

Running head: An Inexpensive and Versatile IDS

Building an Inexpensive and Versatile Intrusion Detection System using Snort, a  
Cable/DSL Router, and OpenWRT

David Schwartzburg

East Carolina University

12/01/2005

## An Inexpensive and Versatile NIDS2

### Abstract

An intrusion detection system can be an effective technical control in the modern world of information and network security. One option that provides for low cost NIDS sensor deployment is the use of the open source IDS software Snort in combination with a consumer grade LinkSys cable/DSL router and the open source firmware distribution OpenWrt. These three items together form a powerful yet inexpensive unit that delivers IDS, routing, firewall, wireless, and NAT functionality for use in a light-weight environment, i.e. consumer or small business deployments.

Building an Inexpensive and Versatile Intrusion Detection System using Snort, a  
Cable/DSL Router, and OpenWRT

Intrusion detection and prevention are important technical controls in the realm of information and network security that can be used to detect and defend against malicious, unsolicited, and suspicious types of network activity. There are two main categories of intrusion detection systems, host-based intrusion detection systems (HIDS) and network-based intrusion detection systems (NIDS). This paper will evaluate a possible option for an inexpensive and flexible NIDS sensor that could potentially be deployed in a consumer or small business environment. The specific goal is to determine the feasibility and effectiveness of using open source NIDS software (Snort) in conjunction with a cable/DSL router (LinkSys WRT54GS, Fig. 1) and open source firmware (OpenWRT) to create an effective NIDS sensor and networking device. Snort is open source software that is available for public use and improvement. OpenWRT, another open source software package, has been developed as an alternative to the native firmware that is installed on many cable/DSL router type devices. The only component of the three that will be used for this evaluation that requires a financial expenditure is the LinkSys cable/DSL router. This hardware device can be purchased for the modest price of approximately \$90 at an electronics store (BestBuy.com, November 19<sup>th</sup>, 2005). In this particular case, a \$20 mail-in-rebate was offered on the LinkSys device. So the total expenditure, if this experiment succeeds, for deploying a NIDS is \$70, not including the value of one's time spent working on deploying the sensor. In comparison, corporate grade IDS devices can cost significantly more. According to an article available on Security Focus, "commercially available sensors run in the \$5,000-\$20,000 area depending on vendor, bandwidth and functional capabilities" (Kinn and

Timm, 2002). If successful, this method of NIDS sensor deployment could prove to be feasible and significantly less expensive for a home user or small business than use of the typical commercial product. This could greatly benefit small businesses that have a need for protecting their networks, but do not have the finances to invest in commercial IDS costing thousands of dollars.

### Method

Three individual components are needed in order to implement this configuration. The first necessary component is a physical piece of networking hardware that can provide switch, firewall, routing, and NAT functionality. For this evaluation a LinkSys WRT54GS version 2.1 (Fig. 1) was purchased from a local electronics store. In addition to satisfying functional requirements mentioned above, which this device satisfies, it can also operate as an encrypted (WPA or WEP) enabled 802.11g wireless access point. The LinkSys WRT54GS provides a single WAN port for connection to the Internet. It also has four LAN ports for connecting wired devices on the local segment. See the bottom of Figure 1 for a rear shot of the device. Based on the OpenWRT hardware compatibility chart (OpenWRT TableOfHardware, 2005), the WRT54GS v2.1 is compatible with the OpenWRT firmware, has 8MB of flash memory, 32MB of RAM, and a 200 MHz processor. Besides the potential to function as a NIDS, the WRT54GS can also serve as the firewall and networking device for a home user or small business. The multi-purpose utility of this device saves the consumer or the business owner money that might otherwise be spent on the purchase of separate networking and NIDS equipment. The second component necessary to conduct this trial is a firmware implementation that can run applications, such as NIDS software, on the hardware device of choice. The capabilities of the native LinkSys

firmware are relatively limited and do not include any IDS compatibility or functionality. Figure 2 displays the web utility that is the sole means of administration of the native LinkSys firmware. The firmware option selected for use in this experiment is the WhiteRussian RC3 firmware Linux distribution available from the OpenWRT web site (OpenWRT Downloads, 2005). This distribution is a minimal Linux install that allows for customization through the installation of add-on packages via a package management system named ipkg. The WhiteRussian firmware provides similar functionality to that which is available via the native LinkSys firmware, yet also provides enhancements such as SSH device access, more granular configuration of the device's network interfaces, the ability to install customizable software packages, and more refined control over firewall functionality. The third and final component necessary to this experiment is a software package capable of providing IDS functionality. As free, open source software, Snort is a widely used and accepted standard for NIDS. A snort package (version 2.3.3) exists for the WhiteRussian OpenWRT distribution, and it will be used to add IDS functionality to the device.

Now that all three necessary components for this experiment have been defined, it is essential to integrate them into a single functioning IDS. The first step is to flash the WhiteRussian OpenWRT firmware onto the LinkSys WRT54GS. There are two different styles of OpenWRT installations that can be performed, squashFS and Jffs2. The main differences are in the layout and complexity of the file systems that are stored in the flash memory of the selected device. For this experiment, the Jffs2 version of the firmware will be used. It can be uploaded onto the device via the TFTP protocol, however, by default the device does not pause when powered on to wait for firmware updates, and therefore it does not easily accept TFTP

uploads during the boot process. In order for the device to accept the firmware update, it is necessary to manipulate the LinkSys through a vulnerability in the ping diagnostic utility available in its native web interface (Figs. 3, 4, and 5). LinkSys has fixed this vulnerability in recent versions of their firmware, so it may be necessary to flash the LinkSys to a down-level version of firmware prior to being able to use the ping exploit to modify the *boot\_wait* variable. The command set used to manipulate the value of the *boot\_wait* variable in the router is as follows (OpenWRT – Installing section 3.2.1, 2005):

```
;cp${IFS}*/*/nvram${IFS}/tmp/n  
*/n${IFS}set${IFS}boot_wait=on  
*/n${IFS}commit  
*/n${IFS}show>tmp/ping.log
```

Through this method, the *boot\_wait* variable in the device's NVRAM is modified to the value *on*, which will cause the LinkSys to pause for approximately three seconds when it is first powered on. This enables the device to be flashed via TFTP with the WhiteRussian OpenWRT firmware. Once *boot\_wait* has been changed to *on*, the LinkSys needs to be power cycled. In order to upload the firmware to the LinkSys, a TFTP client, such as the simply named *tftp* included in many Linux distributions, may be used. *Tftp* is used to send the binary firmware image from a separate system to the LinkSys while it is in the boot wait stage. Figure 6 shows the preparation of the necessary *tftp* commands to upload the firmware file *openwrt-wrt54gs-jffs2.bin* to the LinkSys. The *trace* option of the *tftp* command is used in order to provide a representation of data being uploaded from the TFTP client to the LinkSys device, as shown in Figure 7. This indicates that the image is being uploaded successfully to the LinkSys device.

Once the Jffs2 version of the WhiteRussian firmware image is uploaded to the LinkSys, the device needs to be power cycled twice. After power cycling, it is possible to telnet into the

LinkSys. Figure 8 captures the successful initiation of the initial telnet session to the LinkSys. Subsequent to a connection being established via telnet, the root password is set using the *passwd* command, (Fig. 9). Telnet access is disallowed after a root password is configured. As demonstrated in Figure 10, the LinkSys is only accessible via SSH from this point forward. Now that direct access to the system exists via SSH, the next essential task is configuration of the device. At this point, the network interfaces should all be configured as desired, and the DNS server should be configured in */etc/resolv.conf*. As with most Linux distributions, the configuration of the network interfaces can be viewed using the command *ifconfig*. The interfaces can also be configured using *ifconfig*, however, in order to make the configuration permanent, the changes need to be made using the *nvrnm* command. For example, the WAN interface IP is set by modifying the *wan\_ipaddr* variable using the *nvrnm* command. For more detailed information on configuration, please consult the OpenWRT configuration manual (OpenWRT Configuration, 2005). Any other optional configurations can be performed at this time, including the wireless capabilities of the LinkSys device.

The WhiteRussian firmware includes a utility named *ipkg* that is used to list, download, and install packages for OpenWRT. In Figure 11, *ipkg update* is used to update the available package list. This utility is now used to download and install the Snort package on the LinkSys. The command to perform this installation is *ipkg install snort*. The Snort IDS package is then downloaded and installed. After Snort has been successfully installed, it is necessary to locate and install Snort rules. Rules can be created manually, installed from a package, or downloaded from the Snort.org web site. For this experiment, a recent set of verified Snort rules are obtained from Snort.org after a simple registration process (Snort.org Rules, 2005). These rules are then

extracted from the tarball they come packed in and transferred to and stored on the LinkSys device. Figure 12 depicts the extraction of the rules to the Linksys. Due to the limited space available on the LinkSys, it may be necessary to unpack the tarball downloaded from Snort.org on a different system, extract only the rule content, create a new tarball, and upload this tarball to the LinkSys. The rules are stored in */etc/snort/rules*.

