Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by Patrick McHardy on Wed, 27 Jun 2007 12:51:47 GMT
View Forum Message <> Reply to Message

Vasily Averin wrote:
> Patrick McHardy wrote:
>
>>>+ for (i = 0; i < NF_CT_EVICTION_RANGE; i++) {
>>>+ hlist_for_each_entry(h, n, &nf_conntrack_hash[hash], hnode) {
>>>+   tmp = nf_ct_tuplehash_to_ctrack(h);
>>>+   if (!test_bit(IPS_ASSURED_BIT, &tmp->status))
>>>+   ct = tmp;
>>>+ }
>>>+ if (ct) {
>>>+   atomic_inc(&ct->ct_general.use);
>>>+   break;
>>>+ }
>>>+ hash = (hash + 1) % nf_conntrack_htable_size;
>
>
> it is incorrect,
> We should count the number of checked _conntracks_, but you count the number of
> hash buckets. I.e "i" should be incremented/checked inside the nested loop.

I misunderstood your patch then. This one should be better.

[NETFILTER]: nf_conntrack: early_drop improvement

When the maximum number of conntrack entries is reached and a new
one needs to be allocated, conntrack tries to drop an unassured
connection from the same hash bucket the new conntrack would hash
to. Since with a properly sized hash the average number of entries
per bucket is 1, the chances of actually finding one are not very
good. This patch increases those chances by walking over the hash
until 8 entries are checked.

Based on patch by Vasily Averin <vvs@sw.ru>.

Signed-off-by: Patrick McHardy <kaber@trash.net>

---
commit df9f4fc41d7d6a7a51d2fe4b28db2557cb9a0d05
tree 8beb115ce12126b28ce3e5eb3f95b36b71462ea5
parent 665d98d03473cab252830129f414e1b38fb2b038
author Patrick McHardy <kaber@trash.net> Wed, 27 Jun 2007 14:51:38 +0200
committer Patrick McHardy <kaber@trash.net> Wed, 27 Jun 2007 14:51:38 +0200

```
 net/netfilter/nf_conntrack_core.c |   23 +++++++++++++++++++--------
 1 files changed, 15 insertions(+), 8 deletions(-)

diff --git a/net/netfilter/nf_conntrack_core.c b/net/netfilter/nf_conntrack_core.c
index d7e62ad..bbb52e5 100644
--- a/net/netfilter/nf_conntrack_core.c
+++ b/net/netfilter/nf_conntrack_core.c
@@ -377,21 +377,29 @@ nf_conntrack_tuple_taken(const struct nf_conntrack_tuple *tuple,
 }
 EXPORT_SYMBOL_GPL(nf_conntrack_tuple_taken);

+#define NF_CT_EVICTION_RANGE 8
+
 /* There's a small race here where we may free a just-assured
    connection.  Too bad: we're in trouble anyway. */
-static int early_drop(struct hlist_head *chain)
+static int early_drop(unsigned int hash)
 {
  /* Use oldest entry, which is roughly LRU */
  struct nf_conntrack_tuple_hash *h;
  struct nf_conn *ct = NULL, *tmp;
  struct hlist_node *n;
- int dropped = 0;
+ unsigned int i;
+ int dropped = 0, cnt = NF_CT_EVICTION_RANGE;

  read_lock_bh(&nf_conntrack_lock);
- hlist_for_each_entry(h, n, chain, hnode) {
-  tmp = nf_ct_tuplehash_to_ctrack(h);
-  if (!test_bit(IPS_ASSURED_BIT, &tmp->status))
-   ct = tmp;
+ for (i = 0; i < nf_conntrack_htable_size; i++) {
+  hlist_for_each_entry(h, n, &nf_conntrack_hash[hash], hnode) {
+   tmp = nf_ct_tuplehash_to_ctrack(h);
+   if (!test_bit(IPS_ASSURED_BIT, &tmp->status))
+    ct = tmp;
+   if (--cnt <= 0)
+    break;
+  }
+  hash = (hash + 1) % nf_conntrack_htable_size;
  }
  if (ct)
   atomic_inc(&ct->ct_general.use);
@@ -425,8 +433,7 @@ struct nf_conn *nf_conntrack_alloc(const struct nf_conntrack_tuple *orig,
  if (nf_conntrack_max
      && atomic_read(&nf_conntrack_count) > nf_conntrack_max) {
   unsigned int hash = hash_conntrack(orig);
-  /* Try dropping from this hash chain. */
```

```
-  if (!early_drop(&nf_conntrack_hash[hash])) {
+  if (!early_drop(hash)) {
    atomic_dec(&nf_conntrack_count);
    if (net_ratelimit())
     printk(KERN_WARNING
```