

# IPTables Linux firewall with packet string-matching support

*Anton Chuvakin* 2001-12-31

## IPTables Linux firewall with packet string-matching support

by *Anton Chuvakin, Ph.D.*

last updated December 31, 2001

---

Linux firewalling code has come a long way since the time ipfwadm was introduced in kernel version 1.2.1 in 1995. Ipfwadm enabled standard TCP/IP packet filtering features such as filtering by source/target addresses and port numbers. Then, in early 1999, when the first stable 2.2.0 kernel was released, firewalling code was replaced with new ipchains-controlled code. New features included support for chains of rules, fragmentation handling, better network address translation (NAT) support and several usability improvements. Readers should be reminded that Linux firewalling includes kernel-level code (usually in form of loadable module or kernel source patch) and user-level code (a control utility such as /usr/bin/ipchains, that is used to insert packet rules into kernel-space). Thus whenever new Linux firewalling code was introduced it involved both kernel and userspace code rewrite.

With the release of 2.4 in 2001, iptables code came out. It introduced many important improvements such as stateful firewalling, filtering packets based on any combination of TCP flags and on MAC address, more configurable and flexible logging, powerful and easy to use support for network address translation (NAT) and transparent proxies, DoS blocking support by rate-limiting and others. (For details see, for example [A Comparison of iptables Automation Tools](#).)

However, the most important architectural addition was the introduction of the modular architecture. For example, ipchains and ipfwadm compatibility mode is implemented as a set of kernel modules that can be inserted into the running kernel to provide the corresponding functionality. In addition, user-coded modules are now possible. For example, filtering by port range, TTL value and time of packet arrival, stateful inspection for custom protocols, and random packet inspection are not part of the iptables suite, but were implemented later. Many new and interesting modules were coded. To program a module one has to create a kernel-level part that will be compiled into a loadable module and user-level part that will be used to control the filtering behavior. For more details see [Rusty Russell's Linux iptables HOWTO](#).

This article will cover string pattern matching functionality (evidently, implemented as a module), which allows limited investigation of the packet payload. This is a significant breakthrough for iptables technology since it goes beyond inspecting TCP/IP flags, which is standard for packet filter firewalls. It is well-known that firewalls can be loosely categorized into proxies and packet filters. The latter "know" the application-level protocols such as telnet, HTTP or SMTP and can inspect the protocol payloads and verify the commands. This comes at a significant performance penalty since packets have to be processed higher in the network protocol stack in application layer. For each inspected protocol a new proxy should be written. The packet filters, on the other hand, can usually only inspect source and target addresses and ports, TCP/IP flags and have to totally ignore higher-layer protocol payloads. Due to that reason, they are usually much faster than proxy firewalls (3-10 times). Thus proxies are used for more granular security while packet filters are used on higher bandwidth lines for higher throughput.

In light of the above, adding content inspection capabilities in iptables presents an attempt to cross the bridge between two firewall groups without getting stuck in disadvantages of either method. It also clearly demonstrated an advantage of a new modular architecture over old ipchains code. It should be noted that packet filter such as iptables does not become higher-layer-protocol-aware since it still operates at network level (layer 3 in OSI structure), but is only allowed to peek at payloads, rather than analyze the application-level communication structure.

Before the string matching module was started in May 2001, there were attempts to add content inspection to iptables-based firewalls. One such project is [Hogwash](#), which couples the Snort IDS rule-matching engine with iptables in order to respond to packets with attack signatures in them.

Now we will provide a step-by-step direction for enabling string matching packet inspection for RedHat Linux. Standard RH 7.2 comes with iptables 1.2.3 and 1.2.4 is available as an RPM update. However, string-matching functionality is not included in either since it is marked as "experimental" by developers. Thus some kernel recompilation is required.

If you are using RH 7.1-7.2 you already have kernel version 2.4 installed. You need at least 2.4.4 for the latest iptables 1.2.4 to function. It is always recommended to download the latest kernel version available from your system distributor. At present, there is one exception from this rule: iptables-1.2.4 string matching patch does not seem to work with 2.4.9 kernel version. You should have kernel source RPM package installed (that usually puts the full source tree in /usr/src/linux-2.4.x) or just have the kernel source downloaded from elsewhere (such as [www.kernel.org](http://www.kernel.org) or one of its mirrors).

For this article, the latest 2.4.16 will be used as an example. Test were also run with 2.4.7 kernel shipped with RedHat 7.2, but the 2.4.7 kernel is not recommended due to several minor bugs in security mechanisms such as SYN-cookie protection and iptables save/restore functionality.