After Snort is installed and rules have been downloaded, it is necessary to configure Snort. The default configuration file for snort in OpenWRT is */etc/snort/snort.conf*. Inside this configuration file, it is important to modify a number of different settings. The HOME\_NET variable is a list of IP addresses or IP ranges that describe the devices and networks that are being protected by Snort. EXTERNAL\_NET defines the networks that should be considered untrusted. These would be the networks from which Snort would be detecting incidents. The default configuration lists the EXTERNAL\_NET as any network that is not listed as part of HOME\_NET. The default configuration file includes entries for DNS, SMTP, HTTP, SQL, and other servers. These should be configured as needed in a real deployment. Another important configuration option is RULE\_PATH, which determines where Snort will look to access any rules that should be utilized. The default value for this setting in the OpenWRT Snort distribution is */etc/snort/rules/*. Various types of preprocessing components can also be enabled or disabled in the configuration. These control whether or not different types of detection are used, such as port scan and defragmentation modules. The output section of the configuration file determines where alerts generated by Snort will be directed. For the purpose of this trial, the logging of alerts to the local syslog daemon are sufficient. For an actual deployment, it might be ideal to log Snort alerts to a database, such as mysql or postgres, or a syslog daemon on a remote

server that would have vastly greater storage capacity than the LinkSys device. The last major section in the Snort configuration file controls which rules files should be used in detection. It is important to tailor these rules based on the specific situation at hand. For the purposes of this experiment, a basic rule set is used, including: `local.rules`, `bad-traffic.rules`, `exploit.rules`, `scan.rules`, `icmp.rules`, and `netbios.rules`. Figure 13 is a truncated version of the Snort configuration file to be used for this trial. Comments and unused options have been removed for the sake of brevity.

Snort can be run either from the command line or started as a service from `/etc/init.d`. If Snort will be run as a service, then there is one additional file that is important to configure. For the OpenWRT distribution of Snort, the interface that Snort should evaluate traffic from, when it is run as a service, needs to be configured in the file `/etc/default/snort`. In Figure 14, the `INTERFACE` option is set, which determines on which of the LinkSys' network interfaces Snort should listen. For this experiment, Snort is configured to examine traffic on `vlan1`, which is the LinkSys' WAN interface.

To run Snort from the command line, there are a number of different options that can be utilized. These options can also be used when running Snort as a service. The `-A` option controls what alert mode Snort should use, and the `-c` option should be used to specify the configuration file. To background Snort, or run it in daemon mode, use `-D`. The home device(s) and/or network(s) can be specified on the command line using the `-h` parameter. The interface that snort should examine traffic on is specified via the `-i` option. In order to control where Snort logs alerts to, the `-l` parameter is used. Finally, the `-s` option can be used to tell Snort to log alert messages to syslog. Running Snort as a service in OpenWRT is essentially the same as running

it on the command line. The only difference is that the command line options are specified in */etc/default/snort* in the *OPTIONS* variable. To start or stop the Snort service manually the command */etc/init.d/snort (start|stop)* is used. In order to configure the Snort service to start automatically when the LinkSys is powered on, it is necessary to rename the script */etc/init.d/snort* with a prefix of *S##*, where *##* is an integer that controls in what order the service will start. This will cause the LinkSys to automatically run the Snort service upon booting. It is important to ensure that Snort runs successfully before configuring it to run at boot.

Once Snort is configured and running on the LinkSys, it is necessary to generate traffic to determine if Snort properly detects incidents originating from systems that are not specified in *HOME\_NET*. In order to carry out this activity, the free tool *nmap* is used to create port scans from what Snort considers to be an external network. A basic port scan is initiated against the LinkSys' WAN interface using the command: *nmap 192.168.44.110*. In order for Snort to have the opportunity to detect the traffic and generate an alert, it is necessary that the scan not be blocked by *iptables*, the firewall that is used by default with OpenWRT. Therefore, an additional rule is added to the *INPUT* chain of the firewall that allows all traffic that has a source address of the system that is to originate the port scan. Figure 15 shows a listing of the firewall rules in place by performing the *iptables --list* command on the LinkSys. Rule three in the *INPUT* chain has been added to allow traffic from the scanning system, 192.168.44.100.

### Results

Snort will run successfully on the LinkSys using OpenWRT. Figure 16 shows a listing of all the processes running on the LinkSys, produced using the *ps* command. Snort is running as

PID 799 and is using approximately 20MB of memory. It is necessary to carefully choose which rule sets are configured for use. The LinkSys WRT54GS has a limited amount of resources, especially RAM (32MB), and running Snort with too many configured rule sets can easily use all available RAM. This can lead to crashing the LinkSys or causing the Snort process to terminate unexpectedly. This happened once or twice in initial runs of this experiment when too many rule sets were selected for use in the snort configuration. As previously mentioned, it is important to iron this out prior to setting Snort to run at boot, otherwise it could continuously crash the LinkSys every time that it boots. The user should experiment with different configurations to determine the optimal situation for maximizing the number of useful rule sets that can be utilized without completely depleting all of the LinkSys' resources. In Figure 17, the command *top* is used to show the resource utilization on the LinkSys while Snort is running. By eliminating rule sets based on types of traffic that will always be blocked at the firewall, it is possible to reduce the amount of resources that are used by the Snort process on the LinkSys. In addition to limited memory resources, the amount of storage space on the LinkSys is also very limited. The total storage capacity is 8MB of flash memory. The Snort rules take up approximately 2MB, and after accounting for other software, there is 3.6MB of flash storage left with a base install of OpenWRT.

After running the port scan from a system against the WAN interface of the LinkSys, Snort generated a port scan alert and logged it to the LinkSys syslog daemon. Figure 18 shows the syslog, which includes the initialization of Snort, as it was run as a daemon from */etc/init.d*, and the alert generated by Snort.

### Conclusion

The NIDS deployment used in this study proved to be low cost and effective. There are a few considerations that the user should keep in mind. For permanent deployment of a LinkSys as a NIDS, it is important that any alerts and logging generated by Snort be stored on a system other than the LinkSys, due to the limited storage resources available. It is also important that the Snort rules selected to be run be minimized as much as possible in order to prevent over-utilization of the memory available on the LinkSys. Additional testing would be required in order to see how the LinkSys would fair running Snort in a production environment. For this experiment, no stress testing was performed to determine what types of load the device could sustain and still continue to function properly. For consumer and small business use, a LinkSys is a practical and effective alternative to more costly commercial-grade IDS equipment. The sub-one-hundred dollar price tag for a NIDS sensor with routing, firewall, NAT, and wireless capabilities is an appealing option that should not be quickly overlooked.

## References

BestBuy.com. Linksys SpeedBooster 802.11g Wireless Broadband Router, <http://www.bestbuy.com/site/olspage.jsp?skuId=6304082&type=product&cmp=++&id=1074787298555>. Viewed on November 19<sup>th</sup>, 2005.

Kinn, David, and Timm, Kevin. Justifying the Expense of IDS, Part One: An Overview of ROIs for IDS. <http://online.securityfocus.com/infocus/1608>, July 18, 2002.

OpenWRT.org Configuration. <http://wiki.openwrt.org/OpenWrtDocs/Configuration>, referenced on November 19<sup>th</sup>, 2005.

OpenWRT.org Downloads, openwrt-wrt54gs-jffs2.bin. Downloaded from <http://downloads.openwrt.org/whiterussian/rc3/bin/> November 19<sup>th</sup>, 2005.

OpenWRT.org Installing – OpenWRT. <http://wiki.openwrt.org/OpenWrtDocs/Installing>, referenced on November 19<sup>th</sup>, 2005.

OpenWRT.org. TableOfHardware – OpenWrt, Obtained from <http://wiki.openwrt.org/TableOfHardware> on November 19<sup>th</sup>, 2005. Last updated October 12<sup>th</sup>, 2005.

Snort.org Rules. The tarball snortrules-snapshot-2.3.tar.gz was obtained from <http://www.snort.org/pub-bin/downloads.cgi> using a registered account. Downloaded on November 19<sup>th</sup>, 2005.

Figure 1. A front and rear image of the LinkSys WRT54GS version 2.1.



Figure 2. A screenshot of the web utility available with the original LinkSys firmware.

