How to Secure RHEL 6.2
Part 2

Motivation

This paper is part of a multi-part series on securing Redhat Enterprise Linux 6.2. This paper focuses on implementing IPtables as a host based firewall. If you need an enterprise class firewall, I would recommend first reading Next Generation Firewall for Dummies and researching as a starting place this company's products: (http://www.paloaltonetworks.com/products/overview/). The intended audience for this paper is laymen and journeymen system administrators that need a quick reference and solid solutions for common host based firewalls. If you have comments or feedback on how I can represent this better, please email me your ideas to the email address listed on the website (securityhardening.com).

Principles

Effectiveness of a firewall

Firewalls are only one part of the security of a system. Implemented correctly, they reduce the risk of a system or network being remotely compromised, but they do not remove all threat. Insiders are one of the greatest threats to an organization because of their ability to circumvent policy that is weakly implemented. Do not dismiss Firewalls, in this case we want to look at what a host based firewall can do to reduce the threat of remote attacks. Implemented correctly, the firewall can prevent direct connections from remote attacks.

Is it a router or a Firewall

For early Firewall's, they were performing the same functionality as a router. They routed packets and blocked based off of the implemented rules. The idea was to offload from the routing device and have one central location where most rules were located. This idea is at leat 20+ years old and does not keep up with the threats coming from the Internet today.

Top Down or Bottom Up reading of rules

This is an important point to understand. Most of the older firewalls read from top-down and used the first rule that it matched. Today's firewalls have optimization rules that put them into the most efficient order. For IPtables, it will be reading from top-down, so putting the deny-all rule at the bottom make the most sense.

Block all by default and allow only the known through the firewall

Allow only those services that you expressly permit. Think of a service as the end TCP or UDP port that you wish to connect to from a source IP address. Do not allow any direct TCP/IP connections between applications on internal systems and servers on the Internet. The notion that I operate under is that I never trust connections from un-trusted systems. Be aware that another system can impersonate a secure system. When attackers use this type of attack, they impersonate a trusted known host on your network. This impersonation, which is also called IP spoofing, allows the host to bypass your security controls to connect to your network. Finally, the routing table update that you receive from a neighboring router may redirect your network traffic to an unintended destination.

Point-to-Point connections

Establish a controlled link from trusted IP addresses. If I am writing a rule on a database server, I only wish to accept the database port as a destination port from the IP address of the web server(s). I would also then only accept SSH connections from the group of workstations that I use to manage and maintain the database server. What other ports would you need to have open? I would suggest none, but you know your organization better than I do and therefore must adapt to meet your organizational goals.

Deploying applications in a split zone

DMZ and Trusted zones with a firewall protecting the zones. DMZ and Trusted zones are still a worthwhile thought process at this point in time.

Using a screening router to offload some rules

If your organization has the money, I recommend to deploy a screening router on the outside of the firewall that allows known bad traffic to be discarded before getting to the main firewall. Such as preventing ICMP traffic originating from the internet to the internal network or firewall -- but allow ICMP outbound. Using an internal router to only allow trusted protocols in the Trusted zone (this can be performed on smart switches that understand layer 2-4 traffic).   Securing the switches to handle layer 4 traffic and limit only trusted ports and protocols (if a system is infected, it can't get past the switch port).

KISS -- Keep it Simple Stupid!

Keep the rules minimal and straight forward to meet your organizations goals while being usable. One of the stupidest things I've ever done was loading 600,000+ rules into 16MB of memory. The rules loaded and worked, but every packet going out had to be compared against this colossal set of rules, top-down and match one of the rules. My connections for an application like Internet Explorer would take at least 3-8 minutes just to start and connect to Google.com. This is unacceptable in my opinion. Minimize the rules and rely on the default block to protect the system. Use a holistic approach, such as hardening the Operating System, host based firewalls, anti-virus protection, regular log reviews, and back ground checks on all employees. It really comes down to a trust issue; how much do you really trust the people that maintain your systems? Keep the rules simple and do the right thing every day!

