GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials Bootcamp Style (Security 401)"
at http://www.giac.org/registration/gsec

Netfilter and IPTables – A Structural Examination

By Alan Jones

GSEC Practical v. 1.4b, Option 1

February 26, 2004

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Abstract

In this paper a study is made of the Linux packet manipulation framework, Netfilter, and the packet matching system built on top of it, IPTables. The structure of both systems is discussed, detailing both their arrangement as well as the mechanism through which they perform their functions. Also discussed are features of the framework germane to its role as a firewall. Additionally, installation of the system is covered. Finally, usage of the package is detailed. Ultimately, the discussion herein examines in detail the Netfilter/IPTables system, extending beyond a simple "how-to" style document and providing the reader with a deeper understanding of the technology.

Netfilter and IPTables, an introduction

Netfilter is a flexible packet manipulation framework built into the Linux 2.4 and 2.6 series of kernels. Layered on top of this framework is the packet selection system, IPTables. Combined, they produce a modular system that enables the Linux kernel to perform firewalling, Network Address Translation [NAT], Port Address Translation [PAT], and many other useful manipulations of network data.[1]

Netfilter is implemented in the Linux kernel as a framework that allows callback functions to be attached to network events. These callback functions can be implemented as kernel modules, thus allowing IPTables to inherit the flexibility of the Linux kernel module system. A wide variety of modules have been built, allowing the Netfilter framework to provide a number of useful services.

Software Structure and Design

Netfilter is implemented as a series of hooks that are inserted into the Linux networking code. These hooks permit kernel modules to register with them, listening to data arriving at each one of these locations and acting accordingly. Multiple callbacks can be attached to each hook, and are called in order of precedence. Once a packet has been passed to a module, the module can manipulate the packet freely. Once this is done, the module can instruct Netfilter to allow the packet to continue traversal of the system, drop the packet, stop further processing of the packet, queue the packet for manipulation by userspace programs, or repeat the callback function again.[2]

Netfilter hooks exist in numerous places throughout the Linux networking code. First is the NF_IP_PRE_ROUTING hook, which is called when incoming packets arrive at the system but before routing decisions are made. NF_IP_LOCAL_IN is

[1] Welte, Harald. "netfilter/iptables project homepage." URL:http://www.netfilter.org/index.html (25 Feb 2004)
[2] Russell, Rusty and Welte, Harald. "Linux netfilter hacking howto." 2 July 2002 URL: http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO-3.html (25 Feb 2004)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

called after routing, but only if the packet is destined for a process on the local computer. If the packet is determined to be sent to another interface, or forwarded, then the NF_IP_FORWARD hook is called. If the packet is from the IPTables server, but destined for another system, the NF_IP_LOCAL_OUT hook is called. Finally the NF_IP_POST_ROUTING hook is called for all packets leaving the computer, regardless of destination.[3] Programmatically, IPTables is a set of modules that attach to these hooks, firing certain functions based on their attachment points within the Netfilter framework.

Logically, IPTables is implemented as a set of layers on top of the Netfilter framework. The IPTables system has two layers. The first layer is the table layer. In its default configuration, IPTables has three tables, or categories of packet manipulation, through which data is directed. Within each of these tables are "chains" of rules. IPTables rules are processed in numeric order within each chain and are "first match" style rules, meaning that once a rule matches a particular packet, the actions of that rule are executed and further processing of the chain stops. Actions are table dependant, and will be discussed per table.

The Filter Table

The first and most commonly used of these is the filter table. The filter table is where the bulk of firewalling rules are created. The filter table contains three chains of rules. The first of these chains is the INPUT chain. The INPUT chain is referred to when the kernel sees packets that are destined for the computer on which Linux is running. The second chain is the FORWARD chain. This chain is applied when Linux has packets for which it is not the source, nor the destination. The final chain is the OUTPUT chain, which is referred to for packets that originate from the computer destined for another location. These three chains comprise the filter table.

The actions that can be performed by the filter table are somewhat limited. The function of this table is to filter data, as its name suggests. Thus the actions, also referred to as targets, that are permitted within the filter table are limited. The first is to ACCEPT, which passes the packet back to the networking code as permissible. The second action that can be applied is to DROP, which kills the packet and stops processing. Similarly, the REJECT action can be applied. This action drops the packet similarly to the DROP target, but also causes an ICMP error packet to be transmitted back to the sending host. Finally the LOG action is permitted, which causes the kernel to log data about the matching packet to the syslog facility. It is important to note that no rules are allowed within the filter table which cause the packet to be altered.

