Cisco driver will turn off certain features if running Cisco Internetwork Operating System (Cisco IOS) versions before release 10.3. Also, it cannot handle Cisco IOS versions before release 9.21 because of their lack of support for inbound access-lists.
- `routerBootMethod` How the router boots. The Cisco driver understands `default` (flash-EPROM or ROM depending on the model router) and `network` (`tftp`).
- `routerConfigMethod` How to configure the router on boot-up. The Cisco driver understands `none`, which uses the configuration in flash-ROM if it's there, and `network`, which uses `tftp`.

4.2. *Variables Describing the Setup of the Router's Interfaces*

There is only one variable that describes the setup of the router's interfaces, `routerInterfaces`, but its value is a complex table. Each line of the table describes one interface, and each white-space-separated field describes one aspect of the interface. The fields are as follows:

- `interfaceFunction` Specifies what the interface is used for, which determines what type of default firewall-filtering will be setup for this interface. Recognized values are:
  - —`shutdown` The interface is not in use.
  - —`exteriorFirewall` An outer-most (exterior) interface of a firewall-router.
  - —`dmz` An inner-most (DMZ-facing) interface on an outer-router of a two-router DMZ firewall configuration.
  - —`dmzFirewall` An outer-most (DMZ-facing) interface on an inner-router of a two-router DMZ firewall configuration.

—`interior` An inner-most (protected-network-facing) interface on the inner-most router of any firewall configuration, or any interface on a non-firewall router.

—`combinedDmzFirewall` A DMZ interface in a one-router DMZ firewall configuration.

- `interface` The physical interface being referred to. The Cisco driver passes this value directly to the '`interface`' command, with any underscores ('_') converted to spaces. Thus, it understands `ethernet_`*N*, `serial_`*N*, `fddi_`*N*, etc.
- `mediaType` What type of physical media is attached to this interface. Valid values for the Cisco driver are anything acceptable to the Cisco `media-type` command (e.g., `10BaseT`, `aui`, `fddi`, `serial`). This data is unused for Cisco models below 4000, which have fixed media-types. This, and other subsequent fields, are optional for `shutdown` interfaces.
- `address[/netMask]` The IP address [and optional net-mask] bound to the interface.
- `broadcastAddress` The broadcast address for the directly-attached network.
- `ipHelpers` List of IP-Helper addresses for doing fancy things with IP broadcasts. Use `none` or `noHelpers` to specify an empty-list. List elements are separated with '`,`'s.
- `reachableNetworks` The name of a variable containing the list of networks reachable through this interface. Use `none` or `noReachableNetworks` to specify interfaces with no reachable networks that are not direct-attached. The use of reachableNetworks variables is described below.
- `inboundFilteringRules` The name of a variable containing rules about how to setup inbound packet filtering for this interface. Use none or `noInboundFilteringRules` to specify no additional rules. The use of `inboundFilteringRules` variables is described below.
- `outboundFilteringRules` Similar to `inboundFilteringRules`, but for outbound packet filtering. Use none or `noOutboundFilteringRules` to specify no additional rules. The use of `outboundFilteringRules` variables is described below.

4.3. *Variables Describing Routing*

- `routingProtocols` Blank-separated list of protocols supported by the router. The Cisco driver passes this value directly to the '`router`' command, with any underscores ('_') converted to spaces. Thus, it understands `bgp_`*autonomous-system*, `egp_`*autonomous-system*, `egp_0`, `igrp_`*autonomous-system*, `isis_`*tag*, `iso-igrp_`*tag*, `ospf_`*ospf-process-id*, and `rip`.
- `routingNetworks` Blank-separated list of networks to exchange routing data with.
- `reachableNetworks` variables. These are the variables defined by the `reachableNetworks` fields of the `routerInterfaces` table. This information is used both for routing and default packet-filtering. These variables contain a list of network addresses and masks, as well as routing information for these networks. Like `routerInterfaces` itself, this data is in the form of a table. In this case, each line refers to one network, and the fields are:
  - —`netAddress[/netmask]` Network address [and mask] for this network.
  - —`routerAddress` Name/address of the router to which to pass data for this network.
  - —`routingCost` The cost of getting there. In hops, or OSPF cost if using OSPF.

