Worksheet 9

Linux as a router, packet filtering, traffic shaping
Linux as a router

- Capable of acting as a router, firewall, traffic shaper
- (so are most other modern operating systems)
- Tools:
  - netfilter/iptables
  - tc
Netfilter / Iptables

- The Linux Packet filtering framework
- 2 axes of organisation:
  - Chains - *when* does the interception occur?
  - Tables - *what* can be done (functionality)?
Graphical Overview

- **PREROUTING**
  - Local?
    - yes → **INPUT**
    - no → **FORWARD**
  - Local?
    - no → **OUTPUT**
  - Local?
    - yes → **POSTROUTING**
  - Local?
    - yes → Socket API

**Flowchart Diagram:**

- **PREROUTING**
- **Local?** (diamond shape)
- **INPUT**
- **FORWARD**
- **OUTPUT**
- **POSTROUTING**
- **Socket API**
Iptables: Chains

- Chains - when?
  - **PREROUTING** - before remote/local decision
  - **INPUT** - before locally destined traffic is admitted
  - **OUTPUT** - before locally generated traffic is routed
  - **FORWARD** - non-local packets
  - **POSTROUTING** - before packet leaves system
Iptables: Tables

- Tables: functionality
  - **filter (default)** - block packets
    on **INPUT, OUTPUT, FORWARD**
  - **nat** - change packet src/dst address/port
    on **PREROUTING, POSTROUTING**
  - ...

### The Matrix - common uses

<table>
<thead>
<tr>
<th></th>
<th>PRE-ROUTING</th>
<th>INPUT</th>
<th>OUTPUT</th>
<th>FORWARD</th>
<th>POST-ROUTING</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td></td>
<td>filter</td>
<td>filter</td>
<td>filter</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>incoming</td>
<td>outgoing</td>
<td>forwarded</td>
<td></td>
</tr>
<tr>
<td>nat</td>
<td>DNAT</td>
<td></td>
<td></td>
<td></td>
<td>SNAT</td>
</tr>
<tr>
<td>mangle</td>
<td>mark, manipulate packets</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Anatomy of a chain

- Essentially a list of tuples \(<\text{pattern}, \text{target}\)>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>-s 10.1.2.3</td>
<td>ACCEPT</td>
</tr>
<tr>
<td>-d 10.2.3.4 -p tcp --dport 3306</td>
<td>DROP</td>
</tr>
<tr>
<td>-s 10.2.3.4 -p tcp --dport 22 -m state --state=NEW</td>
<td>LOG</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- First match wins!
Iptables invocation

- To add a rule to a chain:
  ```bash
  iptables [-t <TABLE>] -A <CHAIN> <PATTERN> -j <TARGET>
  ```

- List existing rules:
  ```bash
  iptables [-t <TABLE>] -L [<CHAIN>] [-v] [-n]
  ```

- Delete a rule from a chain:
  ```bash
  iptables [-t <TABLE>] -D <CHAIN> rule num
  ```

- As always: `man iptables` is your friend
## Some rule patterns

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-s 1.2.3.4</code></td>
<td>from source IP 1.2.3.4</td>
</tr>
<tr>
<td><code>-d 1.2.3.5</code></td>
<td>to destination IP 1.2.3.5</td>
</tr>
<tr>
<td><code>-p tcp</code></td>
<td>protocol tcp</td>
</tr>
<tr>
<td><code>tcp/udp: --[sd]port 80</code></td>
<td>src/destination port 80</td>
</tr>
<tr>
<td><code>icmp: --icmp-type echo-request</code></td>
<td><code>ping</code> echo request</td>
</tr>
</tbody>
</table>
Some rule targets

<table>
<thead>
<tr>
<th>ACCEPT</th>
<th>accept packet for this stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP</td>
<td>drop packet immediately</td>
</tr>
<tr>
<td></td>
<td>(and silently)</td>
</tr>
<tr>
<td>LOG</td>
<td>log packet to syslog</td>
</tr>
<tr>
<td>REJECT</td>
<td>drop packet and send an</td>
</tr>
<tr>
<td></td>
<td>ICMP error message to the</td>
</tr>
<tr>
<td></td>
<td>source</td>
</tr>
</tbody>
</table>
Stateful Filtering

- Problem of stateless filters: Related packets flow in both directions - how to correlate
  - TCP - can look at TCP-state (and rely on the TCP state of the protected host to behave properly)
  - UDP - stateless...

- How would you create a rule that matches „Answers to DNS queries that were sent out“?
- => Stateful Filtering
Stateful Filtering: Principles

- The firewall tracks and maintains higher layer communication state
  - "A has sent out a DNS query to B and is expecting an answer"
- Rules can be built that match the protocol / correspondence state
Stateful Filtering in iptables

- Rules match communication state
  ```
  ... -m state --state NEW|ESTABLISHED|RELATED
  ```
- State automatically tracked by the `conntrack` module
  - TCP state
  - UDP <src_ip,srcport,dst_ip, dst_port> tuples w/ timeout
  - application specific helper modules (FTP)
Traffic Shaping

- limit bandwidth allocation to specific classes of service
- by nature of the Internet: can only limit what you send, not what you receive
- ... but most of the bulky traffic will adapt! (TCP Slowstart)
The principle: Token bucket

- bucket can hold $b$ tokens
- tokens generated at rate $r$ token/sec unless bucket full
- only send packet if you have a token

Kurose, Ross
Token buckets in Linux

- tc can be used with the **tbf** (token bucket) **qdisc** (queuing discipline) to limit throughput on an interface:

```bash
tc qdisc add dev <DEV> root tbf rate <rate>kbit latency <latency>ms burst <burst_rate>kbit
```

- Parameters:
  - **rate** maximum allowed average bandwidth
  - **burst** - maximum allowed burst bandwidth
Classful Trafficshaping: HTB

- HTB := Hierarchical Token Bucket
- Can define a hierarchy of traffic classes, and assign limits
  - rate - the average allowed bandwidth
  - ceil - burst bandwidth allowed when buckets are present
  - prio - priority for spare bandwidth - classes with lower prios are offered the bandwidth first
Deploying HTB (I)

1. Enable `qdisc` (Queuing discipline) for the device and define a root class handle (1:0)

```bash
tc qdisc add dev <DEVICE> root handle 1:0
htb default <default_class>
```

2. Define a class (1:10 here)

```bash
tc class add dev <DEVICE> parent 1:0 classid 1:10
htb rate 100kbit ceil 150kbit prio 0
```
Deploying HTB (II)

3. Mark packets that should belong to the class, using iptables’ mangle facility (there is other ways, but follow me on this)

4. Stuff marked packets with x into class x and assign to appropriate qdisc.

```bash
iptables -A POSTROUTING -t mangle <PATTERN> -j MARK --set-mark 10

tc filter add dev <DEV> parent 1:0 prio 0 protocol ip handle 10 fw flowid 1:10
```
That’s all!

- Worksheet 9 is due
  
  Friday, July 3th, 2009, 08:00 am