The NAT table

[3] Russell, Rusty and Welte, Harald. "Linux netfilter hacking howto." 2 July 2002 URL: http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO-3.html (25 Feb 2004)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The second table in the IPTables system is the nat table. Like the filter table, it has three chains in its default configuration. The first of these is the PREROUTING chain, which is called by the kernel before a routing decision is made about the packet. Naturally, another chain in the nat table is the POSTROUTING chain, which is called after routing decisions have been made about the packet. Finally there is the OUTPUT chain, which is processed on packets leaving the IPTables machine.[4]

The actions available in the nat table are similar to the filter table, with some additional options available that help facilitate its purpose. The standard ACCEPT, DROP, REJECT, and LOG targets are permitted. In addition, the SNAT, DNAT, and MASQUERADE targets are usable. The SNAT target causes the packet data to be manipulated in such a way as the source address is altered to its specifications. SNAT is only available in the POSTROUTING chain. DNAT operates similarly, but changes the destination address rather than the source. It is only available within the PREROUTING and OUTPUT chains. Addtionally, the REDIRECT target is available in the PREROUTING and OUTPUT chains. It simply alters the destination of the matched packet to the address of the IPTables system itself. Finally the MASQUERADE is available in the POSTROUTING chain. It works similarly to the SNAT target, but simply maps the outgoing packet to the IP address of the interface on which it is leaving. It also has the effect of dropping all of the existing MASQUERADE'd connections once that interface is downed. The MASQUERADE target is designed for NATing dynamic connections that change addresses frequently.[5]

The Mangle Table

Finally, IPTables includes the mangle table. The mangle table originally contained only the PREROUTING chain, and the OUTPUT chain. Since the release of kernel 2.4.18, the mangle table has added the INPUT, FORWARD, and POSTROUTING chains as well. These chains function similarly to their counterparts in other tables. PREROUTING is used to manipulate packets before a routing decision has been made. INPUT is used for packets destined for the machine on which IPTables is running. FORWARD is referred to when packets are being sent through the system rather than destined for it or originating from it. OUTPUT is used on packets that originate at the computer. Finally, POSTROUTING is used on packets after routing decisions have been made.[6]

[4] Eychenne, Herve "iptables man page" (9 Mar 2002) URL: http://node1.yo-linux.com/cgi-bin/man2html?cgi_command=iptables(8) 25 Feb 2004
[5] Eychenne, Herve "iptables man page" (9 Mar 2002) URL: http://node1.yo-linux.com/cgi-bin/man2html?cgi_command=iptables(8) 25 Feb 2004
[6] Eychenne, Herve "iptables man page" (9 Mar 2002) URL: http://node1.yo-linux.com/cgi-bin/man2html?cgi_command=iptables(8) 25 Feb 2004

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The mangle table has all of the same targets as the filter table, with the addition of three others. The first of these is the MARK target, which causes the packet to be flagged internally by Netfilter in such a way that the packet can be matched in other chains by specifying the mark. The second target is the TOS target, which allows manipulation of the Type of Service field in the IP header. Finally is the TTL target, which manipulates the Time To Live field of the packet header.[7]

A Packet's Journey: Traversing the Netfilter/IPTables Framework

When discussing paths through the IPTables framework, a single packet cannot be discussed. This is because IPTables is designed in such a way that packets with different source and destination addresses get treated differently based on those addresses, so three packets must be discussed. The first of these is a packet destined for the computer on which IPTables is running. This packet will be referred to as an input packet.

Input Packets

Once a packet arrives at the computer and simple sanity tests are performed, such as verifying checksums, it normally is passed to the kernel's routing code. In the case of an input packet, it is instead first passed to the mangle table's PREROUTING chain. At this point any matching rules in that chain are applied. IPTables rules are first match style rule sets. If a match occurs, processing of the rule set stops, and the packet is passed back to the networking code. In the case of an input packet, it is instead passed to the next table, nat. Input packets are next passed to the nat table's PREROUTING chain. Again any matching rules are processed. Next, the packet is processed through the filter table's INPUT chain. Finally the packet is processed through the mangle table's INPUT chain. Once these steps are completed, the packet is passed to its destination process, if any.[8]

