Subject: [PATCH 2.6.21-rc6] [netfilter] early_drop imrovement
Posted by vaverin on Fri, 06 Apr 2007 08:00:29 GMT
View Forum Message <> Reply to Message

When the number of conntracks is reached ip_conntrack_max limit, early_drop() is called and tries to free one of already used conntracks in one of the hash buckets. If it does not find any conntracks that may be freed, it leads to transmission errors.
However it is not fair because of current hash bucket may be empty but the neighbour ones can have the number of conntracks that can be freed. With the following patch early_drop() will search conntracks in all hash buckets.

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
--- 2.6.21-rc6/net/ipv4/netfilter/ip_conntrack_core.c.erdrp
+++ 2.6.21-rc6/net/ipv4/netfilter/ip_conntrack_core.c
@@ -517,7 +517,7 @@ ip_conntrack_tuple_taken(const struct ip

 /* 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 list_head *chain)
+static int __early_drop(struct list_head *chain)
 {
  /* Traverse backwards: gives us oldest, which is roughly LRU */
  struct ip_conntrack_tuple_hash *h;
@@ -547,6 +547,20 @@ static int early_drop(struct list_head *
  return dropped;
 }

+static int early_drop(const struct ip_conntrack_tuple *orig)
+{
+ unsigned int i, hash;
+ int ret = 0;
+
+ hash = hash_conntrack(orig);
+
+ for (i = 0;
+    !ret && i < ip_conntrack_htable_size;
+    ++i, hash = ++hash % ip_conntrack_htable_size)
+  ret = __early_drop(&ip_conntrack_hash[hash]);
+ return ret;
+}
+
 static struct ip_conntrack_helper *
 __ip_conntrack_helper_find( const struct ip_conntrack_tuple *tuple)
 {
@@ -631,9 +645,7 @@ struct ip_conntrack *ip_conntrack_alloc(
```

```
  if (ip_conntrack_max
      && atomic_read(&ip_conntrack_count) > ip_conntrack_max) {
-  unsigned int hash = hash_conntrack(orig);
-  /* Try dropping from this hash chain. */
-  if (!early_drop(&ip_conntrack_hash[hash])) {
+  if (!early_drop(orig)) {
     atomic_dec(&ip_conntrack_count);
     if (net_ratelimit())
      printk(KERN_WARNING
--- 2.6.21-rc6/net/netfilter/nf_conntrack_core.c.erdrp
+++ 2.6.21-rc6/net/netfilter/nf_conntrack_core.c
@@ -542,7 +542,7 @@ EXPORT_SYMBOL_GPL(nf_conntrack_tuple_tak

 /* 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 list_head *chain)
+static int __early_drop(struct list_head *chain)
 {
  /* Traverse backwards: gives us oldest, which is roughly LRU */
  struct nf_conntrack_tuple_hash *h;
@@ -572,6 +572,20 @@ static int early_drop(struct list_head *
  return dropped;
 }

+static int early_drop(const struct nf_conntrack_tuple *orig)
+{
+ unsigned int i, hash;
+ int ret = 0;
+
+ hash = hash_conntrack(orig);
+
+ for (i = 0;
+   !ret && i < nf_conntrack_htable_size;
+   ++i, hash = ++hash % nf_conntrack_htable_size)
+  ret = __early_drop(&nf_conntrack_hash[hash]);
+ return ret;
+}
+
 static struct nf_conn *
 __nf_conntrack_alloc(const struct nf_conntrack_tuple *orig,
        const struct nf_conntrack_tuple *repl,
@@ -591,9 +605,7 @@ __nf_conntrack_alloc(const struct nf_con