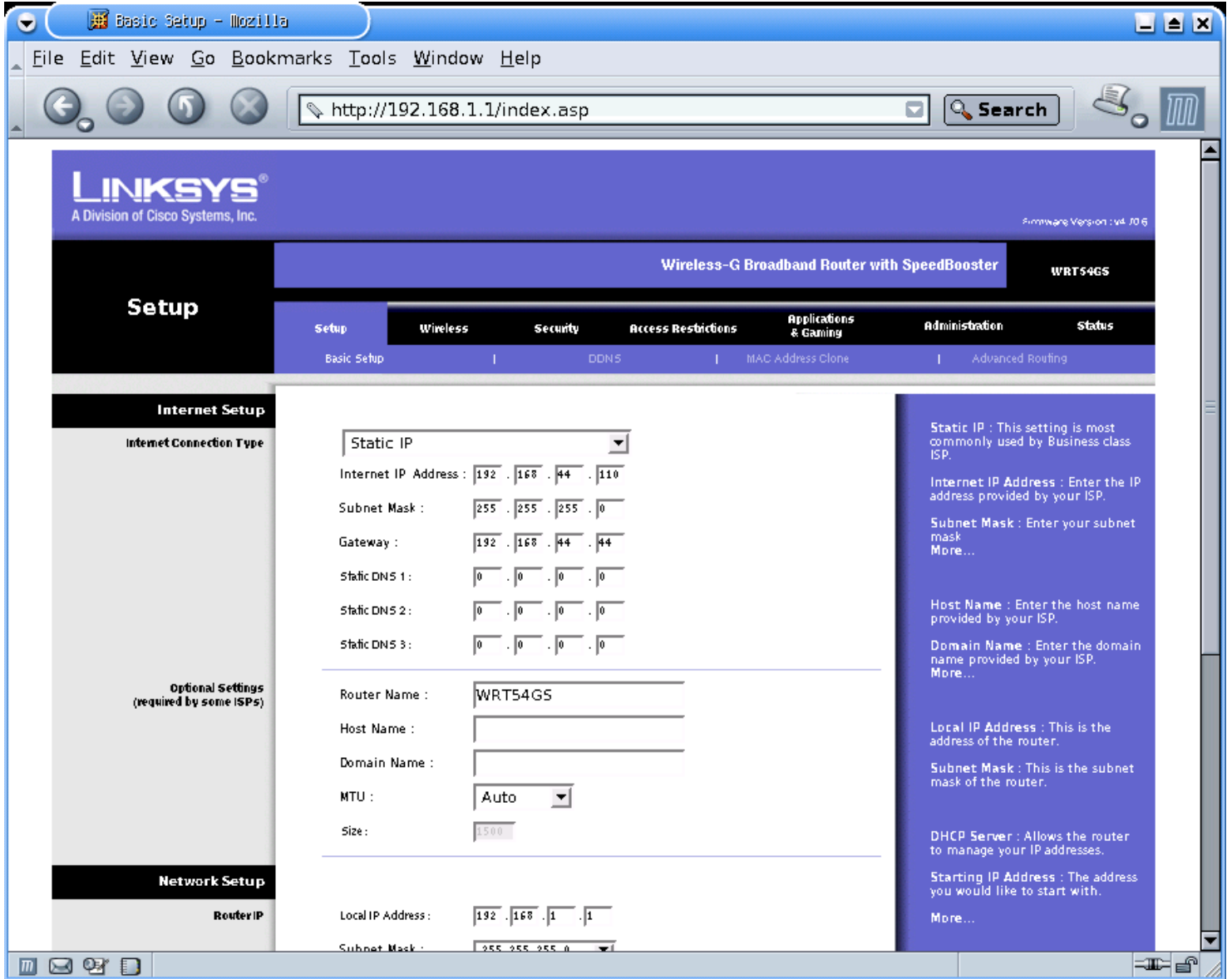


Figure 3. The diagnostics page in the native LinkSys administration utility.

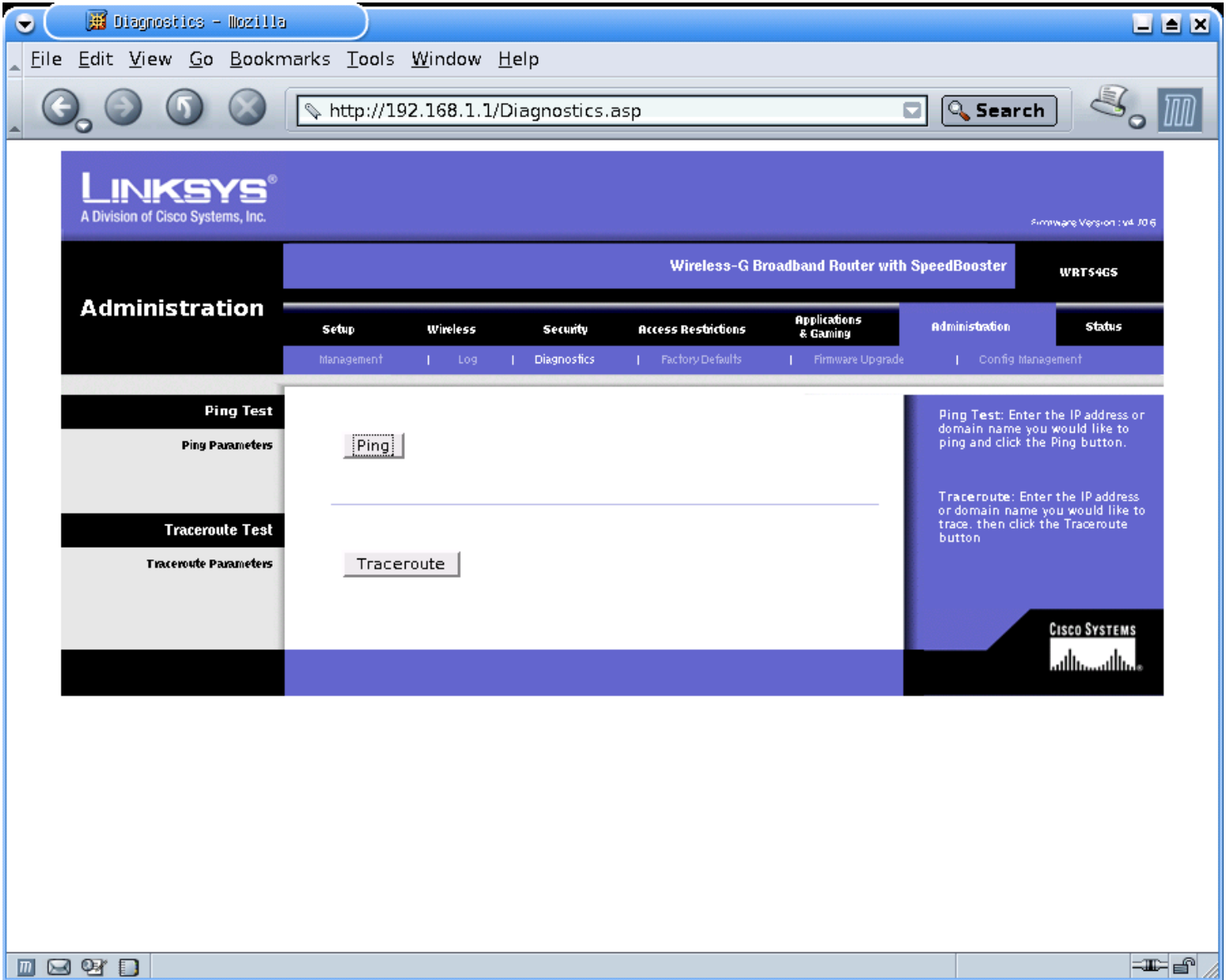


Figure 4. Inputting the commands into the ping utility in order to modify the boot\_wait variable.

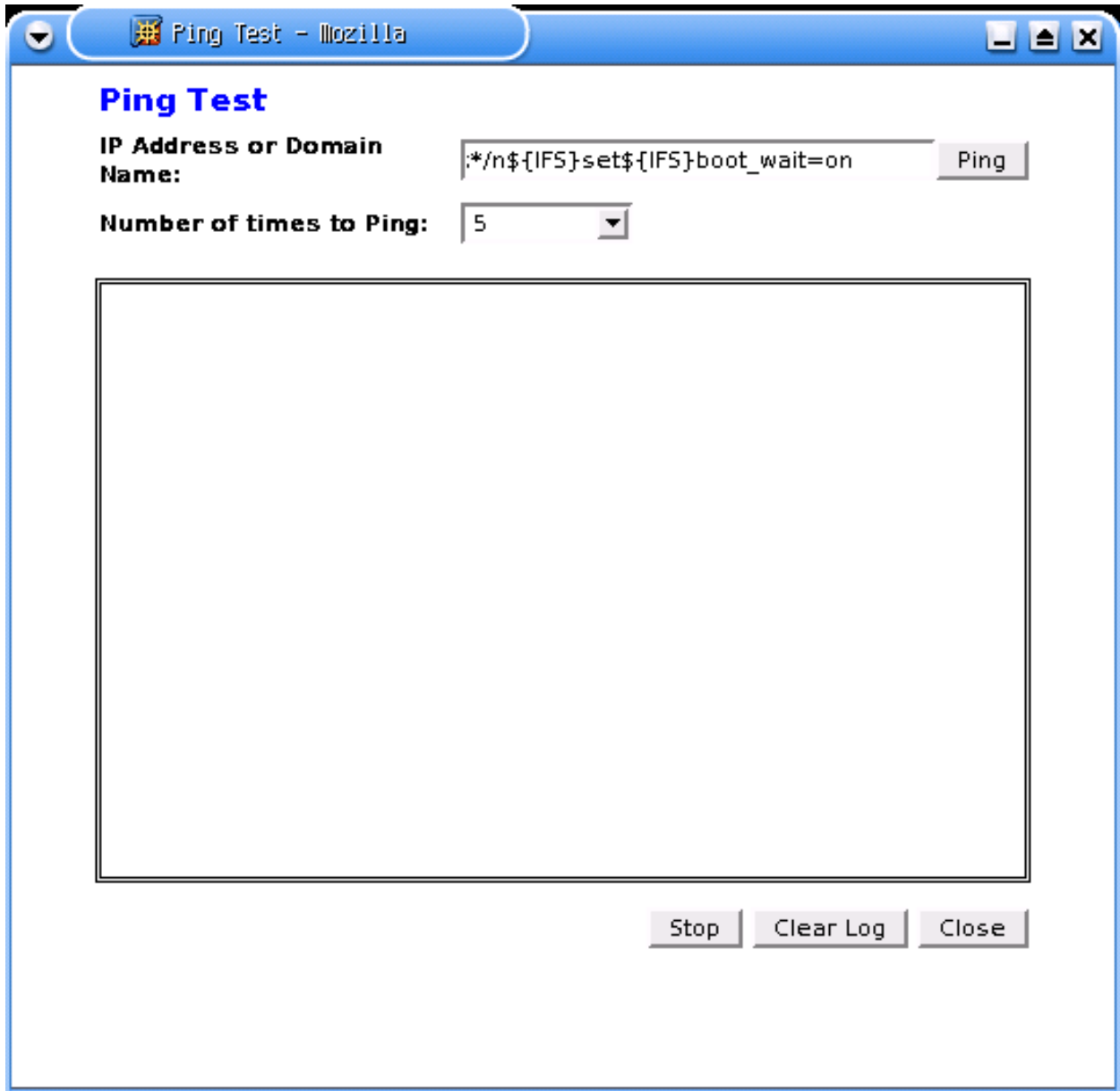


Figure 5. Successful results from modifying the boot\_wait variable on the router.