Forward Packets

The next type of data to be considered is the packet which is being forwarded across two network interfaces in the IPTables machine, but for which the machine is not the destination. This packet will be described as the forward packet. Forward packets operate along a similar path as the input packets, but have additional chains through which they pass. They first move through the PREROUTING chains of the mangle and nat tables. Once a routing decision has been made, the packet traverses the FORWARD chain of the mangle and filter tables, respectively. Finally the packet will pass through the POSTROUTING

[7] Andreasson, Oskar. "Iptables Tutorial 1.1.19" http://www.faqs.org/docs/iptables/mangletable.html 25 Feb 2004

[8] Russell, Rusty and Welte, Harald. "Linux netfilter hacking howto." 2 July 2002 URL: http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO-3.html (25 Feb 2004)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

chains of the mangle and nat tables. Once this has been completed the packet is placed on the outgoing interface determined by the earlier routing decision.

Output Packets

The final packet type which must be considered is the output packet. It originates from the computer running the IPTables software, but is not destined for it. Once the packet reaches the network stack, the first action is that a routing decision is made. Once this has been completed, the packet is passed to the OUTPUT chain of the mangle and filter tables. Subsequently, the packet is sent through the POSTROUTING chains of the mangle and nat tables. Finally the packet is placed on the exiting interface.

Specialized Features of the Netfiler/IPTables Framework

Netfilter and IPTables sport numerous features that make them extremely useful in a networking environment, both from a security and a practicality standpoint. Illustrating this point, two of the more powerful features of the product will be defined, connection tracking and packet marking.

Connection Tracking

IPTables supports the tracking of entire datastreams, rather than simply filtering individual packets. This feature, called connection tracking, is implemented in such a fashion that even stateless protocols such as UDP can be tracked. This system is implemented as a set of memory tables in the kernel that the IPTables "ip_conntrack" module refers to for each packet traversing the filter. The module maintains state information based on source and destination IP addresses, source and destination ports, protocols, and timeouts. It does not simply track TCP states, thus allowing it to be used with other protocols such as UDP and ICMP.[9]

Each packet that crosses the filter is relegated to one of four states. NEW packets are packets that do not yet belong to an existing connection, but are otherwise valid. ESTABLISHED packets are part of an existing connection that has seen traffic in both directions. RELATED packets are those which are starting a new connection, but are associated with an existing connection. These packets can include such items as FTP data connections or ICMP error messages. Finally there are INVALID packets for which there is no existing connection, but which are also not attempting to start a new connection.

Connection tracking is extremely useful in a firewalling system. It is written in such a way as to facilitate the use of NAT. The ip_conntrack module attaches to the high priority NF_IP_PRE_ROUTING and NF_IP_POST_ROUTING hooks, as

[9] Stephens, James C. "IPtables: Connection Tracking" 2 Oct 2002 URL:http://www.sns.ias.edu/~jns/security/iptables/iptables_conntrack.html 25 Feb 2004

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

well as the NF_IP_LOCAL_OUT hook for packets originating at the machine running IPTables. This enables the module to see the packet data before manipulations are made. Thus an entry is made in the state table before any address manipulation occurs by the NAT code. This enables connection state to be maintained independently of changes to the source and destination addresses of the packet.[10]

Connection tracking also has advantages from a security perspective. This is best illustrated by way of example. The Tribe Flood Network master software communicates with its slaves through the use of ICMP Echo Reply packets with the commands and arguments embedded in the ID field and data portions of the packet.[11] In a simple packet filter scenario, ICMP echo reply packets from any source must be permitted through the firewall as that is the only way to receive valid replies from ICMP echo requests originating from inside the firewall. Thus TFN masters can easily communicate with their slaves in this scenario. In the case of an IPTables firewall employing connection tracking, the echo reply would be seen as INVALID, as it is not a NEW connection, nor is it RELATED to an existing connection. INVALID packets can simply be dropped at the firewall. Thus ICMP echo replies that are in response to requests from inside the firewall are permitted, but others are not. In the terminology of firewalls, this is generally referred to as 'stateful inspection' and is highly prized as a tool to improve network security.

Packet Marking

Within the mangle table a specialized target is permitted known as MARK. The MARK target allows for the creation of a flag on the packet that can be recognized both within other chains and tables, as well as by additional programs outside the Netfilter/IPTables framework that operate at the level of the network stack. This feature is highly useful in conjunction with the Linux iproute2 package to allow for the creation of policy based routing schemes. Using this feature to enable transparent proxies is an excellent example germane to the security arena.