  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(orig)) {
   atomic_dec(&nf_conntrack_count);
   if (net_ratelimit())
    printk(KERN_WARNING
```

Subject: Re: [PATCH 2.6.21-rc6] [netfilter] early_drop imrovement
Posted by Eric Dumazet on Fri, 06 Apr 2007 08:24:33 GMT
View Forum Message <> Reply to Message

On Fri, 06 Apr 2007 12:00:29 +0400
Vasily Averin <vvs@sw.ru> wrote:

> When the number of conntracks is reached ip_conntrack_max limit, early_drop() is
> called and tries to free one of already used conntracks in one of the hash
> buckets. If it does not find any conntracks that may be freed, it
> leads to transmission errors.
> However it is not fair because of current hash bucket may be empty but the
> neighbour ones can have the number of conntracks that can be freed. With the
> following patch early_drop() will search conntracks in all hash buckets.

Have you tested your patch in a DOS situation ?
Some machines have a huge ip_conntrack_max.
A single scan of the whole table might take 1000 ms or even more.

Subject: Re: [PATCH 2.6.21-rc6] [netfilter] early_drop imrovement
Posted by vaverin on Fri, 06 Apr 2007 10:26:02 GMT
View Forum Message <> Reply to Message

Eric Dumazet wrote:
> On Fri, 06 Apr 2007 12:00:29 +0400
> Vasily Averin <vvs@sw.ru> wrote:
>
>> When the number of conntracks is reached ip_conntrack_max limit, early_drop() is
>> called and tries to free one of already used conntracks in one of the hash
>> buckets. If it does not find any conntracks that may be freed, it
>> leads to transmission errors.
>> However it is not fair because of current hash bucket may be empty but the
>> neighbour ones can have the number of conntracks that can be freed. With the
>> following patch early_drop() will search conntracks in all hash buckets.
>
> Have you tested your patch in a DOS situation ?
> Some machines have a huge ip_conntrack_max.
> A single scan of the whole table might take 1000 ms or even more.

No, I've not investigated this scenario. It looks like you are right and my

patch can leads to a long delays.

In my experiments I've decreased ip_conntrack_max lower than number of hash
buckets and got the "table full, dropping packet" errors in logs. I've looked on
the conntrack list and found a huge number of conntracks that can be freed.
However my hash bucket was empty and therefore I even did not have any chances
to free something. That's why I would like to check the other hash buckets too.

Ok, let's limit the number of conntracks that can be checked inside
early_drop(). What do you prefer: some round number (for example 100) or
fraction of ip_conntrack_max (for example 1%)?

Thank you,
Vasily Averin

Subject: Re: [PATCH 2.6.21-rc6] [netfilter] early_drop imrovement
Posted by Patrick McHardy on Fri, 06 Apr 2007 15:08:28 GMT
View Forum Message <> Reply to Message

Vasily Averin wrote:
> No, I've not investigated this scenario. It looks like you are right and my
> patch can leads to a long delays.
>
> In my experiments I've decreased ip_conntrack_max lower than number of hash
> buckets and got the "table full, dropping packet" errors in logs. I've looked on
> the conntrack list and found a huge number of conntracks that can be freed.
> However my hash bucket was empty and therefore I even did not have any chances
> to free something. That's why I would like to check the other hash buckets too.
>
> Ok, let's limit the number of conntracks that can be checked inside
> early_drop(). What do you prefer: some round number (for example 100) or
> fraction of ip_conntrack_max (for example 1%)?

A (small) fraction sounds better. We could even consider keeping track
of the number of conntracks that can be evicted (not assured), so in a
DOS situation we could save unnecessary table scans. Not sure if its
worth the effort though.

Anyway, please base your patch on the net-2.6.22 tree, which doesn't
include ip_conntrack anymore.

Subject: [PATCH nf-2.6.22] [netfilter] early_drop imrovement
Posted by vaverin on Sat, 07 Apr 2007 11:45:35 GMT
View Forum Message <> Reply to Message

When the number of conntracks is reached nf_conntrack_max limit, early_drop() is called and tries to free one of already used conntracks in one of the hash buckets. If it does not find any conntracks that may be freed, it leads to transmission errors.
However it is not fair because of current hash bucket may be empty but the neighbour ones can have the number of conntracks that can be freed. On the other hand the number of checked conntracks is not limited and it can cause a long delay.
The following patch limits the number of checked conntracks by average number of conntracks in one hash bucket and allows to search conntracks in other hash buckets.

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
diff --git a/net/netfilter/nf_conntrack_core.c b/net/netfilter/nf_conntrack_core.c
index e132c8a..d0b5794 100644
--- a/net/netfilter/nf_conntrack_core.c
+++ b/net/netfilter/nf_conntrack_core.c
@@ -525,7 +525,7 @@ EXPORT_SYMBOL_GPL(nf_conntrack_tuple_taken);

 /* 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 list_head *chain)
+static int __early_drop(struct list_head *chain, unsigned int *cnt)
 {
  /* Traverse backwards: gives us oldest, which is roughly LRU */
  struct nf_conntrack_tuple_hash *h;
@@ -540,6 +540,10 @@ static int early_drop(struct list_head *chain)
    atomic_inc(&ct->ct_general.use);
    break;
   }
+  if (!--(*cnt)) {
+   dropped = 1;
+   break;
+  }
  }
  read_unlock_bh(&nf_conntrack_lock);

@@ -555,6 +559,21 @@ static int early_drop(struct list_head *chain)
  return dropped;
 }

+static int early_drop(const struct nf_conntrack_tuple *orig)
+{
+ unsigned int i, hash, cnt;
+ int ret = 0;
+
+ hash = hash_conntrack(orig);
+ cnt = (nf_conntrack_max/nf_conntrack_htable_size) + 1;
+
```

```
+ for (i = 0;
+   !ret && i < nf_conntrack_htable_size;
+   ++i, hash = ++hash % nf_conntrack_htable_size)
+  ret = __early_drop(&nf_conntrack_hash[hash], &cnt);
+ return ret;
+}
+
 static struct nf_conn *
 __nf_conntrack_alloc(const struct nf_conntrack_tuple *orig,
        const struct nf_conntrack_tuple *repl,
@@ -574,9 +593,7 @@ __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(orig)) {
    atomic_dec(&nf_conntrack_count);
    if (net_ratelimit())
     printk(KERN_WARNING
```