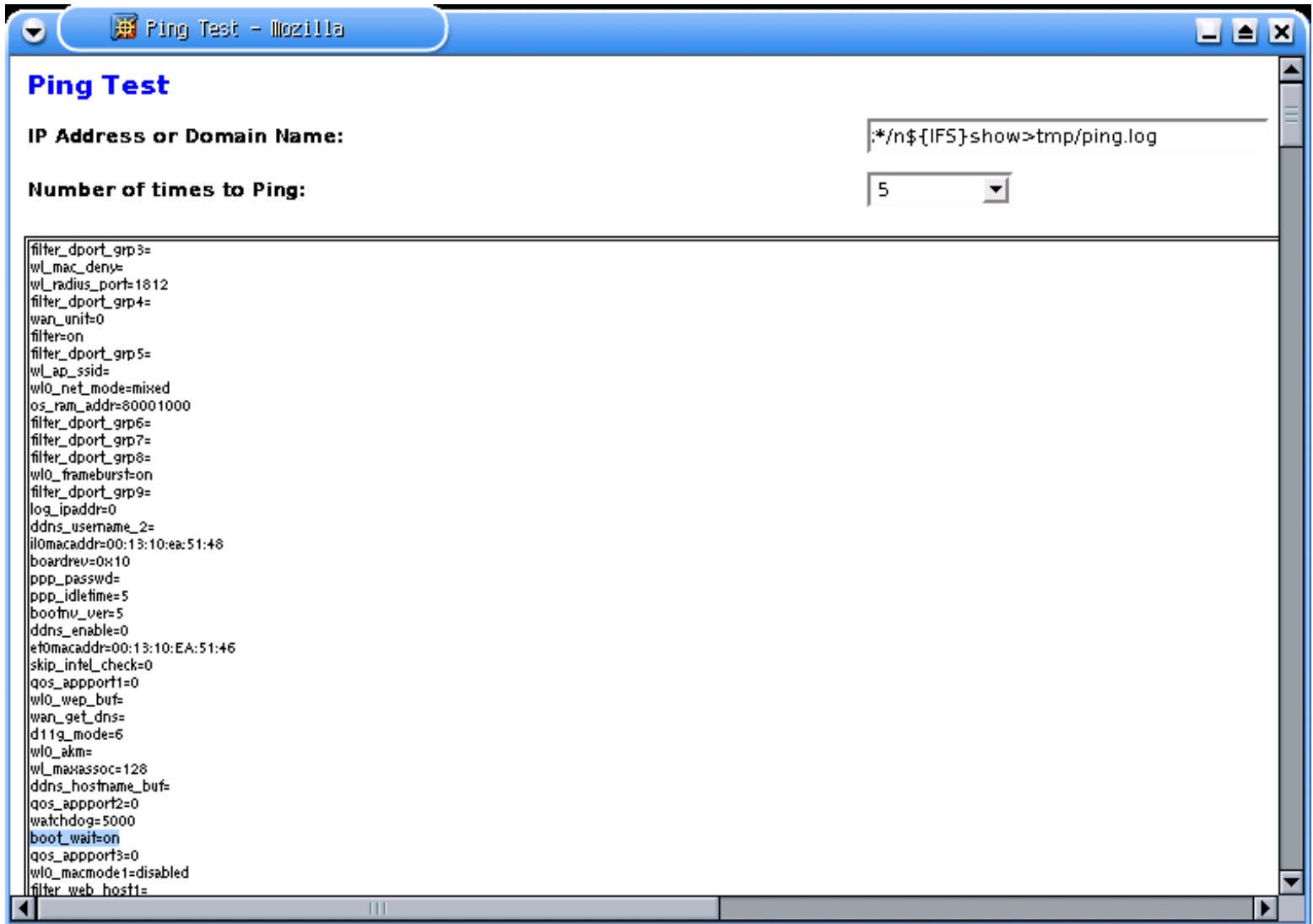


Figure 6. Preparing TFTP to upload the binary file openwrt-wrt54gs-jffs2.bin to the LinkSys.

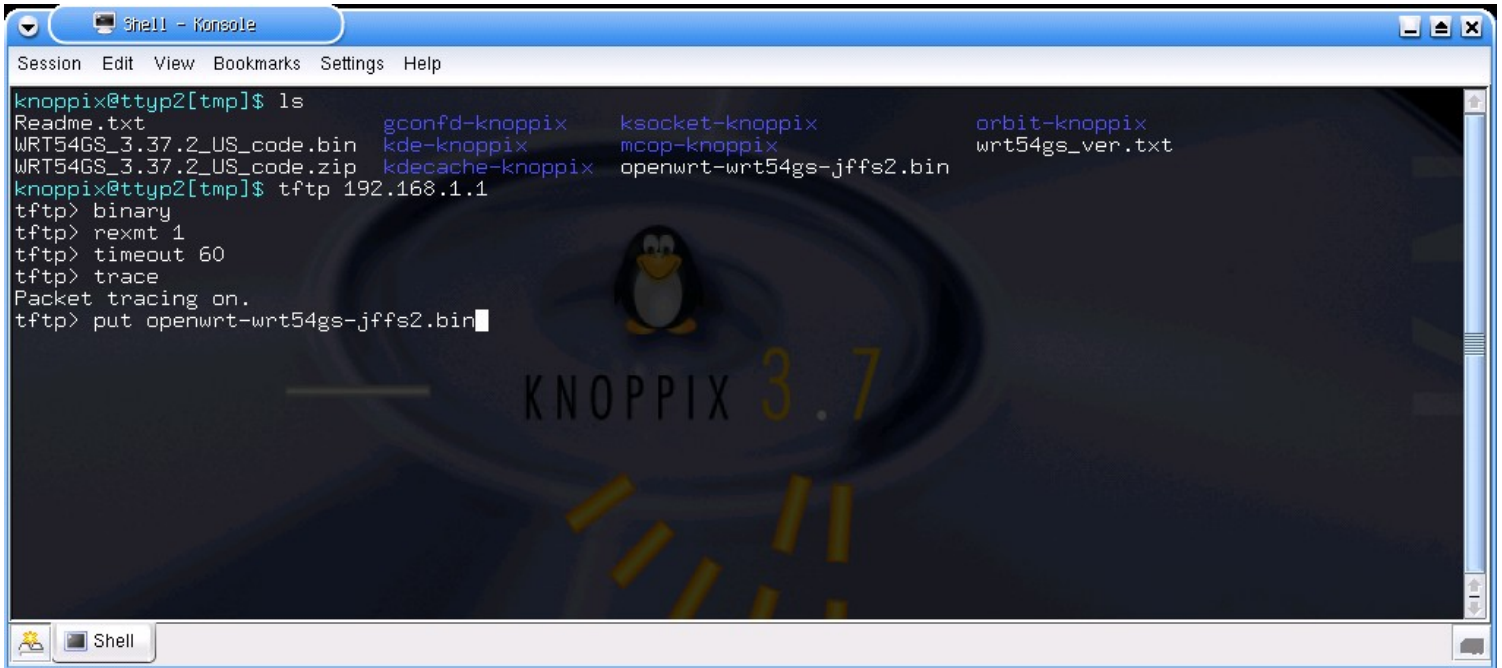


Figure 7. Data successfully being uploaded via TFTP to the LinkSys.