Implementation

Copy the below rules examples into /etc/sysconfig/iptables.

Restart the service with:
```
service iptables restart
```

Make sure that the service restarts on reboot:
```
chkconfig --level 2345 iptables on
```

Case Studies

securing a workstation

```
# Firewall configuration written by system-config-firewall
# Manual customization of this file is not recommended.
*filter
##
#       define default chains:
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
##
#       allow already created sessions:
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
##
#       accept loopback and ICMP traffic:
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
##
#       only accept inbound port 22 (SSH) connections.
-A INPUT -m state --state NEW -m tcp -s 192.168.139.0/24 -d 192.168.139.88 →
         -p tcp --dport 22 -j ACCEPT
##
#       block all other traffic.
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

The Accept line with "-s 192.168.139.0/24" allows the entire subnet to SSH into the node with destination IP address "-d 192.168.139.88". The character "→" means that the line wraps around.

securing a web server

```
# Firewall configuration written by system-config-firewall
# Manual customization of this file is not recommended.
*filter
##
#       define default chains:
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
##
#       allow already created sessions:
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
##
#       accept loopback and ICMP traffic:
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
##
#       accept inbound port 80 (HTTP) connections.
-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
##
#       accept inbound port 443 (HTTPS) connections.
-A INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT
##
#       only accept inbound port 22 (SSH) connections.
-A INPUT -m state --state NEW -m tcp -s 192.168.139.0/24 -d 192.168.139.88 →
         -p tcp --dport 22 -j ACCEPT
##
#       block all other traffic.
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

securing a database in a trusted zone

```
# Firewall configuration written by system-config-firewall
# Manual customization of this file is not recommended.
*filter
##
#       define default chains:
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
##
#       allow already created sessions:
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
##
#       accept loopback and ICMP traffic:
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
##
#       accept inbound database connections.
-A INPUT -m state --state NEW -m tcp -s 192.168.139.22 -d 192.168.139.88 →
         -p tcp --dport 1521 -j ACCEPT
##
#       only accept inbound port 22 (SSH) connections.
-A INPUT -m state --state NEW -m tcp -s 192.168.139.0/24 -d 192.168.139.88 →
         -p tcp --dport 22 -j ACCEPT
##
#       block all other traffic.
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

We want the database only allowed connections from a known web server, here it is: “-s 192.168.139.22”. Also, the dport shows 1521, which would be tied to Oracle DB. Adjust this to your needs.

securing a web server with a database on the same host

Use the rules for the web server and then set your database to operate on IP address 127.0.0.1. That way the communication is forced to operate on the host system alone. Change your web application to connect to the database on the 127.0.0.1 address.

Ultra Paranoid

Test these extensively before deploying in production.

```
# Firewall configuration written by system-config-firewall
# Manual customization of this file is not recommended.
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
#
-A INPUT -i lo -j ACCEPT
-A OUTPUT -o lo -j ACCEPT
#
-A INPUT -p icmp --icmp-type any -j ACCEPT
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -m state --state NEW -m tcp -s 192.168.139.1 -d 192.168.139.88 →
         -p tcp --dport 22 -j ACCEPT
-A OUTPUT -s 192.168.139.88 -d 192.168.139.1 -p tcp --dport 22 -j ACCEPT
-A INPUT -i eth0 -p tcp -m tcp --dport 22 -m state --state NEW →
         -m recent --set --name ssh_attempt --rsource →
         -j LOG --log-level 7 --log-prefix "SSH connection attempt: "
#
## Catch all others
-A INPUT -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

This was recommended by a friend to validate traffic inbound and outbound. It is a little overkill in my opinion, but then again, for the truly paranoid, it feeds their insecurities.

