
Subject: Re: unlock iptables in netns

Posted by [Patrick McHardy](#) on Mon, 16 Jun 2008 10:26:03 GMT

[View Forum Message](#) <> [Reply to Message](#)

Patrick McHardy wrote:

> Alexey Dobriyan wrote:

>> Hi,

>>

>> Den basically banned iptables in netns via this patch

>>

>> --- a/net/netfilter/core.c

>> +++ b/net/netfilter/core.c

>> ...

>> , however, at least some of netfilter pieces are ready for usage in netns

>> and it would be nice to unlock them before release.

>>

>> If I'm deciphering chengelog correctly it's all about code which does

>> nf_register_hook{,s} but not netns-ready itself:

>>

>> br_netfilter.c

>> iptable_mangle (via ip_route_me_harder)

>> conntracking (both IPv4 and IPv6)

>> NAT

>> ...

>> Patch above can be applied and we can mark above list as "depends

>> !NET_NS"

>> and move on.

>>

>> Comments? Den, was there something else you're afraid of?

>

>

> That might result in some bad surprises for people how have already

> turned on NET_NS. I'd prefer a way that doesn't potentially disable

> half the netfilter options in existing configs.

By the way, is there already work done for conntrack/NAT namespace support? I have this patch that uses marks for something very similar that should be easy to adjust.

```
diff --git a/include/net/netfilter/nf_conntrack_tuple.h b/include/net/netfilter/nf_conntrack_tuple.h
index e69ab2e..49c4d0c 100644
--- a/include/net/netfilter/nf_conntrack_tuple.h
+++ b/include/net/netfilter/nf_conntrack_tuple.h
@@ -91,6 +91,8 @@ struct nf_conntrack_tuple
```

```

/* The direction (for tuplehash) */
u_int8_t dir;
} dst;
+
+ u_int32_t mark;
};

struct nf_conntrack_tuple_mask
@@ -140,7 +142,8 @@ static inline int __nf_ct_tuple_src_equal(const struct nf_conntrack_tuple
*t1,
    t1->src.u3.all[2] == t2->src.u3.all[2] &&
    t1->src.u3.all[3] == t2->src.u3.all[3] &&
    t1->src.u.all == t2->src.u.all &&
-   t1->src.l3num == t2->src.l3num);
+   t1->src.l3num == t2->src.l3num &&
+   t1->mark == t2->mark);
}

static inline int __nf_ct_tuple_dst_equal(const struct nf_conntrack_tuple *t1,
@@ -151,7 +154,8 @@ static inline int __nf_ct_tuple_dst_equal(const struct nf_conntrack_tuple
*t1,
    t1->dst.u3.all[2] == t2->dst.u3.all[2] &&
    t1->dst.u3.all[3] == t2->dst.u3.all[3] &&
    t1->dst.u.all == t2->dst.u.all &&
-   t1->dst.protonum == t2->dst.protonum);
+   t1->dst.protonum == t2->dst.protonum &&
+   t1->mark == t2->mark);
}

static inline int nf_ct_tuple_equal(const struct nf_conntrack_tuple *t1,
@@ -187,7 +191,8 @@ static inline int nf_ct_tuple_src_mask_cmp(const struct
nf_conntrack_tuple *t1,
    return 0;

    if (t1->src.l3num != t2->src.l3num ||
-       t1->dst.protonum != t2->dst.protonum)
+       t1->dst.protonum != t2->dst.protonum ||
+       t1->mark != t2->mark)
        return 0;

    return 1;
diff --git a/net/ipv4/netfilter/nf_conntrack_l3proto_ipv4.c
b/net/ipv4/netfilter/nf_conntrack_l3proto_ipv4.c
index a65b845..7b50593 100644
--- a/net/ipv4/netfilter/nf_conntrack_l3proto_ipv4.c
+++ b/net/ipv4/netfilter/nf_conntrack_l3proto_ipv4.c
@@ -52,9 +52,10 @@ static int ipv4_invert_tuple(struct nf_conntrack_tuple *tuple,
static int ipv4_print_tuple(struct seq_file *s,