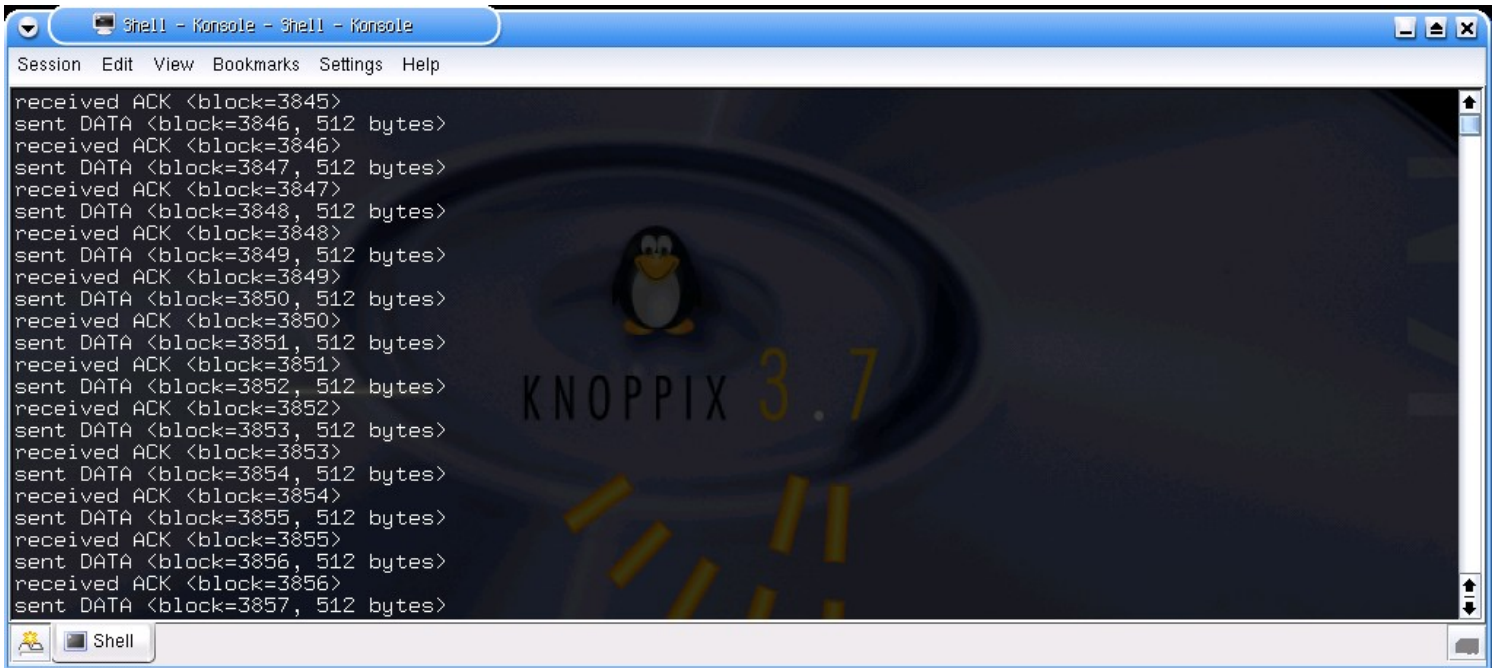


Figure 8. Telnet access is available to the LinkSys device after successful openWRT installation.

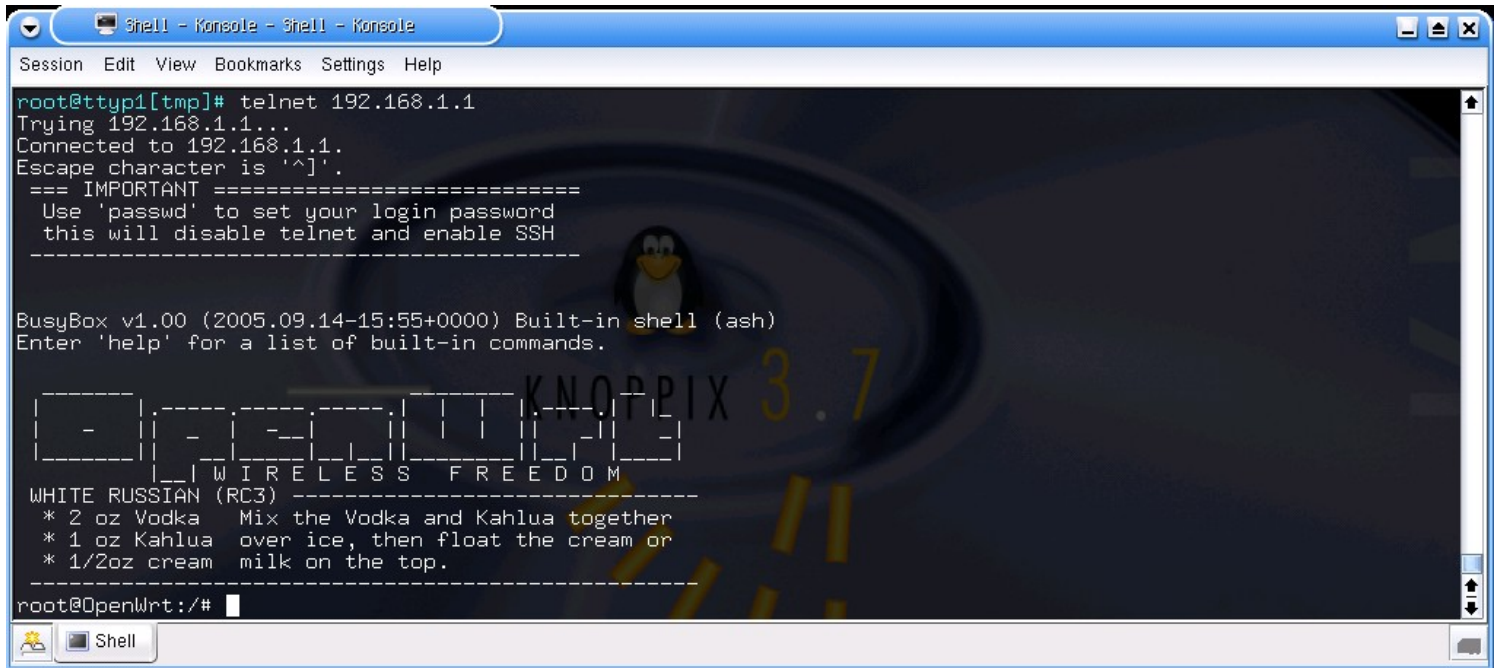


Figure 9. The passwd command is used to set the root password.

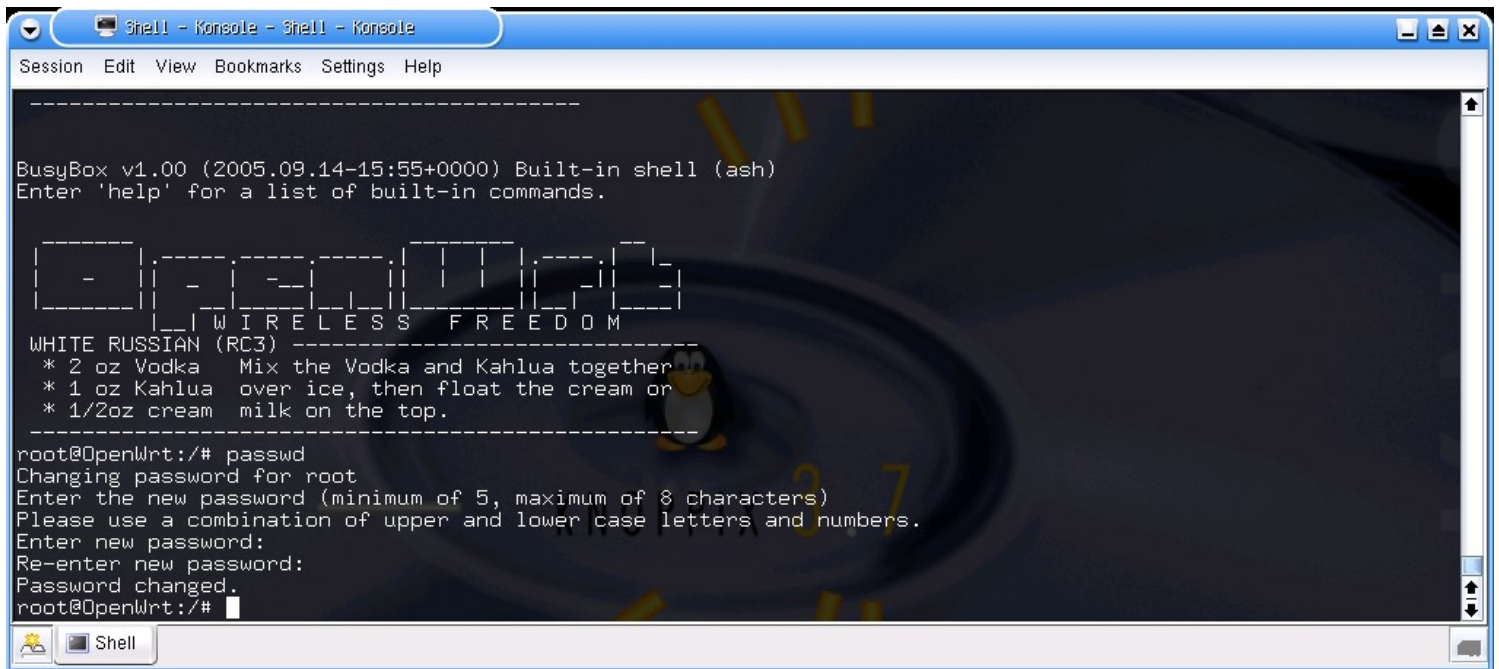
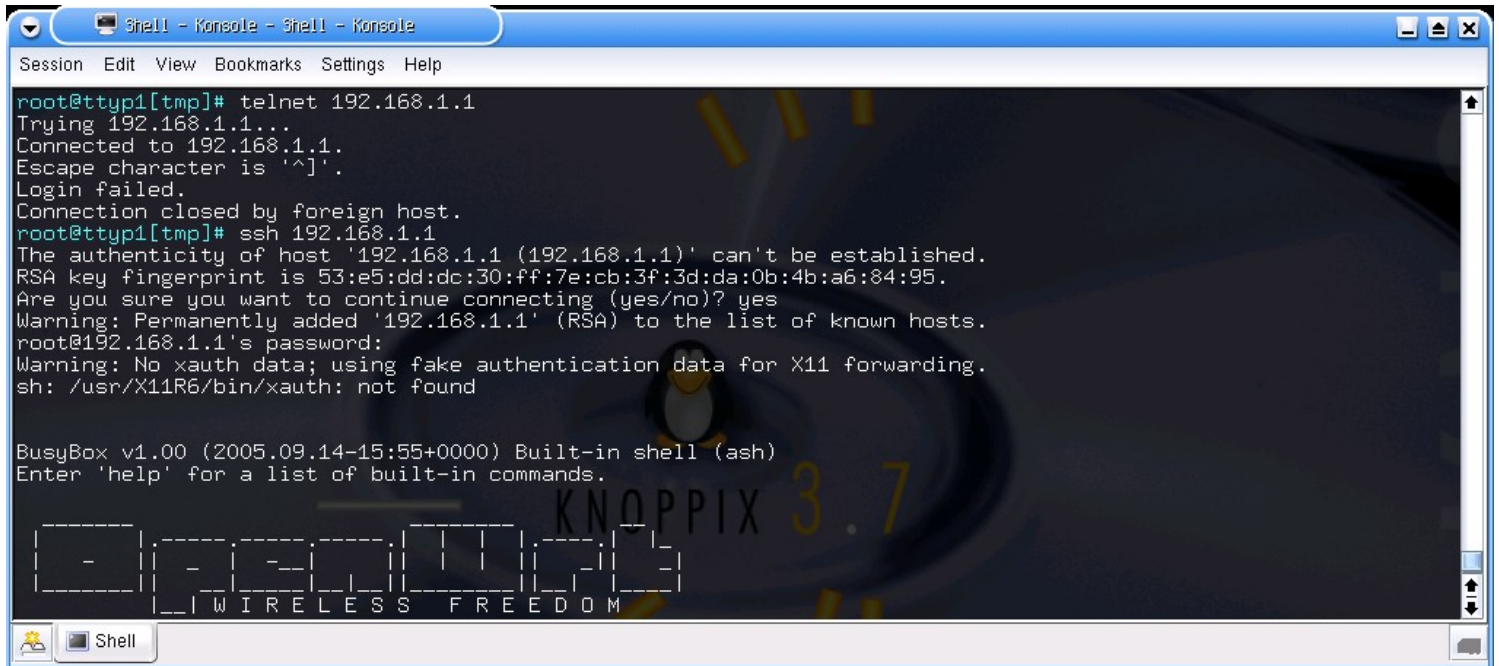