In the PREROUTING chain of the mangle table, a rule may be set to mark packets with destination port 25 with the mark "2." The packet is otherwise unaltered, but is sent on to the routing engine. The Linux iproute2 package can then recognize the mark by using the "fwmark" tag and route the packet based on a routing table crafted specifically for that purpose. In this case the route table would have one entry, a default route to the SMTP proxy. Packets without the appropriate "fwmark" tag are routed according to the default routing table and

[10] Russell, Rusty and Welte, Harald. "Linux netfilter hacking howto." 2 July 2002 URL:http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO-4.html#ss4.3 (25 Feb 2004)

[11] Author Unknown. "CERT Incident Note IN-99-07: Distributed Denial of Service Tools." 15 Jan 2001 URL:http://www.cert.org/incident_notes/IN-99-07.html (25 Feb 2004)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

are otherwise unaffected. As is evident from this example, sophisticated policy based routing can be accomplished using these tools.[12]

Building the IPTables package

The complete Netfiler/IPTables system is comprised of two components. They are the Netfilter/IPTables kernel code, which is included in the Linux kernel 2.4 and 2.6 series. The kernel code is activated by building the appropriate modules during the kernel build process, which is largely outside the scope of this document. The modules are located in the "Networking Options → IP: Netfilter Configuration" section of the "make menuconfig" section of the kernel build process. There are a number of options available, including additional match targets outside the base installation such as the experimental UNCLEAN match which matches packets that are malformed. Another option is the Owner match that matches packets originating on the IPtables server with the uid of the owner of the process that created them.

The second component in the Netfilter/IPTables framework is the userspace utilities. These are not distributed as part of the kernel and must be retrieved separately. The base distribution point for these tools is the web page http://www.netfilter.org/downloads.html. At the time of this writing, the latest version of the IPTables system is 1.2.9. Thus the command:

wget http://www.netfilter.org/files/iptables-1.2.9.tar.bz2

will retrieve this package. The package may also be downloaded via web browser, ftp at ftp.netfilter.org, RSYNC or CVS. At this point the software must be uncompressed and extracted from its distribution format. This is accomplished with the commands:

```
bunzip2 iptables-1.2.9.tar.bz2
tar xvf iptables-1.2.9.tar
```

Once the appropriate package has been retrieved and extracted, the builder must execute the following commands within the directory created by the extraction process:

```
make KERNEL_DIR={kernel build location}
make install KERNEL_DIR={kernel build location}
```

These two commands build the IPTables userspace binaries and install them, respectively. Once this is completed and the kernel modules are installed, the package is ready for use.

[12] Narula, Ram "Transparent web-caching using netfilter, iproute2, ipchains and squid." URL: http://lartc.org/howto/lartc.cookbook.squid.html (25 Feb 2004)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

As an additional caveat, it is important to enable packet forwarding in the Linux kernel in order to ensure that network data is allowed to traverse interfaces. This is an often forgotten requirement that causes a great deal of frustration at the outset.

Designing IPTables based firewalls

Once a thorough understanding of Netfilter and IPTables is established, extremely effective firewalling rulesets can be established with very little effort. By way of example, a small office network will be used. The network is connected to the Internet via a commercial DSL line. This network uses the RFC 1918 reserved network of 192.168.1.0/24 for its internal systems, as additional public IP addresses are prohibitively expensive. For purposes of demonstration the public address of this network will be 10.10.1.1. The company requires that an internal web server and an SMTP server be exposed to the internet. The SMTP server requires access to send and receive email on TCP port 25. The web server requires TCP port 80 for purposes of delivering web requests. The Linux server that will be used is connected directly to the DSL router by way of a 10Mbit Ethernet link, and is connected to the company network through their 100Mbit Ethernet switch. The DSL connection will be referred to as “eth0” and the internal connection as “eth1.”

This rulebase will be built on the principles of least access, in which all traffic will be denied which is not explicitly allowed. To this end the following rules are created:

```
/sbin/iptables -P INPUT DROP
/sbin/iptables -P FORWARD DROP
```

This sets the default policy of the INPUT and FORWARD chains of the filter table to DROP. This effectively disallows all traffic through the system without the addition of exception rules. Note that the OUTPUT chain policy has not been altered. This is done on the assumption that traffic originating at the firewall is always valid. If this is not the case, then the assumption can be made that the firewall is already compromised, and thus the firewall rules are subject to question regardless.

