Active Defense

What:
Strategies employed to prevent, obstruct, or otherwise block unwanted access to a system or network.
**Active Defense**

**What:**
Strategies employed to prevent, obstruct, or otherwise block unwanted access to a system or network.

**How:**
- **Firewalls:** parse packet information and take action based on some rules
- **VPN:** encrypt traffic, even packet headers
- **Proxy server:** controls access, hides topology, adds visibility to application traffic
- **Configuration and choice of network services:**
  - ftp servers can be vulnerable but secure ftp (SFTP) is less vulnerable and is used in a similar way
  - telnet must be used carefully
  - rsh must be used carefully
- **Permissions:** set uid must be used very carefully
Active Defense

Firewall
Firewalls

What They Can Do:
Parse inbound and outbound packets
Test parsed results against criteria defined in tables
Block packets or allow the packets to continue
Generate reports and alerts
Generate log files for future analysis

What They Cannot Do:
Deal with lapses in security practices and policies
  e.g. an employee might bypass a firewall by attaching a modem to an office phone and dialing in!
Cannot protect an entire network – only a single point
  e.g. the attacker may be an insider
  (VPN will help with this)
Generally cannot authenticate except by ip address
  (so we use something like kerberos for that)
Generally cannot authorize the use of a service
Firewalls

Performance Issues:
- Must buffer traffic to allow time for a decision to be made.
- Maximum hi-end throughput is typically 800 Mbps.
- Typically TCP traffic occurs in bursts but buffer size may be too small – either release the traffic before analysis or get larger buffers but there is still a throughput problem.
Firewalls

Typical Architecture:
Indexed on src addr/port, dest addr/port
Says some connection has established

129.137.4.132:80  192.168.1.109:58959
Firewalls

**Typical Architecture:**
Indexed on src addr/port, dest addr/port
Says some connection has established

129.137.4.132:80 192.168.1.109:58959

Firewall Rule Set

incoming packet

Protocol State Table

<table>
<thead>
<tr>
<th>http</th>
</tr>
</thead>
<tbody>
<tr>
<td>icmp</td>
</tr>
<tr>
<td>http</td>
</tr>
<tr>
<td>ssh</td>
</tr>
<tr>
<td>imap</td>
</tr>
<tr>
<td>http</td>
</tr>
</tbody>
</table>
Firewalls

Typical Architecture:
Indexed on src addr/port, dest addr/port
Says some connection has established

129.137.4.132:80  192.168.1.109:58959
incoming packet proceeds on match
Firewalls

Typical Architecture:
Indexed on src addr/port, dest addr/port
Says some connection has established

129.137.4.132:80  192.168.1.109:58959

129.137.4.132:80  192.168.1.109:58959  ssh

incoming packet

Firewall Rule Set

Protocol State Table

http
icmp
http
ssh
imap
http
**Firewalls**

**Typical Architecture:**
Indexed on src addr/port, dest addr/port
Says some connection has established

129.137.4.132:80  192.168.1.109:58959

Diagram:
- **Firewall Rule Set**
- **Protocol State Table**
  - http
  - icmp
  - http
  - ssh
  - imap
  - http

Incoming packet: 129.137.4.132:80  192.168.1.109:58959  ssh
Firewalls

Typical Architecture:
Indexed on src addr/port, dest addr/port
Says some connection has established

129.137.4.132:80  192.168.1.109:58959

incoming packet   OK to proceed
add an entry to the state table

Firewall Rule Set

Protocol State Table

http
icmp
http
ssh
ssh
imap
http
Firewalls

**Typical Architecture:**
Indexed on src addr/port, dest addr/port
Says some connection has established

129.137.4.132:80  192.168.1.109:58959

**Protocol State Table**

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Rule Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>http</td>
<td></td>
</tr>
<tr>
<td>icmp</td>
<td></td>
</tr>
<tr>
<td>http</td>
<td></td>
</tr>
<tr>
<td>ssh</td>
<td></td>
</tr>
<tr>
<td>imap</td>
<td></td>
</tr>
<tr>
<td>http</td>
<td></td>
</tr>
</tbody>
</table>
**Firewalls**

**Typical Architecture:**
Indexed on src addr/port, dest addr/port
Says some connection has established

129.137.4.132:80  192.168.1.109:58959

Incoming packet – rule set does not permit packet – it is dropped

**Protocol State Table**

- http
- icmp
- http
- ssh
- imap
- http

**Firewall Rule Set**
Firewalls

Typical Architecture:
Indexed on src addr/port, dest addr/port
Says some connection has established