Add rate limiting

Test these extensively before deploying in production.

```
# Firewall configuration written by system-config-firewall
# Manual customization of this file is not recommended.
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
#
-A INPUT -i lo -j ACCEPT
-A OUTPUT -o lo -j ACCEPT
#
-A INPUT -p icmp --icmp-type any -j ACCEPT
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -m state --state NEW -m tcp -s 192.168.139.1 -d 192.168.139.88 →
         -p tcp --dport 22 -j LOG --log-level 7 --log-prefix "My_Message"
-A INPUT -m state --state NEW -m tcp -s 192.168.139.1 -d 192.168.139.88 →
         -p tcp --dport 22 -j ACCEPT
```

```
-A OUTPUT -s 192.168.139.88 -d 192.168.139.1 -p tcp --dport 22 -j ACCEPT
-A INPUT -i eth0 -p tcp -m tcp --dport 22 -m state --state NEW →
         -m recent --set --name ssh_attempt --rsource →
         -j LOG --log-level 7 --log-prefix "SSH connection attempt: "
### Limit the number of incoming connections on eth0
-A INPUT -i eth0 -p tcp -m tcp --dport 22 -m state --state NEW →
         -m recent --set --name ssh_attempt --rsource -j LOG →
         --log-level 7 -m limit --limit 2/second --limit-burst 5 →
         --log-prefix "SSH connection attempt: "
#
## Catch all others
-A INPUT -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

This can be used to rate limit incoming connections. Change the TCP port to your needs. Make sure you test this thoroughly before implementing in production.

Testing and Troubleshooting

Testing is absolutely essential when dealing with firewall rules. When you deploy a new application on RHEL 6.2, turn off the firewall rules to make sure the application works 100% correctly, then turn the firewall back on and test again. If something doesn't work, that means you need to add a new port number into the rules. Use the command, "netstat -an" to get a list of open ports with the firewall turned off and the application running. I like using the command "watch" to help with this, as in:

```
watch –d netstat -talpn
```

Output:

```
Every 2.0s: netstat -talpn
Sun Oct 28 16:00:17 2012

Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address               Foreign Address             State
PID/Program name
tcp        0      0 192.168.139.88:22           0.0.0.0:*                   LISTEN    1893/sshd
tcp        0      0 127.0.0.1:631               0.0.0.0:*                   LISTEN    1631/cupsd
tcp        0      0 127.0.0.1:25                0.0.0.0:*                   LISTEN    1977/master
tcp        0      0 192.168.139.88:22           192.168.139.1:3943          ESTABLISHED 2514/sshd
tcp        0      0 ::1:631                     :::*                        LISTEN    1631/cupsd
tcp        0      0 ::1:25                      :::*                        LISTEN    1977/master
```

Finally, my last tool to help is using the command "iptables" with the List flag.

```
iptables –L
```

Output:

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
ACCEPT     icmp --  anywhere             anywhere            icmp any
ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED
ACCEPT     tcp  --  192.168.139.1        192.168.139.88      state NEW tcp dpt:ssh
LOG        tcp  --  anywhere             anywhere            tcp dpt:ssh state NEW recent: →
           SET name: ssh_attempt side: source LOG level debug prefix →
           `SSH connection attempt: '
REJECT     all  --  anywhere             anywhere            reject-with icmp-host-prohibited

Chain FORWARD (policy DROP)
target     prot opt source               destination
```

```
Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED
ACCEPT     tcp  --  192.168.139.88       192.168.139.1       tcp dpt:ssh
```

I find it imperative when troubleshooting firewall rules to know exactly how the system is handling and using the rules. Monitor your log files, /var/log/messages, to understand what's being blocked and adjust the rules to fit your organizational policy.

Conclussion

Deploying the local firewall will prevent almost all unwanted remote connections. There is a slight management overhead introduced with these rules, but the benefit outweighs the cost.   These are in no way the magic silver bullet to protect everything and need to again be combined with "a holistic approach, such as hardening the Operating System, host based firewalls, anti-virus protection, regular log reviews, and back ground checks on all employees". Failure to implement a full set of deterrence's will result in a weaker target that will cost the organization time, money and possibly customers or embarrassing data loss highlighted on the nightly news and worst case, in front of congress.