Next a catchall table will be built for connection tracking, as well as to do simple scrubbing of the network traffic. To accomplish this the following commands are executed:

```
/sbin/iptables -N SCRUB
/sbin/iptables -A SCRUB -m state --state ESTABLISHED,RELATED -j ACCEPT
/sbin/iptables -A SCRUB -m state --state INVALID -j DROP
/sbin/iptables -A SCRUB -s 255.255.255.255 -i eth0 -j DROP
/sbin/iptables -A SCRUB -s 192.168.0.0/255.255.0.0 -i eth0 -j DROP
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
/sbin/iptables -A SCRUB -s 172.16.0.0/255.240.0.0 -i eth0 -j DROP
/sbin/iptables -A SCRUB -s 10.0.0.0/255.0.0.0 -i eth0 -j DROP
/sbin/iptables -A SCRUB -s 224.0.0.0/224.0.0.0 -i eth0 -j DROP
/sbin/iptables -A SCRUB -j RETURN
```

These rules accomplish several things simultaneously. First a user-defined chain has been built that can be attached to the system level chains. In this chain are created state rules that allow existing connections to pass unrestricted, and drop any connections that are invalid. Additionally a ruleset has been created that explicitly denies RFC 1918 reserved addresses access from the public interface. Having built this user defined chain, it is now attached to the filter table's INPUT and FORWARD chains with the following:

```
/sbin/iptables -A INPUT -j SCRUB
/sbin/iptables -A FORWARD -j SCRUB
```

These commands force all traffic coming to and being forwarded through the IPTables server to first pass through the new SCRUB chain. Using this method, a single table has been built through which explicit denial policies can be set without having to duplicate them to both chains. In addition, by including the state rules in this chain the number of rules have been reduced that must be added to the INPUT and FORWARD chains, eliminating duplication and consequently possible errors.

Once the default policy of the firewall has been set, and catchall rules have been built, it is then time to make the exception rules that permit the internal network access to the internet, and to allow internet access to the company's internal servers. First to be considered is the company's internal access. The company has strict access policies that prohibit any access from the user's desktops other than web access. Thus will be added the following rule to the FORWARD chain on the filter table:

```
/sbin/iptables -A FORWARD -i eth1 -s 192.168.1.0/24 -m tcp -p tcp --dport 80 -m
state --state NEW -j ACCEPT
```

It is important to note several things about this rule. "-A FORWARD" indicates that the rule is being added to the FORWARD chain of the default filter table. Also notice that the "-i eth0" and "-s 192.168.1.0/24" flags carefully define where the traffic must originate from in order to match. Next note that the "-m tcp -p tcp --dport 80" flags only allow TCP connections with a destination port of 80. Finally consider that the rule only accepts NEW connections. This is important to consider as the catchall rule was created earlier to match against ESTABLISHED connections. It is thus very important that all subsequent rules use the NEW match syntax so that they can be added to the conntrack tables and take advantage of this feature.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Now exception rules must be added for access from the Internet to the company's web and SMTP servers. These servers are addressed as 192.168.1.1 and 192.168.1.2, respectively. This is easily accomplished through the following rules:

```
/sbin/iptables -A FORWARD -i eth0 -d 192.168.1.1 -m tcp -p tcp --dport 80 -m state --state NEW -j ACCEPT
/sbin/iptables -A FORWARD -i eth0 -d 192.168.1.2 -m tcp -p tcp --dport 25 -m state --state NEW -j ACCEPT
```

These are similar to the web access rule above, but match against traffic flowing in the opposite direction. This can be seen in the "-i eth0" flags, which match only match against traffic arriving on the eth0 interface, the Internet facing NIC. Additionally note the "-d 192.168.1.2" flags designate a specific host rather than the entire network, as in the web access rule.

When looking at the above rules, it should be apparent that they will not successfully grant access to privately addressed resources alone. There must be additional NAT rules elsewhere to facilitate translation of these addresses into Internet routable format. This is by design. When the IPTables developers were originally planning out the project, they wanted a clear separation of the NAT code from the filter code. Thus the nat table is created independently of the filter table. Filter rules are written without consideration of the requirements of NAT. To allow for this separation, IPTables NAT features take advantage of the early and late attachments of the conntrack modules discussed earlier. Due to this excellent design, our NAT requirements can be reduced to three rules:

```
/sbin/iptables -t nat -A PREROUTING -i eth0 -d 10.10.1.1 -m tcp -p tcp --dport 80 -j DNAT --to-destination 192.168.1.1
/sbin/iptables -t nat -A PREROUTING -i eth0 -d 10.10.1.1 -m tcp -p tcp --dport 25 -j DNAT --to-destination 192.168.1.2
/sbin/iptables -t nat -A POSTROUTING -o eth0 -s 192.168.1.0/24 -j SNAT --to-source 10.10.1.1
```

