NAME

uruk − wrapper for Linux iptables, for managing firewall rules

SYNOPSIS

uruk

DESCRIPTION

uruk loads an *rc* file (see **uruk-rc(5)**) which defines network service access policy, and invokes **iptables(8)** to set up firewall rules implementing this policy. By default the file `/etc/uruk/rc` is used; one can overrule this by specifying another file in the URUK_CONFIG environment variable. Under some circumstances, it's useful to use another command for iptables; this can be achieved by setting the URUK_IPTABLES (and/or URUK_IP6TABLES) environment variables. See **uruk-rc(5)** for details.

QUICK SETUP GUIDE

Uruk will *not* "just work" out of the box. It needs manual configuration. For those of you who don't like reading lots of documentation:

```
# cp /usr/share/doc/uruk/examples/rc \
    /etc/uruk/rc
# vi /etc/uruk/rc
# urukctl start
```

GETTING STARTED

Once the **uruk** script is installed, you want to go use it, of course. We'll give a detailed description of what to do here.

First, create an *rc* file. See **uruk-rc(5)** for info on how to do this. Once this file is created and installed (this script looks in `/etc/uruk/rc` by default), you're ready to run **uruk**. You might want to test your *rc* file by running **uruk** in debug mode, see **uruk-rc(5)**. There are at least 3 ways to load your *rc* file. We'll first describe a low level one: using vanilla iptables.

Vanilla iptables

After editing *rc*, load your rules like this. First flush your current rules:

```
# iptables −F
# ip6tables −F
```

Then enable your *rc* rules

```
# uruk
```

. Inspect the rules by doing:

```
# iptables −L
# ip6tables −L
```

. If you want to make these changes survive a reboot, use the init script as shipped with this package. If you'd rather write your own init script, the **iptables-restore(8)** and **iptables-save(8)** commands from the iptables package might be helpful.

Using the Uruk init script

Assumed is the Uruk init script is installed as explained in the README file. Optionally, install `/etc/default/uruk` (or `/etc/sysconfig/uruk`) and tweak it. An example file is in `/usr/share/doc/uruk/examples/default` (You might like to enable support for **uruk-save**.) Now activate uruk by doing:

```
# urukctl start
```

Now your pre-uruk iptables rules (if any) are saved as the "inactive" ruleset. While executing `urukctl start`, your box is open during a short while. If you don't like this, read below about **uruk-save**.

When rebooting, everything will be fine: `/etc/init.d/uruk` stores state in `/var/lib/uruk/iptables`, using iptables-save(8), which comes with Linux iptables.

Using Debian ifupdown

In case you have just one network interface which should get protected, you could use **interfaces(5)** from the Debian ifupdown package instead of the init script. Suppose you'd like to protect `ppp0`, and would like not to interfere with traffic on eth0: your other network interface. First write an *rc* file. Be sure it features

```
interfaces_unprotect="lo eth0"
```

Then run:

```
# mkdir -p /var/lib/uruk/iptables

# iptables -F

# iptables-save -c > /var/lib/uruk/iptables/down
# uruk
# iptables-save -c > /var/lib/uruk/iptables/up
```

Add

```
pre-up iptables-restore < /var/lib/uruk/iptables/up
post-down iptables-restore < /var/lib/uruk/iptables/down
```

to your interfaces stanza, in your `/etc/network/interfaces` .

Similar tricks might be possible on GNU/Linux systems from other distributions. The author is interested.

LOADING A NEW *rc* FILE

Need to change your rules?

Using the Uruk init script

Do

```
# vi /etc/uruk/rc
# urukctl force-reload
```

While executing `urukctl force-reload`, your box is open during a short while. If you don't like this, read below about **uruk-save**.

THE GORY DETAILS: uruk INTERNALS

The **uruk** script works like (and looks like) the list of statements below. Of course, take a look at `/sbin/uruk` for the final word on the workings.

1

rc is sourced as a shell script

2

Traffic on $interfaces_unprotect (just lo per default) is trusted:

```
$iptables -A INPUT -i $iface -j ACCEPT
```

3
$rc_a is sourced as a shell script, or, in case $rc_a is a directory, all files matching $rc_a/*.rc are sourced as shell scripts

4
ESTABLISHED and RELATED packets are ACCEPT-ed:

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

5
$rc_b is sourced

6
$interfaces gets protected against spoofing: we don't allow anyone to spoof non-routeable addresses. We block outgoing packets that don't have our address as source: they are either spoofed or something is misconfigured (NAT disabled, for instance). We want to be nice and don't send out garbage.

```
$iptables -A INPUT -i $iface --source $no_route_ip \
 -j DROP
```