4.4. *Variables Describing Packet Filtering Rules*

By default, the software generates a configuration that denies any packets coming from or going to *illegal* places (e.g., packets coming from the outside with *inside* source-addresses) and otherwise allows:

- All outbound[3] packets.
- All inbound[4] TCP packets from established connections (i.e., the TCP `ACK` bit is set).

3. Outbound packets are those traveling from inside the firewall to the DMZ, from inside the firewall to outside the firewall, or from the DMZ to outside the firewall.
4. Inbound packets are those traveling from outside the firewall to the DMZ, from outside the firewall to inside the firewall, or from the DMZ to inside the firewall.

These defaults can be supplanted and/or superseded through the variables specified by the `inboundFilteringRules` and `outboundFilteringRules` fields of the `routerInterfaces` table. The values of these variables are tables with one line per rule, each of which contains the following fields:

- `type` Whether this rule defines packets that will be allowed (`permit`) or disallowed (`deny`).
- `fromAddress[/fromMask]` IP address[/mask] of the machine/network the rule allows/denies access *from* (e.g., `Inet-Mail.mycompany.com`).
- `toAddress[/toMask]` IP address[/mask] of the machine/network the rule allows/denies access *to* (e.g., `mail.mycompany.com`).
- `protocol` The protocol of the packets the rule refers to. Valid values are `ip`, `icmp`, `tcp`, and `udp`.
- `operator` Optional port-number operator for `tcp` and `udp` rules. Valid values are `lt`, `gt`, `le`, `ge`, `eq`, and `ne`. If supplied, the `port` field must also be supplied.
- `port` Optional port-number for `tcp` and `udp` rules. If supplied, the `operator` field must also be supplied.
- `established` Optional keyword-flag to indicate that `tcp` packets are for an established connection (i.e., they have the `ACK` bit set).
- `ICMP-Message-Type` The message type for ICMP messages. Valid values are shown in Table 1.

4.5. Variables Describing Router Administration

- `hostName` Name of the router (e.g., `Inet-Router`).
- `routerPassword` Password used for console/`telnet` access (e.g., `Th1s1sTh3Passwd`).
- `routerEnablePassword` Password enabling access to privileged commands (e.g., `Th1s1sTh3SUPasswd`).
- `domainName` DNS domain (e.g., `mycompany.com`).
- `dnsServers` Blank-separated list of DNS servers (e.g., '`ns.mycompany.com ns2.mycompany.com`').

Table 1.

Valid ICMP Message Types and their Representations		
Message Type	Number	Text
Echo Reply	0	echo_reply
Destination Unreachable	3	destination_unreachable
Source Quench	4	source_quench
Redirect	5	redirect
Echo Request	8	echo_request
Router Advertisement	9	router_advertisement
Router Solicitation	10	router_solicitation
Time Exceeded	11	time_exceeded
Parameter Problem	12	parameter_problem
Timestamp Request	13	timestamp_request
Timestamp Reply	14	timestamp_reply
Address Mask Request	17	address_mask_request
Address Mask Reply	18	address_mask_reply

- `trustedAdminMachines` Blank-separated list of addresses[/masks] that will be trusted for administrative access to the router (i.e., through `telnet`).
- `loghost` Host to send `syslog` output to (e.g., `loghost.mycompany.com`).
- `snmpContact` Contact string for SNMP (e.g., security-`admin@mycompany.com`).
- `snmpLocation` Location string for SNMP (e.g., '`My Company, Anytown, USA`').
- `snmpRwCommunity` Community string for read/write SNMP access (e.g., `router-community`).
- `snmpRoCommunity` Community string for read-only SNMP access.
- `snmpTrapCommunity` Community string to send when raising SNMP traps.
- `snmpRwMachines` Blank-separated list of addresses[/nets] to allow read/write SNMP access from (e.g., '`snmp1.mycompany.com/host snmp2.mycompany.com`').
- `snmpRoMachines` Blank-separated list of addresses[/nets] to allow read-only SNMP access from.
- `snmpTrapMachines` Machines to send SNMP traps to.

Table 2.