The first two rules translate packets destined for port 80 and 25 to their internal addresses. They are appended to the PREROUTING chain of the nat table. The last translates packets in the other direction. It is added to the POSTROUTING chain of the nat table. This last rule allows both regular web browsing as well as the port redirection to the two internal servers. Note as before the usage of source and destination addresses, and incoming and outgoing interfaces as selecting points to determine what to do with the data. Also notice that state entries are not included in the nat table rules. This is because session state will be maintained by the other rules in the filter table.

As can be plainly seen in the above example, taking advantage of default policy, connection tracking, and NAT can result in very effective firewalls with a

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

minimum of effort. An effective small business firewall has been established with only nineteen IPTables rules. A simple packet filtering firewall would require many more rules and would be far less secure.

Conclusions

Based on the results of this examination, it can easily be seen that the Netfilter/IPTables framework provides a powerful toolkit for the development of network security systems. Powerful features, such as stateful inspection and packet mangling, combined with the natural and easy to understand layout, provide for a extremely useful tool. In spite of this initial ease, it is important to have a strong grasp of the true structure and function of the system. Armed with a thorough understanding of the framework, network administrators can use IPTables to generate comprehensive firewall rulesets with a minimum of effort.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

References:

1. Welte, Harald. “netfilter/iptables project homepage.” URL:http://www.netfilter.org/index.html (25 Feb 2004)

2. Russell, Rusty and Welte, Harald. “Linux netfilter hacking howto.” 2 July 2002 URL: http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO-3.html (25 Feb 2004)

3. Eychenne, Herve “iptables man page” (9 Mar 2002) URL: http://node1.yo-linux.com/cgi-bin/man2html?cgi_command=iptables(8) (25 Feb 2004)

4. Andreasson, Oskar. “Iptables Tutorial 1.1.19” http://www.faqs.org/docs/iptables/mangletable.html (25 Feb 2004)

5. Stephens, James C. “IPtables: Connection Tracking” 2 Oct 2002 URL:http://www.sns.ias.edu/~jns/security/iptables/iptables_conntrack.html (25 Feb 2004)

6. Author Unknown. “CERT Incident Note IN-99-07: Distributed Denial of Service Tools.” 15 Jan 2001 URL:http://www.cert.org/incident_notes/IN-99-07.html (25 Feb 2004)

7. Narula, Ram “Transparent web-caching using netfilter, iproute2, ipchains and squid.” URL: http://lartc.org/howto/lartc.cookbook.squid.html (25 Feb 2004)

8. Stephens, James C . “IPTables.” 5 Sep 2003 URL:http://www.sns.ias.edu/~jns/security/iptables/index.html (25 Feb 2004)

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Last Updated: June 25th, 2017

Upcoming Training

SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Phoenix SEC401	Phoenix, AZ	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Munich Summer 2017	Munich, Germany	Jul 10, 2017 - Jul 15, 2017	Live Event
Mentor Session - SEC401	Ventura, CA	Jul 12, 2017 - Sep 13, 2017	Mentor
Mentor Session - SEC401	Macon, GA	Jul 12, 2017 - Aug 23, 2017	Mentor
Community SANS Atlanta SEC401	Atlanta, GA	Jul 17, 2017 - Jul 22, 2017	Community SANS
SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
SANSFIRE 2017 - SEC401: Security Essentials Bootcamp Style	Washington, DC	Jul 24, 2017 - Jul 29, 2017	vLive
Community SANS Fort Lauderdale SEC401	Fort Lauderdale, FL	Jul 31, 2017 - Aug 05, 2017	Community SANS
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
Community SANS Omaha SEC401*	Omaha, NE	Aug 14, 2017 - Aug 19, 2017	Community SANS
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
Community SANS Trenton SEC401	Trenton, NJ	Aug 21, 2017 - Aug 26, 2017	Community SANS
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
Community SANS San Diego SEC401	San Diego, CA	Aug 21, 2017 - Aug 26, 2017	Community SANS
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
Mentor Session - SEC401	Minneapolis, MN	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Sep 06, 2017 - Oct 18, 2017	Mentor