We drop all incoming packets which don't have us as destination:

```
$iptables -A OUTPUT -o $iface --source ! "$ip" \
 -j DROP
```

And we always allow outgoing connections:

```
$iptables -A OUTPUT -m conntrack --ctstate NEW -o $iface \
 -j ACCEPT
```

7
$rc_c is sourced

8
Allow traffic to offered services, from trusted sources:

```
$iptables -A INPUT -m conntrack --ctstate NEW \
 -i $iface --protocol $proto --source "$source" \
 --destination "$ip" --destination-port "$port" \
 -j ACCEPT
```

9
$rc_d is sourced

10
Don't answer broadcast and multicast packets:

```
$iptables -A INPUT -i $iface --destination "$bcast" \
 -j DROP
```

11
$rc_f is sourced

12

Explicitly allow a subset of the ICMP types. (We disallow all other traffic later.)

```
$iptables -A INPUT --protocol icmp --icmp-type $type \
  -j ACCEPT
```

13

$rc_g is sourced

14

Log packets (which make it till here)

```
$iptables -A INPUT -j LOG --log-level debug \
  --log-prefix 'iptables: '
```

15

$rc_h is sourced

16

Reject all other packets

```
$iptables -A INPUT -j REJECT
```

17

$rc_i is sourced

USING uruk-save AS THE INITSCRIPT BACKEND

By default, **uruk-save** is not used by the uruk init script. You might want to use it, though. The **uruk-save** script is faster and when using **uruk-save**, your box won't be open while loading new rules. But beware: **uruk-save** is not as robust as using **uruk** itself.

The script **urukctl** (and thus the uruk init script) will use **uruk-save** only if asked to do so in `/etc/default/uruk` (or `/etc/sysconfig/uruk`). If this file features

```
enable_uruk_save=true
```

uruk-save is used whenever appropriate. See **uruk-save(8)** for more details.

DEFAULT POLICY

By default, **uruk** drops packets which have unknown RFC 1918 private network addresses in their source or destination.

It rejects packets with source nor destination for one of our IPs.

Packets belonging to locally initiated sessions are allowed: we match state; the local host can act as a client for any remote service.

By default, **uruk** drops all ICMP packets (except those for interfaces in $interfaces_unprotect) with type other than

- address-mask-reply
- address-mask-request
- destination-unreachable (this is a catch-all for a lot of types)
-

- echo-request
- echo-reply
- parameter-problem (catch-all for ip-header-bad and required-option-missing)
- timestamp-reply
- timestamp-request
- ttl-zero-during-transit
- ttl-zero-during-reassembly

By default, the FORWARD chain is left untouched, so has policy ACCEPT. (This won't do much harm, since packet forwarding is disabled by default in the Linux kernel. However, if you don't mind being paranoid, you might want to add

```
iptables --policy FORWARD REJECT
```

to your $rc_a uruk hook. See **uruk-rc(5)**.)

By default, **uruk** logs all UDP and TCP packets which are blocked by the user defined policies. Loglevel is debug, logprefix is "iptables:". See also the notes on *loglevel* in **uruk-rc(5)**.

Blocked TCP packets are answered with a tcp-reset.

WARNING

In order to keep the **uruk** script small and simple, the script does very little error handling. It does not check the contents of the *rc* file in any way before executing it. When your *rc* file contains bogus stuff, **uruk** will very likely behave in unexpected ways. Caveat emptor.

ENVIRONMENT

You can override some defaults in the shell before executing the uruk script. **uruk** honors the following variables:

- "URUK_CONFIG" Full pathname of *rc* file; `/etc/uruk/rc` by default.
- "URUK_IPTABLES" Full pathname of iptables executable. `/sbin/iptables` by default. Overrides *iptables*.
- "URUK_IP6TABLES" Full pathname of ip6tables executable, for IPv6 support. Overrides *ip6tables*.
- "URUK_INTERFACES_UNPROTECT" Default list of unprotected interfaces. Overrides *interfaces_unprotect*. The default default is `lo`.

SEE ALSO

uruk-rc(5), **uruk-save(8)**. The Uruk homepage is at http://mdcc.cx/uruk/ .

iptables(8), **iptables-save(8)**, **iptables-restore(8)**, **ip6tables(8)**, **ip6tables-save(8)**, **ip6tables-restore(8)**, http://www.netfilter.org/

interfaces(5), http://packages.debian.org/ifupdown.

COPYRIGHT

AUTHOR

Joost van Baal-IliÄ <joostvb-uruk@mdcc.cx>