129.137.4.132:80  192.168.1.109:58959

consider a full table

Firewall Rule Set

Protocol State Table

http
icmp
http
ssh
ssh
imap
http
Firewalls

**Typical Architecture:**
Indexed on src addr/port, dest addr/port
Says some connection has established

129.137.4.132:80 192.168.1.109:58959

incoming packet – must drop – no room for a new connection in table

Firewall Rule Set

**Protocol State Table**

- http
- icmp
- http
- ssh
- ssh
- imap
- http
Firewalls

Performance Problems:

1. Not fast enough – too much overhead due to lookups
   - but parallelization makes this problem moot

2. Buffer size may not be large enough

3. May reach the maximum table size

4. Loss of service:
   - firewall times a connection out after, say, 15 minutes
   - firewall may not have removed the table entry
   - firewall may not re-establish a connection that it thinks
     may be due to a hijack (policy decision)

5. Advanced features may not be available
   - since firewall can see protocols, it can check packet
     payloads for sanity or for some patterns that may be
     of interest to the organization – but this is not common
Firewalls

Performance Problems:
6. Some applications, for example science and medical:
   Few TCP connections, long idle periods between bursts
   Transport large quantities of data to some analysis center
   – so lots of packets with similar headers and different
     payloads and traffic is bursty!

7. But firewall designs are for many connections
   Plus firewalls typically have many processing engines, all
   operating at speeds lower than the interface speed

   Thus, TCP data bursts, handled by one of the processing
   engines will lose packets unless the firewall buffer
   is large enough, which it probably is not!!

Finally, if connection timeouts are too short and idle times
are too long, which they probably are, more packets will
be lost
Firewalls

Performance Problems:
Example: traffic between two points, top figure with firewall 
bypasses firewall
Network Performance Tool

nuttcp:
Nice net performance tool – put a server on one computer and push traffic to the server from another

Server (192.168.1.112):
```
nuttcp -S -p 8000 --nofork
```
Server listening on port 8000, not forked

Client:
```
nuttcp -T 10 -i 1 -p 8000 192.168.1.112
```
Client sends 10 packets, one a second, to port 8000 of server

See next page for sample output
Network Performance Tool

nuttcp:

[fanco@franco ~]$ nuttcp -T 10 -i 1 -p 8000 192.168.1.112

1.2500 MB / 1.00 sec = 10.4851 Mbps 0 retrans
1.2500 MB / 1.00 sec = 10.4857 Mbps 0 retrans
1.3125 MB / 1.00 sec = 11.0093 Mbps 0 retrans
1.3750 MB / 1.00 sec = 11.5355 Mbps 0 retrans
1.5000 MB / 1.00 sec = 12.5826 Mbps 0 retrans
1.3750 MB / 1.00 sec = 11.5346 Mbps 0 retrans
1.5000 MB / 1.00 sec = 12.5824 Mbps 0 retrans
1.4375 MB / 1.00 sec = 12.0586 Mbps 0 retrans
1.4375 MB / 1.00 sec = 12.0591 Mbps 0 retrans
1.3750 MB / 1.00 sec = 11.5340 Mbps 0 retrans

13.9977 MB / 10.10 sec = 11.6274 Mbps 0 %TX 4 %RX 0
retrans 2.40 msRTT
Router with Access Control List

Why:
ACLs are usually implemented in the router's hardware, do not compromise performance of sci/med applications.
Firewalls

**iptables (Linux):**
places rules into predefined chains that are checked against any network traffic relevant to those chains what to do with packet based on outcome of those rules
Firewalls

iptables (Linux):
places rules into predefined chains that are checked against any network traffic relevant to those chains what to do with packet based on outcome of those rules

chains:
INPUT: All packets destined for the host computer.
OUTPUT: All packets originating from the host computer.
FORWARD: All packets neither destined for nor originating from the host computer, but routed by the host computer. (if computer is used as a router)
Firewalls

iptables (Linux):
places rules into predefined chains that are checked against any network traffic relevant to those chains what to do with packet based on outcome of those rules

chains:
INPUT: All packets destined for the host computer.
OUTPUT: All packets originating from the host computer.
FORWARD: All packets neither destined for nor originating from the host computer, but routed by the host computer. (if computer is used as a router)

rules:
added in a list to a chain
packet is checked against each rule in turn
a decision is made – if it is 'drop' then done otherwise, perform action associated with the rule
Iptables

Default policies within chains:

- can set default to DROP then add rules to ACCEPT packets to/from trusted IP addresses and ports
- can set default to ACCEPT then add rules to DROP packets to/from machines, nets, etc

Usually, packets leaving our net are trusted so OUTPUT policy is likely to be ACCEPT first.

INPUT policy is likely to be DROP first but watch out if setting policy remotely – you could prevent traffic from you!!