Subject: Re: [PATCH nf-2.6.22] [netfilter] early_drop imrovement
Posted by Eric Dumazet on Sat, 07 Apr 2007 12:08:47 GMT
View Forum Message <> Reply to Message

Vasily Averin a e'crit :
> When the number of conntracks is reached nf_conntrack_max limit, early_drop() is
> called and tries to free one of already used conntracks in one of the hash
> buckets. If it does not find any conntracks that may be freed, it
> leads to transmission errors.
> However it is not fair because of current hash bucket may be empty but the
> neighbour ones can have the number of conntracks that can be freed. On the other
> hand the number of checked conntracks is not limited and it can cause a long delay.
> The following patch limits the number of checked conntracks by average number of
> conntracks in one hash bucket and allows to search conntracks in other hash buckets.

Hi Vasily

>
>   atomic_inc(&ct->ct_general.use);
>   break;
>   }
> + if (!--(*cnt)) {
> +   dropped = 1;
> +   break;
> + }

> + cnt = (nf_conntrack_max/nf_conntrack_htable_size) + 1;

I am sorry but this wont help in the case you mentioned in an earlier mail :

If nf_conntrack_max < nf_conntrack_htable_size, cnt will be set to 1.

Then in __early_drop() you endup in breaking the list_for_each_entry_reverse()
loop after the first element was tested ! Not what you intended I'm afraid,
because you wont event scan the whole chain as before your patch :(

I believe you should not test --cnt in __early_drop() but in the caller.