Figure 10. Telnet access is no longer allowed, only SSH access.

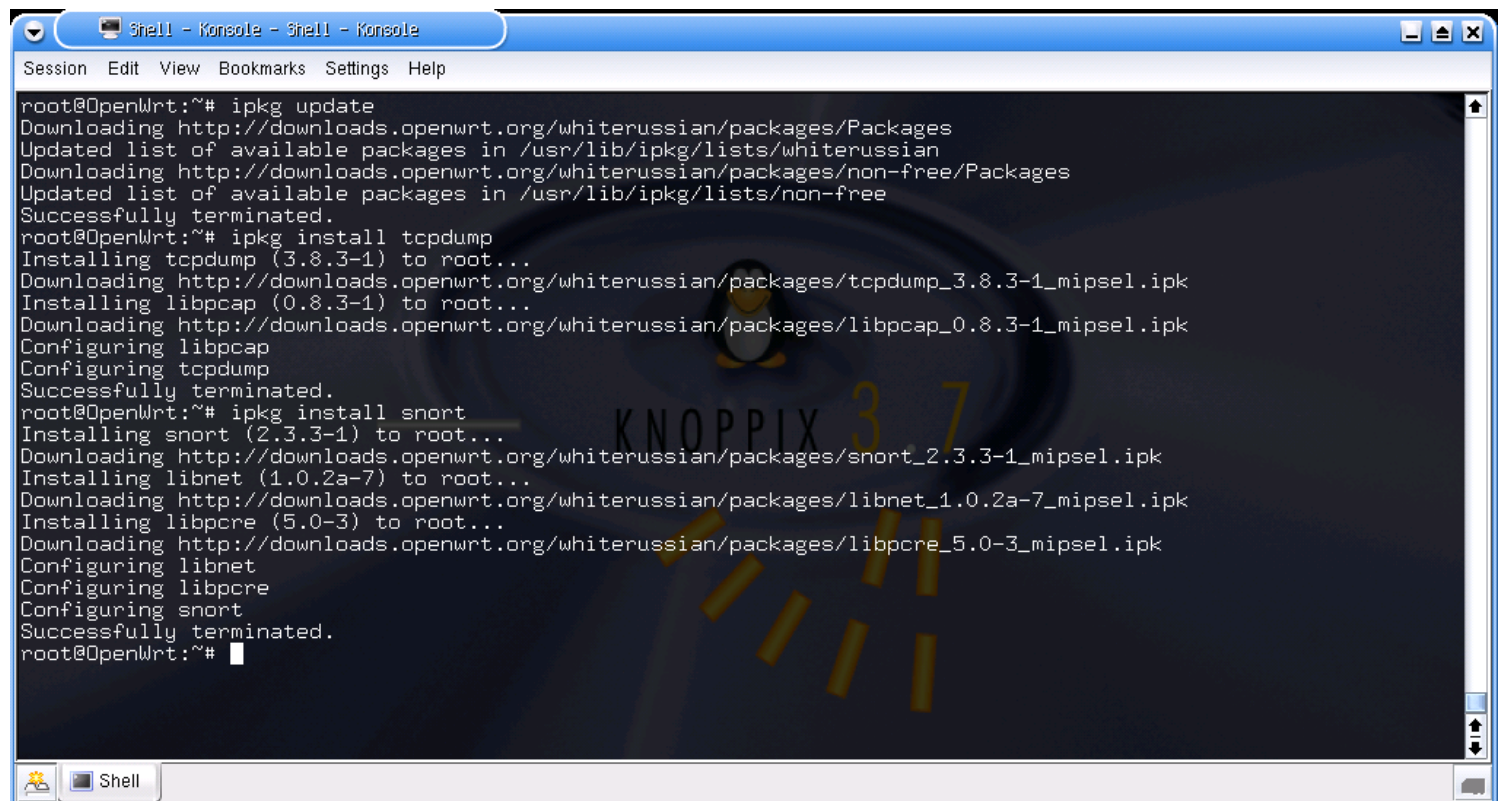
A terminal window titled "Shell - Konsole - Shell - Konsole" with a menu bar (Session, Edit, View, Bookmarks, Settings, Help). The terminal output shows a telnet attempt to 192.168.1.1 which fails with "Login failed." and "Connection closed by foreign host." followed by an ssh attempt to 192.168.1.1. The ssh attempt shows a warning about the host's RSA key fingerprint and asks for confirmation to continue, which is answered "yes". It then prompts for a password and shows a warning about X11 forwarding. The prompt changes to "sh: /usr/X11R6/bin/xauth: not found". Below the terminal output is a logo for "KNOPPIX 3.7" with the text "WIRELESS FREEDOM" underneath it.

```
root@tty1[tmp]# telnet 192.168.1.1
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.
Login failed.
Connection closed by foreign host.
root@tty1[tmp]# ssh 192.168.1.1
The authenticity of host '192.168.1.1 (192.168.1.1)' can't be established.
RSA key fingerprint is 53:e5:dd:dc:30:ff:7e:cb:3f:3d:da:0b:4b:a6:84:95.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.1' (RSA) to the list of known hosts.
root@192.168.1.1's password:
Warning: No xauth data; using fake authentication data for X11 forwarding.
sh: /usr/X11R6/bin/xauth: not found

BusyBox v1.00 (2005.09.14-15:55+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

[... W I R E L E S S F R E E D O M ...]
```

Figure 11. Updating the package list for ipkg and then using ipkg to install tcpdump and snort.