Surely it is redundant to remind the reader to backup the entire kernel source tree before starting the experiment or keep a source RPM handy. Keep in mind, that the latter will only help if you have not compiled the kernel before.

Now, iptables code should be downloaded from <http://netfilter.samba.org/iptables-1.2.4.tar.bz2> . After archive is unpacked, the iptables should be configured and the appropriate source code merged with the kernel source tree. The semi-automated program is available for that purpose. First, one might want to run the program to include the iptables patches that are already considered stable, but have not been included in the kernel release. From the directory where iptables are unpacked (in this example iptables are in /home/anton/iptables-1.2.4 and kernel source is in /usr/src/linux-2.4.16) run:

```
make pending-patches KERNEL_DIR=/usr/src/linux-2.4.16
```

That will start the process of interactive patch application. It appears, that, while you can safely apply all of those patches, none are required for iptables string support. Say *y* (yes) to whichever patches you think will be needed for your installation. If you want to play it safe choose none.

Now we are ready to apply experimental patches such as string matching support. Run:

```
make patch-o-matic KERNEL_DIR=/usr/src/linux-2.4.16
```

In the interactive session that follows answer *y* (yes) to application of `string.patch`. The program will go through all available patches, including the stable ones.

```
Testing... string.patch NOT APPLIED ( 2 missing files)
The string patch:
```

```
Author: Emmanuel Roger
Status: Working, not with kernel 2.4.9
```

```
This patch adds CONFIG_IP_NF_MATCH_STRING which allows you to
match a string in a whole packet.
```

```
THIS PATCH DOES NOT WORK WITH KERNEL 2.4.9 !
```

```
Do you want to apply this patch [N/y/t/f/q/?] y
```

While the rest of the patches may sound like fun as well, they are not relevant to this article. If you choose to install any other patches, heed the warnings given by developers about what patches break functionality (such as dropped tables patch). Make sure you do not install the MAC filtering patch since it was recently found to contain a bug.

Now we are ready to compile the user-space code and the libraries:

```
make KERNEL_DIR=/usr/src/linux-2.4.16
```

and then install them (iptables program goes in `/usr/local/user/sbin` and libraries go into `/usr/local/lib/iptables`).  
As root:

```
make install KERNEL_DIR=/usr/src/linux-2.4.16
```

Now we are ready to configure and compile the kernel:

```
cd /usr/src/linux-2.4.16
```

Any of the kernel configuration methods can be used. Detailed discussion of this is provided in a huge number of Internet sources such as kernel HOWTO. In brief:

```
make xconfig
```

In the GUI that appears got to Netfilter configuration and choose *m* (for modular support) in `String match support (EXPERIMENTAL)`. See picture



Then follow with a standard

```
make dep ; make bzImage ; make modules ; make modules_install
```

Now install the kernel itself using you favorite method and reboot. Upon reboot, test whether the iptables with string support is enabled. As root:

```
/usr/local/sbin/iptables -m string -help
```

It should produce the iptables standard help message appended by:

```
STRING match v1.2.4 options:
--string [!] string Match a string in a packet
```

The resulting functionality will allow you to match packets by the content. Some real life tests can be performed using netcat or telnet to make sure we can really grab packets by the content.

Run:

```
iptables -A INPUT -m string --string "test" -j LOG --log-level
info --log-prefix "TEST"
```

Then start a netcat server by:

```
nc -l -p 3456
```

Connect to it via:

```
telnet localhost 3456
```

Now type:

```
test
whatevertestdoes
```

It should produce the message similar to the following in your log files (providing you log messages with severity level 'info' somewhere)

```
Nov 27 23:16:53 pua kernel:
  TEST IN=lo OUT=MAC=00:00:00:00:00:00:00:00:00:00:00:08:00
SRC=127.0.0.1 DST=127.0.0.1 LEN=2154 TOS=0x00 PREC=0x00 TTL=64
ID=42880 DF PROTO=TCP SPT=3128 DPT=33018 WINDOW=32767 RES=0x00 ACK PSH URGP=0
```

```
Nov 27 23:16:53 pua kernel:
  TEST IN=lo OUT=MAC=00:00:00:00:00:00:00:00:00:00:00:08:00
SRC=127.0.0.1 DST=127.0.0.1 LEN=1830 TOS=0x00 PREC=0x00 TTL=64
ID=17451 DF PROTO=TCP SPT=8000 DPT=33017 WINDOW=32767 RES=0x00 ACK PSH URGP=0
```