(That is not counting the number of found cells, but the number of hash chains
you tried)

Subject: Re: [PATCH nf-2.6.22] [netfilter] early_drop imrovement
Posted by vaverin on Sun, 08 Apr 2007 05:02:40 GMT
View Forum Message <> Reply to Message

Eric Dumazet wrote:
> Vasily Averin a e'crit :
>> When the number of conntracks is reached nf_conntrack_max limit,
>> early_drop() is
>> called and tries to free one of already used conntracks in one of the
>> hash
>> buckets. If it does not find any conntracks that may be freed, it
>> leads to transmission errors.
>> However it is not fair because of current hash bucket may be empty but
>> the
>> neighbour ones can have the number of conntracks that can be freed. On
>> the other
>> hand the number of checked conntracks is not limited and it can cause
>> a long delay.
>> The following patch limits the number of checked conntracks by average
>> number of
>> conntracks in one hash bucket and allows to search conntracks in other
>> hash buckets.
>
> Hi Vasily
>
>>
>> atomic_inc(&ct->ct_general.use);
>> break;
>> }
>> + if (!--(*cnt)) {

>> + dropped = 1;
>> + break;
>> + }
>
>
>> + cnt = (nf_conntrack_max/nf_conntrack_htable_size) + 1;
>
> I am sorry but this wont help in the case you mentioned in an earlier
> mail :
>
> If nf_conntrack_max < nf_conntrack_htable_size, cnt will be set to 1.
>
> Then in __early_drop() you endup in breaking the
> list_for_each_entry_reverse() loop after the first element was tested !
> Not what you intended I'm afraid, because you wont event scan the whole
> chain as before your patch :(

I would note that in my experiment I got errors when first checked hash bucket
was empty. With this patch I have guarantee that at least one conntrack will be
checked. I'm agree 1 is not too high, but it is better than nothing. I've
checked, my testcase works now.

> I believe you should not test --cnt in __early_drop() but in the caller.
>
> (That is not counting the number of found cells, but the number of hash
> chains you tried)

I need to count conntracks but not hash buckets. Also it is possible that all
the conntracks will be placed to only one hash bucket, and as you pointed in
your previous letter it may lead to long delays.

However how do you think, is it probably better to set low limit to default
average number of conntracks in hash bucket?

cnt = max(8U, nf_conntrack_max/nf_conntrack_htable_size);

Thank you,
 Vasily Averin

Subject: [NETFILTER] early_drop() imrovement (v3)
Posted by vaverin on Wed, 09 May 2007 06:59:03 GMT
View Forum Message <> Reply to Message

When the number of conntracks is reached nf_conntrack_max limit, early_drop()
tries to free one of already used conntracks. If it does not find any conntracks
that may be freed, it leads to transmission errors.
In current implementation the conntracks are searched in one hash bucket only.

It have some drawbacks: if used hash bucket is empty we have not any chances to
find something. On the other hand the hash bucket can contain a huge number of
conntracks and its check can last a long time.
The proposed patch limits the number of checked conntracks by default number of
conntracks in one hash bucket (NF_CT_PER_BUCKET) and allows to search conntracks
in other hash buckets. As result in any case the search will have the same
chances to free one of the conntracks and the check will not lead to long delays.

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
diff --git a/net/netfilter/nf_conntrack_core.c b/net/netfilter/nf_conntrack_core.c
index e132c8a..d984bce 100644
--- a/net/netfilter/nf_conntrack_core.c
+++ b/net/netfilter/nf_conntrack_core.c
@@ -76,6 +76,8 @@ static unsigned int nf_conntrack_next_id;
 DEFINE_PER_CPU(struct ip_conntrack_stat, nf_conntrack_stat);
 EXPORT_PER_CPU_SYMBOL(nf_conntrack_stat);

+#define NF_CT_PER_BUCKET 8U
+
 /*
  * This scheme offers various size of "struct nf_conn" dependent on
  * features(helper, nat, ...)
@@ -525,7 +527,7 @@ EXPORT_SYMBOL_GPL(nf_conntrack_tuple_taken);