Network Mask Equivalents			
Dotted-Decimal	Hex	bits	Other
0.0.0.0	00000000	0-bits	exactMask
128.0.0.0	80000000	1-bit	
192.0.0.0	C0000000	2-bits	
...	...	...	...
255.0.0.0	FF000000	16-bits	classA
...	...	...	...
255.255.0.0	FFFF0000	32-bits	classB
...	...	...	...
255.255.255.0	FFFFFF00	24-bits	classC
...	...	...	...
255.255.255.255	FFFFFFFF	32-bits	allMask, anyMask

4.6. Notes on Net-Masks, Machine Names/Addresses, and Case-Sensitivity

- Although the software stores all net-masks natively as dotted-decimal notation (e.g. `255.255.0.0`), the equivalents shown in Table 2 are also recognized.
- The software will freely convert machine names into IP addresses.
- The software generally makes case-insensitive comparisons of variable values. Examples in this paper are shown in mixed case for improved readability.

5. An Example Configuration

The best way to understand this tool is to use it, so let's build a router configuration script from the ground up to see what it's made of.

In our example, we'll be building an Internet firewall router for `small.com`'s primary site. The architecture will be a single router with a screened-subnet/DMZ for the Internet-visible machines. We will give them a single Internet-visible e-mail gateway, a couple of internal mail-hubs, and a single DNS server inside the firewall.

```
#
# Description of the Physical Router
#
routerType=cisco
routerSoftwareVersion=10.3
routerHardwareModel=2514
routerBootMethod=default
#
# What the router is doing for us
#
routerInterfaces="
  exteriorFirewall Serial_0 serial \
      small.myisp.net/FFFFFFFC \
      197.4.42.173 noHelpers \
      externalNetworks \
      inboundFirewallRules \
      noOutboundFilteringRules
  Shutdown Serial_1
  interior Ethernet_0 aui \
      inet-fw.small.com/FFFFFFE0 \
      149.2.127.255 noHelpers \
      internalNetworks \
      noInboundFilteringRules \
      noOutboundFilteringRules
  combinedDmzFirewall Ethernet_1 aui \
      inet-dmz.small.com/classC \
      192.27.1.255 noHelpers \
      internalNetworks \
      dmzFirewallRules \
      noOutboundFilteringRules"
#
# How packet routing is accomplished
#
routingProtocols="rip"
routingNetworks="149.2.0.0 192.27.1.0"
internalNetworks="
  149.2.0.0 149.2.1.1 2
  178.18.0.0 149.2.1.1 4"
externalNetworks="allNets mygate.myisp.net 255"
#
# Firewall Policy
#  o Allow the Internet mail gateway to send mail
#    to the internal mail hubs.
#  o Allow access to the DNS server (we don't have
#    a split DNS).
inboundFirewallRules="
```

```
  permit allNets ns.small.com udp eq 53"
dmzFirewallRules="
  permit inet-mail.small.com mail.small.com tcp eq 25
  permit inet-mail.small.com mail.sales.small.com tcp eq 25
  $inboundFirewallRules"
#
# Misc. Administrative Information
#
hostName=Inet-Firewall
routerPassword=somePassword
routerEnablePassword=otherPasswd
domainName=small.com
dnsServers=ns.small.com
trustedAdminMachines="admin.small.com admin.sales.small.com"
loghost=loghost.small.com
snmpContact="admin@small.com"
snmpLocation="Small Corp., Anytown, USA"
snmpRwCommunity=secret
snmpRoCommunity=public
snmpTrapCommunity=secret
snmpRwMachines="$trustedAdminMachines"
snmpRoMachines="$trustedAdminMachines"
snmpTrapMachines="$loghost"
```

5.1. More Complex Configurations

This simple example shows the ability of this tool to express a complex router setup using fairly simple constructs that should look familiar to most UNIX users, but if an organization had several such configuration scripts lying around, they'd have a lot of redundant information in them. Since these *are* scripts, we can solve this problem, and others, using traditional software-development techniques like code-reuse. A set of ready-for-prime-time example configuration scripts is included with the `router-config` software distribution.

6. Is the Tool Useful?

As mentioned earlier, there are several routers out in the real world that have been configured using this tool, so we know that the system produces reasonably-correct results. But, there still is the question of whether the tool is actually a time-saver. Anecdotal evidence suggests that:

Table 3.