Every time the string `test` is present in TCP/IP packets, the above log message is produced. What is it good for? Many things. As was suggested in the excellent article, [Filtering packets based on string matching](#), by sysctl at Linuxguru.net, it can be used to block all those pesky IIS worms from filling your UNIX web server log files with requests to `cmd.exe` etc. Those worms cannot hurt you, but if your `/var` partition (where logs are usually stored) is not too big this measure can be pretty helpful. Just silently drop all those port 80 requests or log them using the message limiting facility:

To silently drop them:

```
iptables -I INPUT -j DROP -p tcp -s 0.0.0.0/0 -m string --string "cmd.exe"
```

To only log no more than 1 message per hour:

```
iptables -I INPUT -j LOG -p tcp -s 0.0.0.0/0 -m string --string "cmd.exe" -m limit
--limit 1/hour
```

Replace the string with whatever the new Windows "worm-of-the-moment" is requesting and just add another rule.

You can also stop requests to things you do not want to have retrieved from your Web server. Apparently, `test.cgi` should be removed, rather than put into iptables rules. However, in some cases the ability to stop requests for specific documents at the TCP/IP level before they reach the Web server access control facility might provide the needed "defense in-depth" for your Web infrastructure.

Another great suggestion from Bill Stearns (author of Mason firewall building script) is to convert your [Snort](#) network IDS rules into iptables rules with string support. Snort IDS attack signature database contains about 1200 signatures and appears to be the biggest publicly available attack database suitable for instant deployment.

The ability to use the ready-made signatures for iptables is of immense value. The page that describes his experimental software is at <http://www.stearns.org/snort2iptables/>. There, you can find the shell script to convert a standard Snort ruleset into iptables rules. Here are a couple of examples for well-known Linux attacks against mountd and bind network daemons:

#### Snort rules:

1. alert udp \$EXTERNAL\_NET any -> \$HOME\_NET 518  
 (msg:"EXPLOIT ntalkd x86 linux overflow";  
 content:"|0103 0000 000 0 0001 0002 02e8|";  
 reference:bugtraq,210; classtype:attempted-admin; sid:313; rev:2;)
2. alert tcp \$EXTERNAL\_NET any -> \$HOME\_NET 53  
 (msg:"EXPLOIT named tsig infoleak";  
 content: "|AB CD 09 80 00 00 00 01 00 00 00 00 00 01 00 01 20 20 20 20 02 61|";  
 reference:cve,CAN-2000-0010; reference:bugtraq,2302; reference:arachnids,482;  
 classtype:attempted-admin; sid:303; rev:3;)
3. alert udp \$EXTERNAL\_NET any -> \$HOME\_NET 635  
 (msg:"EXPLOIT x86 linux mountd overflow";  
 content:"|5eb0 0289 06fe c889 4604 b006 8946|";  
 reference:cve,CVE-1999-0002; classtype:attempted-admin; sid:315; rev:1;)

#### Converted iptables rules:

1. iptables -A SnortRules -p udp -s \$EXTERNAL\_NET -d \$HOME\_NET --dport 518 -m string --string " è" -j LOG --log-prefix "SID313 " # "EXPLOIT ntalkd x86 linux overflow" bugtraq,210 classtype:attempted-admin sid:313
2. iptables -A SnortRules -p tcp -s \$EXTERNAL\_NET -d \$HOME\_NET --dport 53 -m string --string "«Í .a" -j LOG --log-prefix "SID303 " # "EXPLOIT named tsig infoleak" cve,CAN-2000-0010 bugtraq,2302 arachnids,482 classtype:attempted-admin sid:303
3. iptables -A SnortRules -p udp -s \$EXTERNAL\_NET -d \$HOME\_NET --dport 635 -m string --string "^ % pÈ%F ° %F" -j LOG --log-prefix " cve-CVE-1999-0002 " # "EXPLOIT x86 linux mountd overflow" classtype:attempted-admin sid:315

It is easy to see that the above conversion uses the buffer overflow string used for the above exploits to catch the attack. Some rules are not converted mostly due to the fact that Snort is still "smarter" than iptables in fragmentation handling.

Overall, iptables with string support can be used to protect networks (if deployed on the organization gateway) and individual hosts (deployed as a part of host hardening) from many attacks on the network services that have to be open to the world (WWW, mail, DNS, ftp) and have not been protected by ordinary packet filters. In addition, iptables string matching can also help with policy enforcement - just implement the rules that stop inappropriate content by keyword. There might be better solutions of implementing the content scanning, but

another layer of protection never hurts.

*[Anton Chuvakin](#), Ph.D. is in the process of looking for another infosec job after having his fun dot-bomb experience. He spends his plentiful pastime running Linux, building a small [infosec portal](#) and preparing for CISSP.*

[Privacy Statement](#)

Copyright 2006, SecurityFocus