Other actions:

- REJECT - drop the packet, notify sender, stop processing rules in this chain.
- LOG - Log the packet, continue processing rules in this chain allows adding annotations like log level
iptables primer:

Load modules:

```
sudo modprobe iptable_filter
sudo modprobe x_tables
```

Check status of the firewall:

```
sudo service ufw status
ufw stop/waiting
```

Start the firewall:

```
sudo service ufw start
ufw start/running
```

Stop the firewall:

```
sudo service ufw stop
ufw stop/waiting
```
iptables primer:
See rules that exist (I do not have any rules – just starting):

```
sudo iptables -L
```

Chain INPUT (policy ACCEPT)
**target** prot opt source destination

Chain FORWARD (policy ACCEPT)
**target** prot opt source destination

Chain OUTPUT (policy ACCEPT)
**target** prot opt source destination

Meaning:
**target:** an action such as DROP or ACCEPT
**prot:** protocol (icmp, udp, tcp, etc.) to check
**opt:** transmission parameters – long string needs decoding
iptables primer:

A simple rule:

```bash
sudo iptables -A INPUT -m conntrack \ 
--ctstate ESTABLISHED,RELATED -j ACCEPT
```

Meaning:

- **-A INPUT**: append this rule to the INPUT chain
- **-m conntrack**: filter rules match based on connection state
- **--ctstate ...**: valid states to match on as follows:
  - NEW – connection has not yet been set
  - RELATED – new but related to an existing connection
  - ESTABLISHED – connection has been established
  - INVALID – traffic cannot be identified
- **-j ACCEPT**: jump to the ACCEPT target

This allows established connections to receive traffic
iptables primer:

A simple rule:

```
sudo iptables -A INPUT -p TCP \n--dport ssh -j ACCEPT
```

Meaning:
- `-A INPUT`: append this rule to the INPUT chain
- `-p TCP`: TCP protocol
- `--dport ssh`: port 22
- `-j ACCEPT`: jump to the ACCEPT target

This allows incoming traffic on port 22 for ssh
iptables primer:

A simple rule:
```bash
sudo iptables -A INPUT -p TCP --dport 80 -j ACCEPT
```

Meaning:
- `-A INPUT`: append this rule to the INPUT chain
- `-p TCP`: TCP protocol
- `--dport 80`: port 80
- `-j ACCEPT`: jump to the ACCEPT target

This allows incoming web traffic
iptables primer:

three simple rules:

```bash
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
sudo iptables -A INPUT -p TCP --dport ssh -j ACCEPT
sudo iptables -A INPUT -p TCP --dport 80 -j ACCEPT
```

Effect:

```bash
sudo iptables -L
```

Chain INPUT (policy ACCEPT)

<table>
<thead>
<tr>
<th>target</th>
<th>prot</th>
<th>opt</th>
<th>source</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>all</td>
<td>--</td>
<td>anywhere</td>
<td>anywhere ctstate RELATED,ESTABLISHED</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>tcp</td>
<td>--</td>
<td>anywhere</td>
<td>anywhere tcp dpt:ssh</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>tcp</td>
<td>--</td>
<td>anywhere</td>
<td>anywhere tcp dpt:www</td>
</tr>
</tbody>
</table>
iptables primer:

Drop all packets:

```bash
sudo iptables -A INPUT -j DROP
```

Effect:

```bash
sudo iptables -L
```

Chain INPUT (policy ACCEPT)

<table>
<thead>
<tr>
<th>target</th>
<th>prot</th>
<th>opt</th>
<th>source</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>all</td>
<td>--</td>
<td>anywhere</td>
<td>anywhere</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>tcp</td>
<td>--</td>
<td>anywhere</td>
<td>anywhere</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>tcp</td>
<td>--</td>
<td>anywhere</td>
<td>anywhere</td>
</tr>
<tr>
<td>DROP</td>
<td>all</td>
<td>--</td>
<td>anywhere</td>
<td>anywhere</td>
</tr>
</tbody>
</table>
iptables

iptables primer:

Add a rule to the front of the chain:

```sh
sudo iptables -I INPUT -i lo -j ACCEPT
```

Effect:

```sh
sudo iptables -L
```

Chain INPUT (policy ACCEPT)

<table>
<thead>
<tr>
<th>target</th>
<th>prot</th>
<th>opt</th>
<th>source</th>
<th>destination</th>
<th>ctstate RELATED,ESTABLISHED</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>all</td>
<td>--</td>
<td>anywhere</td>
<td>anywhere</td>
<td>ctstate RELATED,ESTABLISHED</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>all</td>
<td>--</td>
<td>anywhere</td>
<td>anywhere</td>
<td>ctstate RELATED,ESTABLISHED</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>tcp</td>
<td>--</td>
<td>anywhere</td>
<td>anywhere</td>
<td>tcp dpt:ssh</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>tcp</td>
<td>--</td>
<td>anywhere</td>
<td>anywhere</td>
<td>tcp dpt:www</td>
</tr>
<tr>
<td>DROP</td>
<td>all</td>
<td>--</td>
<td>anywhere</td>
<td>anywhere</td>
<td></td>
</tr>
</tbody>
</table>
iptables

iptables primer:

Add a rule to the front of the chain:

```
sudo iptables -I INPUT -i lo -j ACCEPT
```

Effect:

```
sudo iptables -L -v
```

Chain INPUT (policy ACCEPT 3 packets, 915 bytes)

```
pkts bytes target prot opt in  out source    destination
  0      0  ACCEPT all  --  lo  any  anywhere  anywhere
355 141K  ACCEPT all  --  any  any  anywhere  anywhere  ctstate RELATED,ESTABLISHED
  0      0  ACCEPT tcp  --  any  any  anywhere  anywhere  tcp dpt:ssh
```
iptables primer:

Log dropped packets:
```bash
sudo iptables -I INPUT 5 -m limit \
--limit 5/min -j LOG --log-prefix "whoa" \
--log-level 7
```

Meaning:
- `-I INPUT 5` : put this rule in position 5 (just before DROP)
- `-m limit` : allows the rule to match only a limited # of times
- `--limit 5/min` : at most 5 of these per minute logged
- `--log-prefix` : string at the beginning of the log entries
- `--log-level` : syslog log level (say 7)
iptables primer:

To save the tables for reboot:

```
sudo sh -c "iptables-save > /etc iptables.rules"
```

Then edit `/etc/network/interfaces`:

```
pre-up iptables-restore < /etc/iptables.rules
```

Put this line at the end of the section concerning the interface that applies – usually eth0

To restore:

See https://help.ubuntu.com/community/IptablesHowTo
iptables primer:

Another example:

```
iptables -P INPUT ACCEPT
iptables -F
iptables -A INPUT -p TCP --dport 6881 -j ACCEPT
iptables -A INPUT -p TCP --dport 6881:6890 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -s 192.168.1.0/24 -j ACCEPT
iptables -P OUTPUT ACCEPT
```

1\textsuperscript{st} line: set policy to INPUT chain to ACCEPT

2\textsuperscript{nd} line: flush all the rules – with 1\textsuperscript{st} line can connect remotely

3\textsuperscript{rd} line: accept bittorrent packets

4\textsuperscript{th} line: range of ports
iptables primer:

Limit new inbound TCP packets to prevent DoS (new Chain):

```bash
iptables -t nat -N syn-flood
iptables -t nat -A syn-flood -m limit \ 
    --limit 12/sec --limit-burst 24 -j RETURN
Iptables -t nat -A syn-flood -j DROP
Iptables -t nat -A PREROUTING -i eth0 -d 10.0.1.9 \ 
    -p tcp --syn -j syn-flood
```

Limit new inbound TCP connections having packets with SYN bit set to 12 per second after 24 connections per second have been seen.

-t nat:

This table is consulted when a packet that creates a new connection is encountered. It has three built-ins: PREROUTING (alter packets as soon as they come in), OUTPUT (alter locally-generated packets before routing), POSTROUTING (alter packets as they are about to go out).
iptables primer:

Block inbound port scans:

```sh
iptables -t nat -A PREROUTING -i eth0 -d 10.0.1.9 \\ -m psd -j DROP
```

-t nat:

This table is consulted when a packet that creates a new connection is encountered. It has three built-ins: PREROUTING (alter packets as soon as they come in), OUTPUT (alter locally-generated packets before routing), POSTROUTING (alter packets as they are about to go out).
iptables

iptables primer:

Drop packets from hosts with > 16 active connections:

```bash
iptables -t nat -A PREROUTING -i eth0 -p tcp --syn
    -d 10.0.1.9 -m iplimit --iplimit-above 16
    -j DROP
```

-t nat:
This table is consulted when a packet that creates a new connection is encountered. It has three built-ins:
PREROUTING (alter packets as soon as they come in), OUTPUT (alter locally-generated packets before routing), POSTROUTING (alter packets as they are about to go out).
iptables primer:

Drop packets related to CodeRed and Nimda viruses:

```
iptables -t filter -A INPUT -i eth0 -p tcp \ 
  -d 10.0.1.9 --dport http -m string \ 
  --string "/default.ida?" -j DROP

iptables -t filter -A INPUT -i eth0 -p tcp \ 
  -d 10.0.1.9 --dport http -m string \ 
  --string ".exe?/c+dir" -j DROP

iptables -t filter -A INPUT -i eth0 -p tcp \ 
  -d 10.0.1.9 --dport http -m string \ 
  --string ".exe?/c+tftp" -j DROP
```

-t filter:

The default table (if no -t option is passed). It contains the built-in chains INPUT (for packets destined to local sockets), FORWARD (for packets being routed through the box), and OUTPUT (for locally-generated packets).