16:08 Naming Network Interfaces

by Cornelius Diekmann

There are only two hard things in Computer Science: misogyny and naming things. Sometimes they are related, though this article only digresses about the latter, namely the names of the beloved network interfaces on our Linux machines. Some neighbors stick to the boring default names, such as `lo`, `eth0`, `wlan0`, or `ens1`. But what names does the mighty kernel allow for interfaces? The Linux kernel specifies that any byte sequence which is not too long, has neither whitespace nor colons, can be pointed to by a `char*`, and does not cause problems when interpreted as filename, is okay.[29]

The church of weird machines praises this nice and clean recognition routine. The kernel is not even bothering its deferential user with character encoding; interface names are just plain bytes.

```
# ip link set eth0 name \
    $(echo -ne 'lol\x01\x02\x03\x04\x05yolo')
$ ip addr | xxd
6c6f 6c01 0203 0405 79 6f 6c6f   lol.....yolo
```

For convenience, our time-honoured terminals interpret byte sequences according to our local encoding, also featuring terminal escapes.

```
# ip link set eth0 name \
    $(echo -ne '\e[31m☃\e[0m')
```

Given a contemporary color display, the user can enjoy a happy red snowman.

For the uplink to the Internet (with capital I), I like to call my interface "+".

```
# ip link set eth1 name +
```

Having decided on fine interface names, we obviously need to protect ourselves from the evil haxXx0rs in the Internet. Yet, our happy red snowman looks innocent and we are sure that no evil will ever come from that interface.

```
# iptables -I INPUT -i + -j DROP
# iptables -A INPUT \
  -i $(echo -ne '\e[31m☃\e[0m') -j ACCEPT
```

Hitting enter, my machine is suddenly alone in the void, not even talking to my neighbors over the happy red snowman interface.

```
# iptables-save
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -j DROP
-A INPUT -i ☃ -j ACCEPT
COMMIT
```

Where did the match "`-i +`" in the first rule go? Why is it dropping all traffic, not just the traffic from the evil Internet?

The answer lies, as envisioned by the prophecy of LangSec, in a mutual misunderstanding of what an interface name is. This misunderstanding is between the Linux kernel and netfilter/iptables. iptables has almost the same understanding as the kernel, except that a "`+`" at the end of an interface's byte sequence is interpreted as a wildcard. Hence, iptables and the Linux kernel have the same understanding about "☃", "`eth0`", and "`eth+++0`", but not about "`eth+`". Ultimately, iptables interprets "`+`" as "any interface." Thus, having realized that iptables match expressions are merely Boolean predicates in conjunctive normal form, we found universal truth in "`-i +`". Since tautological subexpressions can be eliminated, "`-i +`" disappears.

But how can we match on our interface "`+`" with a vanilla iptables binary? With only the minor inconvenience of around 250 additional rules, we can match on all interfaces which are not named "`+`".

```
#!/bin/bash
iptables -N PLUS
iptables -A INPUT -j PLUS
for i in $(seq 1 255); do
  B=$(echo -ne "\x$(printf '%02x' $i)")
  if [ "$B" != '+' ] && [ "$B" != ' ' ] \
     && [ "$B" != "" ]; then
    iptables -A PLUS -i "$B+" -j RETURN
  fi
done
iptables -A PLUS -m comment \
  --comment 'only + remains' -j DROP
iptables -A INPUT \
  -i $(echo -ne '\e[31m☃\e[0m') -j ACCEPT
```

[29] See Figure 3.

```
/* dev_valid_name - check if name is okay for network device
 * @name: name string
 *
 *  Network device names need to be valid file names to allow sysfs to work.  We also
 *  disallow any kind of whitespace.
 */
bool dev_valid_name(const char *name){
        if (*name == '\0')
                return false;
        if (strlen(name) >= IFNAMSIZ)
                return false;
        if (!strcmp(name, ".") || !strcmp(name, ".."))
                return false;

        while (*name) {
                if (*name == '/' || *name == ':' || isspace(*name))
                        return false;
                name++;
        }
        return true;
}
EXPORT_SYMBOL(dev_valid_name);
```

Figure 3. `net/core/dev.c` from Linux 4.4.0.

As it turns out, iptables 1.6.0 accepts certain chars in interfaces the kernel would reject, in particular tabs, dots, colons, and slashes.

With great interface names comes great responsibility, in particular when viewing `iptables-save`. Our esteemed paranoid readers likely never print any output on their terminals directly, but always pipe it through `cat -v` to correctly display non-printable characters. But can we do any better? Can we make the firewall faster and the output of `iptables-save` safe for our terminals?

The rash reader might be inclined to opine that the heretic folks at netfilter worship the golden calf of the almighty "+" character deep within their hearts and code. But do not fall for this fallacy any further! Since the code is the window to the soul, we shall see that the fine folks at netfilter are pure in heart. The overpowering semantics of "+" exist just in userspace; the kernel is untainted and pure. Since all bytes in a `char[]` are created equal, I shall venture to banish this unholy special treatment of "+" from my userland.

```
--- iptables-1.6.0_orig/libxtables/xtables.c
+++ iptables-1.6.0/libxtables/xtables.c
@@ -532,10 +532,7 @@
   strcpy(vianame, arg);
   if (vialen == 0)
     return;
- else if (vianame[vialen - 1] == '+') {
-   memset(mask, 0xFF, vialen - 1);
-   /* Don't remove `+' here! -HW */
- } else {
+ else {
    /* Include nul-terminator in match */
    memset(mask, 0xFF, vialen + 1);
    for (i = 0; vianame[i]; i++) {
```

With the equality of chars restored, we can finally drop those packets.

```
# iptables -A INPUT -i + -j DROP
```

Happy naming and many pleasant encounters with all the naïve programs on your machine not anticipating your fine interface names.