```

```

    const struct nf_conntrack_tuple *tuple)
{
- return seq_printf(s, "src=%u.%u.%u.%u dst=%u.%u.%u.%u ",
+ return seq_printf(s, "src=%u.%u.%u.%u dst=%u.%u.%u.%u mark=%u ",
    NIPQUAD(tuple->src.u3.ip),
-   NIPQUAD(tuple->dst.u3.ip));
+   NIPQUAD(tuple->dst.u3.ip),
+   tuple->mark);
}

/* Returns new sk_buff, or NULL */
diff --git a/net/ipv4/netfilter/nf_nat_core.c b/net/ipv4/netfilter/nf_nat_core.c
index 36b4e3b..a2e76dc 100644
--- a/net/ipv4/netfilter/nf_nat_core.c
+++ b/net/ipv4/netfilter/nf_nat_core.c
@@ -82,7 +82,7 @@ hash_by_src(const struct nf_conntrack_tuple *tuple)
 /* Original src, to ensure we map it consistently if poss. */
 hash = jhash_3words((__force u32)tuple->src.u3.ip,
    (__force u32)tuple->src.u.all,
-   tuple->dst.protonum, 0);
+   tuple->dst.protonum ^ tuple->mark, 0);
 return ((u64)hash * nf_nat_hhtable_size) >> 32;
}

@@ -140,7 +140,8 @@ same_src(const struct nf_conn *ct,
t = &ct->tuplehash[IP_CT_DIR_ORIGINAL].tuple;
return (t->dst.protonum == tuple->dst.protonum &&
 t->src.u3.ip == tuple->src.u3.ip &&
- t->src.u.all == tuple->src.u.all);
+ t->src.u.all == tuple->src.u.all &&
+ t->mark == tuple->mark);
}

/* Only called for SRC manip */
@@ -213,7 +214,7 @@ find_best_ips_proto(struct nf_conntrack_tuple *tuple,
minip = ntohl(range->min_ip);
maxip = ntohl(range->max_ip);
j = jhash_2words((__force u32)tuple->src.u3.ip,
-   (__force u32)tuple->dst.u3.ip, 0);
+   (__force u32)tuple->dst.u3.ip ^ tuple->mark, 0);
j = ((u64)j * (maxip - minip + 1)) >> 32;
*var_ipp = htonl(minip + j);
}

diff --git a/net/ipv6/netfilter/nf_conntrack_l3proto_ipv6.c
b/net/ipv6/netfilter/nf_conntrack_l3proto_ipv6.c
index 3717bdf..633b7bc 100644
--- a/net/ipv6/netfilter/nf_conntrack_l3proto_ipv6.c
+++ b/net/ipv6/netfilter/nf_conntrack_l3proto_ipv6.c

```

```

@@ -56,9 +56,10 @@ static int ipv6_invert_tuple(struct nf_conntrack_tuple *tuple,
static int ipv6_print_tuple(struct seq_file *s,
    const struct nf_conntrack_tuple *tuple)
{
- return seq_printf(s, "src=" NIP6_FMT " dst=" NIP6_FMT " ",
+ return seq_printf(s, "src=" NIP6_FMT " dst=" NIP6_FMT " mark=%u ",
    NIP6(*((struct in6_addr *)tuple->src.u3.ip6)),
-   NIP6(*((struct in6_addr *)tuple->dst.u3.ip6)));
+   NIP6(*((struct in6_addr *)tuple->dst.u3.ip6)),
+   tuple->mark);
}

/*
diff --git a/net/netfilter/nf_conntrack_core.c b/net/netfilter/nf_conntrack_core.c
index b77eb56..f515a06 100644
--- a/net/netfilter/nf_conntrack_core.c
+++ b/net/netfilter/nf_conntrack_core.c
@@ -83,7 +83,7 @@ static u_int32_t __hash_conntrack(const struct nf_conntrack_tuple *tuple,
n = (sizeof(tuple->src) + sizeof(tuple->dst.u3)) / sizeof(u32);
h = jhash2((u32 *)tuple, n,
rnd ^ (((__force __u16)tuple->dst.u.all << 16) |
- tuple->dst.protonum));
+ (tuple->dst.protonum ^ tuple->mark));

return ((u64)h * size) >> 32;
}
@@ -112,6 +112,7 @@ nf_ct_get_tuple(const struct sk_buff *skb,
tuple->dst.protonum = protonum;
tuple->dst.dir = IP_CT_DIR_ORIGINAL;
+ tuple->mark = skb->mark;

return l4proto->pkt_to_tuple(skb, dataoff, tuple);
}
@@ -160,8 +161,8 @@ nf_ct_invert_tuple(struct nf_conntrack_tuple *inverse,
return 0;

inverse->dst.dir = !orig->dst.dir;
-
inverse->dst.protonum = orig->dst.protonum;
+ inverse->mark = orig->mark;
return l4proto->invert_tuple(inverse, orig);
}
EXPORT_SYMBOL_GPL(nf_ct_invert_tuple);
diff --git a/net/netfilter/nf_conntrack_expect.c b/net/netfilter/nf_conntrack_expect.c
index 684ec9c..19791a3 100644
--- a/net/netfilter/nf_conntrack_expect.c
+++ b/net/netfilter/nf_conntrack_expect.c

```

```

@@ -81,7 +81,7 @@ static unsigned int nf_ct_expect_dst_hash(const struct nf_conntrack_tuple
*tuple
}

hash = jhash2(tuple->dst.u3.all, ARRAY_SIZE(tuple->dst.u3.all),
-    (((tuple->dst.protonum ^ tuple->src.l3num) << 16) |
+    (((tuple->dst.protonum ^ tuple->src.l3num ^ tuple->mark) << 16) |
     (_force __u16)tuple->dst.u.all) ^ nf_ct_expect_hash_rnd);
return ((u64)hash * nf_ct_expect_hsize) >> 32;
}
@@ -222,6 +222,7 @@ struct nf_conntrack_expect *nf_ct_expect_alloc(struct nf_conn *me)
return NULL;

new->master = me;
+ new->tuple.mark = me->tuplehash[IP_CT_DIR_ORIGINAL].tuple.mark;
atomic_set(&new->use, 1);
INIT_RCU_HEAD(&new->rcu);
return new;

```