Firewall Maintenance Productivity	
Maintenance Activity	Savings
Add a new network to the list of those assigned to the organization.	90%
Add an entry to the list of machines to access the routers by telnet and SNMP.	67%
Add peepholes for a new internal mail hub.	67%
Add peepholes for a new external mail gateway.	50%
Add a firewall for a new Internet or partner-net connection.	80%
Add a simple partner-net connection to an existing firewall.	97%

- It's easier for UNIX admins to learn how to use this tool than to learn how to configure a Cisco router from scratch.
- One big reason is the amount of documentation you need to read to acquire each skill. The Cisco *Router Products Command Summary* is around 600 pages these days, and a full Cisco document set is several shelf-feet [Cisco 1995]. In contrast, the total documentation for this software is around 20 pages. Even if you count the example scripts and source-code to the tools, that still only adds about another 50 pages.
- Building a firewall-router configuration that embodies an appropriate level of paranoia is an exercise in redundancy and tearing your hair out worrying if you forgot anything. By placing the burden of handling the redundancy and detail-sweating onto a piece of software, the person using the software can concentrate more on the problem being solved rather than worrying about the implementation of the solution.

Finally, the simple example configuration script we developed in the last section is only 51 non-blank non-comment lines, while the Cisco configuration the tool generates from it is 171 non-comment non-blank lines. That's a savings of around 70%. Looking at Novell's production usage of these tools, the savings is more like 58%, but that's still a significant productivity leverage.

Even more significant is the productivity leverage in maintaining existing firewalls. To illustrate this, Table 3 compares the code changes for various common firewall maintenance functions. In the table, *savings* refers to the savings in lines-of-code for performing the indicated operation using this tool (with the example configuration files included with the software, not with Novell's production files, though they yield similar results) versus editing the Cisco configuration files by hand.

7. *Future Work?*

By now you probably think the system's great, but it could still be better. Here are some ways it could be modified to make it even better:

- Tool portability. Although the tools are written in KORNSHELL, they've only been used on UnixWare 2.x and Solaris 2.x systems, so they probably need some work to make them portable to other UNIX systems. They would be difficult to port to non-UNIX systems.
- Drivers for more router types. The Livingston IRX would be the most natural fit.
- Automatic firewall auditing tools. Given that a program can "understand" everything about these configurations, it should be possible to build another program that test whether the routers behave as intended.
- Real-time firewall auditing tools. Taking this one step further, it should also be possible to build a program that monitors a firewall in real-time to make sure it hasn't been breached. Actually, there are already programs that do such monitoring, so it's a question of writing a program that can take a `router-config` configuration script and generate a configuration for one of those existing programs.
- Expanded support for tweeking the low-level details of OSPF/IGRP/BGP routing.
- Support for specifying strong authentication using TACACS and/or RADIUS.
- IPv6 support.
- IPX support.

8. *Conclusions*

By applying the concepts of software-engineering and "little-languages" to the problem of building firewalls, this tool has allowed us to turn what was a complex software development and maintenance problem into a much simpler one. Also helping our cause is the fact that our little-language is embedded in a powerful and well-known macro-language (KORNSHELL), allowing us to avoid building a complex parser, lexer, or macro-expander, and allowing us to build our own software in KORNSHELL.

This software has been used to configure several routers at Novell and at least one router at Santa Cruz Operation.

Tool Availability

The router-config software referenced in this paper, including documentation and example scripts, is available from the author and from the Freebird Archive (`www.freebird.org`). The software is provided "as is" and without any expressed or implied warranties, including, without limitation, the implied warranties of merchantability and fitness for any particular purpose, though bug-reports and enhancements may be forwarded to the author.

Acknowledgments

While the software described in this paper was primarily written by the author, he could have not done it alone. The author gratefully acknowledges all the people at Novell who contributed time and knowledge to building these tools. Especially, Mike Convey (now at Teknekron Software Systems), Don DiPalma, Karl Tunnell-Braun, Philip Branche (now at Santa Cruz Operation), and Martin Sohnius.

References

1. M. I. Bolsky and D. G. Korn, *The KORNSHELL Command and Programming Language*, Prentice Hall, 1989.
2. S. R. Bourne, "An Introduction to the UNIX Shell" in *4.4 BSD User's Supplementary Documents*, O'Reilly and Associates, 1994.
3. W. R. Cheswick and S. M. Bellovin, *Firewalls and Internet Security: Repelling the Wily Hacker*, Addison-Wesley, 1994.
4. *Router Products Command Summary: Cisco Internet Operating System Release 10.3*, Cisco Systems, 1995.
5. R. M. Stallman, *GNU Emacs Manual, 11th Edition*, Free Software Foundation, June 1995.