A terminal window titled "Shell - Konsole - Shell - Konsole" with a menu bar (Session, Edit, View, Bookmarks, Settings, Help). The terminal output shows the execution of "ipkg update" which downloads and updates package lists from "http://downloads.openwrt.org/whiterussian/packages/". It then shows "ipkg install tcpdump" which downloads and installs tcpdump (3.8.3-1) and libpcap (0.8.3-1). Finally, it shows "ipkg install snort" which downloads and installs snort (2.3.3-1), libnet (1.0.2a-7), and libpcrc (5.0-3). The terminal ends with the prompt "root@OpenWrt:~#".

```
root@OpenWrt:~# ipkg update
Downloading http://downloads.openwrt.org/whiterussian/packages/Packages
Updated list of available packages in /usr/lib/ipkg/lists/whiterussian
Downloading http://downloads.openwrt.org/whiterussian/packages/non-free/Packages
Updated list of available packages in /usr/lib/ipkg/lists/non-free
Successfully terminated.
root@OpenWrt:~# ipkg install tcpdump
Installing tcpdump (3.8.3-1) to root...
Downloading http://downloads.openwrt.org/whiterussian/packages/tcpdump_3.8.3-1_mipsel.ipk
Installing libpcap (0.8.3-1) to root...
Downloading http://downloads.openwrt.org/whiterussian/packages/libpcap_0.8.3-1_mipsel.ipk
Configuring libpcap
Configuring tcpdump
Successfully terminated.
root@OpenWrt:~# ipkg install snort
Installing snort (2.3.3-1) to root...
Downloading http://downloads.openwrt.org/whiterussian/packages/snort_2.3.3-1_mipsel.ipk
Installing libnet (1.0.2a-7) to root...
Downloading http://downloads.openwrt.org/whiterussian/packages/libnet_1.0.2a-7_mipsel.ipk
Installing libpcrc (5.0-3) to root...
Downloading http://downloads.openwrt.org/whiterussian/packages/libpcrc_5.0-3_mipsel.ipk
Configuring libnet
Configuring libpcrc
Configuring snort
Successfully terminated.
root@OpenWrt:~#
```

Figure 12. Extracting Snort rules to the /etc/snort/rules directory.

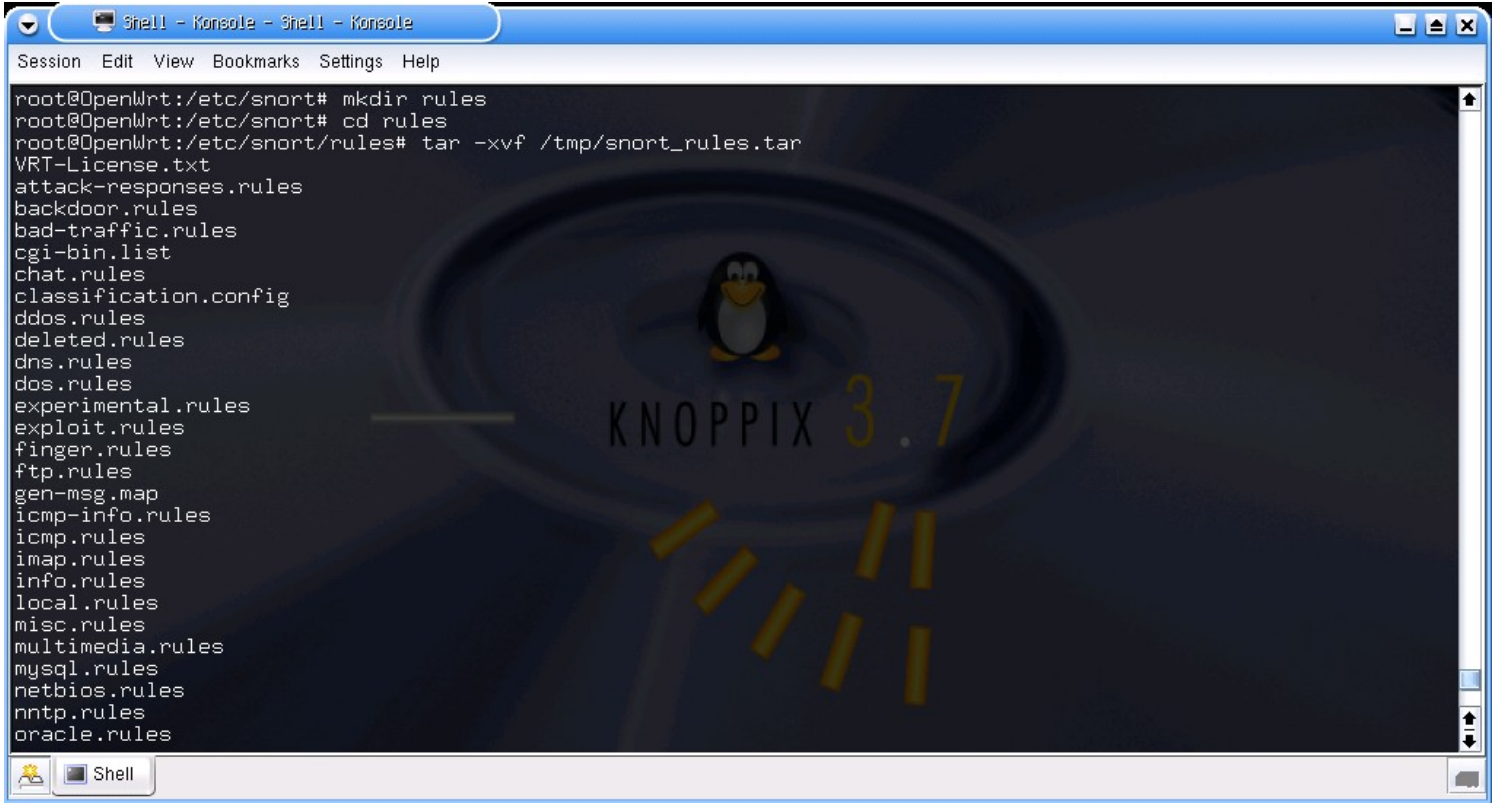


Figure 13. An abridged version of the file /etc/snort/snort.conf used for this experiment.

```
#-----  
# http://www.snort.org  Snort 2.3.3 Ruleset  
# Contact: snort-sigs@lists.sourceforge.net  
#-----  
# $Id: snort.conf,v 1.144.2.11 2005/04/22 19:15:49 jhewlett Exp $  
#  
#####  
# This file contains a sample snort configuration.  
# Most preprocessors and rules were disabled to save memory.  
# You can take the following steps to create your own custom configuration:  
#  
# 1) Set the network variables for your network  
# 2) Configure preprocessors  
# 3) Configure output plugins  
# 4) Customize your rule set  
#  
#####  
# Step #1: Set the network variables:  
#
```

```
# You must change the following variables to reflect your local network. The
# variable is currently setup for an RFC 1918 address space.
#
# You can specify it explicitly as:
#
var HOME_NET 192.168.1.0/24
#
# Set up the external network addresses as well. A good start may be "any"
var EXTERNAL_NET !$HOME_NET

# Configure the snort decoder
# =====
# Use a different pattern matcher in case you have a machine with very limited
# resources:
#
config detection: search-method lowmem

#####
# Step #2: Configure preprocessors
#
# General configuration for preprocessors is of
# the form
# preprocessor <name_of_processor>: <configuration_options>

# Configure Flow tracking module
# -----
preprocessor flow: stats_interval 0 hash 2

# frag2: IP defragmentation support
# -----
preprocessor frag2

# stream4: stateful inspection/stream reassembly for Snort
#-----
preprocessor stream4: disable_evasion_alerts
preprocessor stream4_reassemble

# bo: Back Orifice detector
# -----
# Detects Back Orifice traffic on the network. Takes no arguments in 2.0.
preprocessor bo

# Flow-Portscan: detect a variety of portscans
# -----
preprocessor flow-portscan: \
    talker-sliding-scale-factor 0.50 \
    talker-fixed-threshold 30 \
    talker-sliding-threshold 30 \
    talker-sliding-window 20 \
    talker-fixed-window 30 \
    scoreboard-rows-talker 30000 \
    server-watchnet [10.2.0.0/30] \
```

```
server-ignore-limit 200 \  
server-rows 65535 \  
server-learning-time 14400 \  
server-scanner-limit 4 \  
scanner-sliding-window 20 \  
scanner-sliding-scale-factor 0.50 \  
scanner-fixed-threshold 15 \  
scanner-sliding-threshold 40 \  
scanner-fixed-window 15 \  
scoreboard-rows-scanner 30000 \  
src-ignore-net [192.168.1.1/32,192.168.0.0/24] \  
dst-ignore-net [10.0.0.0/30] \  
alert-mode once \  
output-mode msg \  
tcp-penalties on  
  
# sfPortscan  
# -----  
preprocessor sfportscan: proto { all } \  
                        memcap { 10000000 } \  
                        sense_level { low }  
  
# X-Link2State mini-preprocessor  
# -----  
\preprocessor xlink2state: ports { 25 691 }  
  
#####  
# Step #3: Configure output plugins  
output alert_syslog: LOG_AUTH LOG_ALERT  
  
#####  
# Step #4: Configure snort with config statements  
  
config flowbits_size: 256  
  
#####  
# Step #5: Customize your rule set  
  
include $RULE_PATH/local.rules  
include $RULE_PATH/bad-traffic.rules  
include $RULE_PATH/exploit.rules  
include $RULE_PATH/scan.rules  
include $RULE_PATH/dos.rules  
include $RULE_PATH/ddos.rules  
  
include $RULE_PATH/icmp.rules  
include $RULE_PATH/netbios.rules  
include $RULE_PATH/misc.rules  
include $RULE_PATH/attack-responses.rules  
include $RULE_PATH/virus.rules
```

Figure 14. The /etc/default/snort file used for configuring Snort when it is run as a service.

```
INTERFACE="vlan1" # WAN
OPTIONS="-i $INTERFACE -c /etc/snort/snort.conf -D -N -q"
```

