

# iptables

- iptables/netfilter is a framework for packet filtering and mangling
- Successor of ipchains (2.2.x) and ipfwadm (2.0.x)
- Consists of chains within tables (filter [default], nat, mangle)

# iptables – brief rundown

- Chains are a checklist of rules and what to do if matched
- kernel gets packet
  - meant for machine ? – INPUT chain
  - produced by machine and headed out ? OUTPUT chain
  - headed for another machine? FORWARD chain
  - traverses the chain until it hits a rule that accepts it and has something for it to do
  - packet doesn't match any rules, consults chain's default policy

# iptables – basic commands

- -N new chain
- -A append to chain
- -L list rules in chain
- -P sets default policy on chain
- -X delete chain
- -F flushes chain (rules)

```
/sbin/iptables -N newchain
```

```
/sbin/iptables -A newchain -p tcp --dport 22 -j ACCEPT
```

```
/sbin/iptables -P newchain DROP
```

```
/sbin/iptables -X newchain
```

# iptables - targets (-j, --jump)

- ACCEPT
  - Accepts packet, continues rule traversal
- DROP
  - Drops the packet (ends rule traversal), significant delay in port scanning
- REJECT
  - sends back an error packet, ends rule traversal (can use `–reject-with`)

```
/sbin/iptables -A INPUT -p tcp -i eth0 -dport 23 -j REJECT  
–reject-with tcp-reset
```

# iptables - proof

```
/sbin/iptables -N tcp_chain
```

```
/sbin/iptables -N accepted
```

```
/sbin/iptables -A accepted -p tcp --syn -j ACCEPT
```

```
/sbin/iptables -A accepted -p tcp -m state --state  
ESTABLISHED,RELATED -j ACCEPT
```

```
/sbin/iptables -A accepted -p tcp -j DROP
```

```
/sbin/iptables -A tcp_chain -p tcp -s 0/0 --dport 22 -j  
accepted
```

# iptables - tips, tricks

```
iptables -A OUTPUT -m owner --cmd-owner acroread -j DROP
```

```
iptables -N dynamicdrop
```

```
iptables -A dynamicdrop -m random --average 50 -j DROP
```

```
iptables -A dynamicdrop -j REJECT
```

**<http://www.faqs.org/docs/iptables/>**

# ssh

- encrypted network connectivity tools
- a 'secure' replacement for the old BSD 'r' tools, rlogin, rsh, rcp
- de facto way of 'logging in' to remote servers nowadays

# ssh – good practice

- Manually validate all public keys for servers you have local access to!
- caching public keys
- ssh is susceptible to a MITM attack, by those sitting on your network, via packet and arp spoofing

# ssh – manual validation

```
[jsimpson@mapserver ~]$ ssh nist
```

```
The authenticity of host 'nist (192.168.168.39)' can't be
  established.
```

```
RSA key fingerprint is
```

```
  94:66:6a:89:c1:9c:3a:4a:4f:96:b8:5b:c4:72:ec:d4.
```

```
Are you sure you want to continue connecting (yes/no)?
```

```
...
```

```
jsimpson@nist:/etc/ssh$ ssh-keygen -l -f ssh_host_rsa_key.pub
```

```
1024 94:66:6a:89:c1:9c:3a:4a:4f:96:b8:5b:c4:72:ec:d4
  ssh_host_rsa_key.pub
```

# ssh – tips, tricks

- Disable root login

```
/etc/ssh/sshd_config:
```

```
#PermitRootLogin yes -> PermitRootLogin no
```

```
/etc/init.d/ssh restart
```

```
service sshd restart
```

- Enable compression with '-C'
- Fun: `ssh -l user server /bin/bash -i`

# ssh – limiting users

- How do I limit who can and can't use ssh?
- Use PAM
- Add to `/etc/pam.d/ssh` (or `sshd`, depends on distro):

```
auth required pam_listfile.so onerr=fail  
item=user sense=allow file=/etc/ssh_users
```

- Then add relevant users to `/etc/ssh_users`

# ssh – limiting users (brief PAM)

- *auth*
  - the *module-type*. *auth* prompts for password, or some other authentication, and can also grant credentials (e.g. groups, etc)
- *required*
  - the *control-flag*. *required* means the module must succeed for the *module-type* to succeed.
- *pam\_listfile.so*
  - the module itself. *path* defaults to */lib/security*. There are quite a few different modules, each with it's own argument parameters.

# ssh – limiting users (brief PAM)

- *onerr=fail item=user sense=allow file=/etc/ssh\_users*
- Arguments are specific to their relative module (in this case, pam\_listfile)
  - *onerr* : if for some reason the module fails to run, file can't be found, etc. In this case, it fails (returning an error), if the argument was 'succeed', it'd return PAM\_SUCCESS
  - *item* : this is obviously user, could be group, shell, or others
  - *sense / file* : those in the *file* specified are applied '*sense*' (i.e., you could have *deny* instead of *allow* to keep certain users unable to use ssh)

# More on PAM

**<http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam.html>**