Figure 15. The output from the iptables --list command.

```
Chain INPUT (policy DROP)
target prot opt source destination
DROP all -- anywhere anywhere state INVALID
ACCEPT all -- anywhere anywhere state RELATED,ESTABLISHED
ACCEPT all -- 192.168.44.100 anywhere
DROP tcp -- anywhere anywhere tcp option=!2 flags:SYN/SYN
input_rule all -- anywhere anywhere
ACCEPT all -- anywhere anywhere
ACCEPT icmp -- anywhere anywhere
ACCEPT gre -- anywhere anywhere
REJECT tcp -- anywhere anywhere reject-with tcp-reset
REJECT all -- anywhere anywhere reject-with icmp-port-unreachable

Chain FORWARD (policy DROP)
target prot opt source destination
DROP all -- anywhere anywhere state INVALID
TCPMSS tcp -- anywhere anywhere tcp flags:SYN,RST/SYN TCPMSS clamp to PMTU
ACCEPT all -- anywhere anywhere state RELATED,ESTABLISHED
forwarding_rule all -- anywhere anywhere
ACCEPT all -- anywhere anywhere
ACCEPT all -- anywhere anywhere

Chain OUTPUT (policy DROP)
target prot opt source destination
DROP all -- anywhere anywhere state INVALID
ACCEPT all -- anywhere anywhere state RELATED,ESTABLISHED
output_rule all -- anywhere anywhere
ACCEPT all -- anywhere anywhere
REJECT tcp -- anywhere anywhere reject-with tcp-reset
REJECT all -- anywhere anywhere reject-with icmp-port-unreachable

Chain forwarding_rule (1 references)
target prot opt source destination

Chain input_rule (1 references)
target prot opt source destination

Chain output_rule (1 references)
target prot opt source destination
```

Figure 16. A process listing from the LinkSys that shows Snort running.

PID	Uid	VmSize	Stat	Command
1	root	392	S	init
2	root		SW	[keventd]
3	root		SWN	[ksoftirqd_CPU0]
4	root		SW	[kswapd]
5	root		SW	[bdflush]
6	root		SW	[kupdated]
7	root		SW	[mtdblockd]
26	root		SWN	[jffs2_gcd_mtd2]
41	root	392	S	syslogd -C 16
43	root	348	S	klogd
45	root	368	S	init
419	nobody	328	S	dnsmasq -l /tmp/dhcp.leases -K -F
192.168.1.100,192.168.1.150,255.255.255.0,12h -I				
424	root	360	S	httpd -p 80 -h /www -r WRT54G Router
431	root	304	S	telnetd -l /bin/login
451	root	300	S	/usr/sbin/dropbear
457	root	560	S	/usr/sbin/dropbear
458	root	544	S	-ash
492	root	544	S	/usr/sbin/dropbear
493	root	452	S	-ash
535	root	804	S	tcpdump -i 5
799	root	20492	S	snort -i vlan1 -c /etc/snort/snort.conf -l /tmp -D -N -q
858	root	384	R	ps auxww

Figure 17. A screenshot of a top showing the available resources and processes running.

```

Session  Edit  View  Bookmarks  Settings  Help
Mem: 29680K used, 904K free, 0K shrd, 0K buff, 4844K cached
Load average: 0.00, 0.00, 0.00 (State: S=sleeping R=running, W=waiting)
PID USER      STATUS  RSS    PPID  %CPU  %MEM  COMMAND
 860 root        R        412    458   0.7   1.3   top
 457 root        S        564    451   0.3   1.8   dropbear
 799 root        S        20M     1   0.0  67.0   snort
 535 root        S        804    493   0.0   2.6   tcpdump
 492 root        S        544    451   0.0   1.7   dropbear
 458 root        S        544    457   0.0   1.7   ash
 493 root        S        452    492   0.0   1.4   ash
   1 root        S        392     0   0.0   1.2   init
  41 root        S        392     1   0.0   1.2   syslogd
  45 root        S        368     1   0.0   1.2   init
 424 root        S        360     1   0.0   1.1   httpd
  43 root        S        348     1   0.0   1.1   klogd
 419 nobody      S        328     1   0.0   1.0   dnsmasq
 431 root        S        304     1   0.0   0.9   telnetd
 451 root        S        300     1   0.0   0.9   dropbear
   6 root        SW         0     1   0.0   0.0   kupdated
   3 root        SWN         0     1   0.0   0.0   ksoftirqd_CPU0
   4 root        SW         0     1   0.0   0.0   kswapd
   2 root        SW         0     1   0.0   0.0   keventd
   7 root        SW         0     1   0.0   0.0   mtdblockd
  26 root        SWN         0     1   0.0   0.0   jffs2_gcd_mtd2
   5 root        SW         0     1   0.0   0.0   bdflush

```

Figure 18. The contents of the LinkSys syslog after starting Snort and running the nmap scan.

```

Nov 19 11:33:09 (none) daemon.notice syslog: Initializing daemon mode
Nov 19 11:33:09 (none) daemon.notice syslog: PID path stat checked out ok, PID path set to /var/run/
Nov 19 11:33:09 (none) daemon.notice syslog: Writing PID "799" to file "/var/run//snort_vlan1.pid"
Nov 19 11:33:09 (none) daemon.notice syslog: Parsing Rules file /etc/snort/snort.conf
Nov 19 11:33:09 (none) daemon.notice syslog: Detection:
Nov 19 11:33:09 (none) daemon.notice syslog:   Search-Method = Low-Mem Trie
Nov 19 11:33:09 (none) daemon.notice syslog: ,-----[Flow Config]-----
Nov 19 11:33:09 (none) daemon.notice syslog: | Stats Interval: 0
Nov 19 11:33:09 (none) daemon.notice syslog: | Hash Method: 2
Nov 19 11:33:09 (none) daemon.notice syslog: | Memcap: 10485760
Nov 19 11:33:09 (none) daemon.notice syslog: | Rows : 4099
Nov 19 11:33:09 (none) daemon.notice syslog: | Overhead Bytes: 16400(%0.16)
Nov 19 11:33:09 (none) daemon.notice syslog: `-----
Nov 19 11:33:09 (none) daemon.notice syslog: Portscan Detection Config:
Nov 19 11:33:09 (none) daemon.notice syslog:   Detect Protocols: TCP UDP ICMP IP
Nov 19 11:33:09 (none) daemon.notice syslog:   Detect Scan Type: portscan portsweep decoy_portscan distributed_portscan
Nov 19 11:33:09 (none) daemon.notice syslog:   Sensitivity Level: Low
Nov 19 11:33:09 (none) daemon.notice syslog:   Memcap (in bytes): 10000000
Nov 19 11:33:09 (none) daemon.notice syslog:   Number of Nodes: 36900
Nov 19 11:33:09 (none) daemon.notice syslog:
Nov 19 11:33:09 (none) daemon.notice syslog: X-Link2State Config:
Nov 19 11:33:09 (none) daemon.notice syslog:   Ports: 25 691
Nov 19 11:33:17 (none) daemon.notice syslog: Warning: flowbits key smb.tree.create.llsrpc is set but not ever checked.
Nov 19 11:33:17 (none) daemon.notice syslog: Warning: flowbits key dce.bind.veritas is set but not ever checked.
Nov 19 11:33:17 (none) daemon.notice syslog:
Nov 19 11:33:17 (none) daemon.notice syslog: +-----[thresholding-config]-----

```

```
Nov 19 11:33:17 (none) daemon.notice syslog: | memory-cap : 1048576 bytes
Nov 19 11:33:17 (none) daemon.notice syslog: +-----[thresholding-global]-----
Nov 19 11:33:17 (none) daemon.notice syslog: | none
Nov 19 11:33:17 (none) daemon.notice syslog: +-----[thresholding-local]-----
Nov 19 11:33:17 (none) daemon.notice syslog: | gen-id=1   sig-id=2523   type=Both   tracking=dst count=10 seconds=10
Nov 19 11:33:17 (none) daemon.notice syslog: | gen-id=1   sig-id=2923   type=Threshold tracking=dst count=10
seconds=60
Nov 19 11:33:17 (none) daemon.notice syslog: | gen-id=1   sig-id=2924   type=Threshold tracking=dst count=10
seconds=60
Nov 19 11:33:17 (none) daemon.notice syslog: | gen-id=1   sig-id=2494   type=Both   tracking=dst count=20 seconds=60
Nov 19 11:33:17 (none) daemon.notice syslog: | gen-id=1   sig-id=3527   type=Limit   tracking=dst count=5  seconds=60
Nov 19 11:33:17 (none) daemon.notice syslog: | gen-id=1   sig-id=2496   type=Both   tracking=dst count=20 seconds=60
Nov 19 11:33:17 (none) daemon.notice syslog: | gen-id=1   sig-id=2495   type=Both   tracking=dst count=20 seconds=60
Nov 19 11:33:17 (none) daemon.notice syslog: +-----[suppression]-----
Nov 19 11:33:17 (none) daemon.notice syslog: | none
Nov 19 11:33:17 (none) daemon.notice syslog: +-----
Nov 19 11:33:17 (none) daemon.notice syslog: Rule application order: ->activation->dynamic->alert->pass->log
Nov 19 11:33:17 (none) daemon.notice syslog: Log directory = /tmp
Nov 19 11:33:17 (none) daemon.notice snort: Snort initialization completed successfully (pid=799)
Nov 19 11:33:26 (none) kern.info snort: Portscan detected from 192.168.44.100 Talker(fixed: 30 sliding: 30)
Scanner(fixed: 0 sliding: 0)
```