 /* 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 list_head *chain)
+static int __early_drop(struct list_head *chain, unsigned int *cnt)
 {
  /* Traverse backwards: gives us oldest, which is roughly LRU */
  struct nf_conntrack_tuple_hash *h;
@@ -540,6 +542,8 @@ static int early_drop(struct list_head *chain)
    atomic_inc(&ct->ct_general.use);
    break;
   }
+  if (!--(*cnt))
+   break;
  }
  read_unlock_bh(&nf_conntrack_lock);

@@ -555,6 +559,21 @@ static int early_drop(struct list_head *chain)
  return dropped;
 }

+static int early_drop(const struct nf_conntrack_tuple *orig)
+{
+ unsigned int i, hash, cnt;
```

```
+ int ret = 0;
+
+ hash = hash_conntrack(orig);
+ cnt = NF_CT_PER_BUCKET;
+
+ for (i = 0;
+  !ret && cnt && i < nf_conntrack_htable_size;
+   ++i, hash = ++hash % nf_conntrack_htable_size)
+  ret = __early_drop(&nf_conntrack_hash[hash], &cnt);
+ return ret;
+}
+
 static struct nf_conn *
 __nf_conntrack_alloc(const struct nf_conntrack_tuple *orig,
        const struct nf_conntrack_tuple *repl,
@@ -574,9 +593,7 @@ __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(orig)) {
    atomic_dec(&nf_conntrack_count);
    if (net_ratelimit())
     printk(KERN_WARNING
@@ -1226,7 +1243,7 @@ int __init nf_conntrack_init(void)
   if (nf_conntrack_htable_size < 16)
    nf_conntrack_htable_size = 16;
  }
- nf_conntrack_max = 8 * nf_conntrack_htable_size;
+ nf_conntrack_max = NF_CT_PER_BUCKET * nf_conntrack_htable_size;

  printk("nf_conntrack version %s (%u buckets, %d max)\n",
         NF_CONNTRACK_VERSION, nf_conntrack_htable_size,
```

Subject: Re: [NETFILTER] early_drop() imrovement (v3)
Posted by Patrick McHardy on Mon, 25 Jun 2007 13:53:22 GMT
View Forum Message <> Reply to Message

Vasily Averin wrote:
> +static int early_drop(const struct nf_conntrack_tuple *orig)
> +{
> + unsigned int i, hash, cnt;
> + int ret = 0;
> +
> + hash = hash_conntrack(orig);

> + cnt = NF_CT_PER_BUCKET;
> +
> + for (i = 0;
> + !ret && cnt && i < nf_conntrack_htable_size;
> +   ++i, hash = ++hash % nf_conntrack_htable_size)
> + ret = __early_drop(&nf_conntrack_hash[hash], &cnt);

Formatting is a bit ugly, looks much nicer as:

```
        for (i = 0; i < nf_conntrack_htable_size; i++) {

                ret = __early_drop(&nf_conntrack_hash[hash], &cnt);
                if (ret || !cnt)
                        break;
                hash = ++hash % nf_conntrack_htable_size;
        }
```

> @@ -1226,7 +1243,7 @@ int __init nf_conntrack_init(void)
>   if (nf_conntrack_htable_size < 16)
>   nf_conntrack_htable_size = 16;
>   }
> - nf_conntrack_max = 8 * nf_conntrack_htable_size;
> + nf_conntrack_max = NF_CT_PER_BUCKET * nf_conntrack_htable_size;

I don't like the NF_CT_PER_BUCKET constant. First of all, each
conntrack is hashed twice, so its really only 1/2 of the average
conntracks per bucket. Secondly, its only a default and many
people use nf_conntrack_max = nf_conntrack_htable_size / 2, so
using this constant for early_drop seems wrong.

Perhaps make it 2 * nf_conntrack_max / nf_conntrack_htable_size
or even add a nf_conntrack_eviction_range sysctl.

Subject: Re: [NETFILTER] early_drop() imrovement (v3)
Posted by Jan Engelhardt on Mon, 25 Jun 2007 14:36:50 GMT
View Forum Message <> Reply to Message

On Jun 25 2007 15:53, Patrick McHardy wrote:
>Vasily Averin wrote:
>> +static int early_drop(const struct nf_conntrack_tuple *orig)
>> +{
>> + unsigned int i, hash, cnt;
>> + int ret = 0;
>> +
>> + hash = hash_conntrack(orig);
>> + cnt = NF_CT_PER_BUCKET;

>> +
>> + for (i = 0;
>> + !ret && cnt && i < nf_conntrack_htable_size;
>> +   ++i, hash = ++hash % nf_conntrack_htable_size)
>> + ret = __early_drop(&nf_conntrack_hash[hash], &cnt);
>
>Formatting is a bit ugly, looks much nicer as:
>
>	for (i = 0; i < nf_conntrack_htable_size; i++) {
>
>		ret = __early_drop(&nf_conntrack_hash[hash], &cnt);
>		if (ret || !cnt)
>			break;
>		hash = ++hash % nf_conntrack_htable_size;
>	}

gcc warning: operation on ‘hash’ may be undefined

Jan
--

Subject: Re: [NETFILTER] early_drop() imrovement (v3)
Posted by vaverin on Tue, 26 Jun 2007 13:20:18 GMT
View Forum Message <> Reply to Message

Patrick McHardy wrote:

Patrick, thank you for your tips, I'll remake the patch.

> I don't like the NF_CT_PER_BUCKET constant. First of all, each
> conntrack is hashed twice, so its really only 1/2 of the average
> conntracks per bucket. Secondly, its only a default and many
> people use nf_conntrack_max = nf_conntrack_htable_size / 2, so
> using this constant for early_drop seems wrong.

> Perhaps make it 2 * nf_conntrack_max / nf_conntrack_htable_size
> or even add a nf_conntrack_eviction_range sysctl.

IMHO The number of conntracks checked in early_drop() have following restrictions:
- it should be not too low -- to decrease chances of transmission failures,
- it should be limited by some reasonable value -- to prevent long check delays.

Also I believe it makes sense to have it constant (how about NF_CT_EVICTION
name?) -- to have the same behaviour on various nodes. However I doubt strongly
that anybody will want to change this value. Do you think it is really required?

thank you,
Vasily Averin

Subject: Re: [NETFILTER] early_drop() imrovement (v3)
Posted by Patrick McHardy on Tue, 26 Jun 2007 13:27:30 GMT
View Forum Message <> Reply to Message

Vasily Averin wrote:
> Patrick McHardy wrote:
>> I don't like the NF_CT_PER_BUCKET constant. First of all, each
>> conntrack is hashed twice, so its really only 1/2 of the average
>> conntracks per bucket. Secondly, its only a default and many
>> people use nf_conntrack_max = nf_conntrack_htable_size / 2, so
>> using this constant for early_drop seems wrong.
>>
>> Perhaps make it 2 * nf_conntrack_max / nf_conntrack_htable_size
>> or even add a nf_conntrack_eviction_range sysctl.
>>
>
> IMHO The number of conntracks checked in early_drop() have following restrictions:
> - it should be not too low -- to decrease chances of transmission failures,
> - it should be limited by some reasonable value -- to prevent long check delays.

Agreed.

> Also I believe it makes sense to have it constant (how about NF_CT_EVICTION
> name?) -- to have the same behaviour on various nodes. However I doubt strongly
> that anybody will want to change this value. Do you think it is really required?
>

I don't know. The current behaviour will on average scan 16 entries.
For people manually tuning their hash to saner settings it will scan
a single entry. So we have a quite wide range of values already.
The single entry with sane hash settings is too little IMO, maybe use
some middle-ground, make it 8 by default as you did and rename the
constant. NF_CT_EVICTION_RANGE sounds fine.

Subject: [NETFILTER] early_drop() imrovement (v4)
Posted by vaverin on Wed, 27 Jun 2007 08:46:08 GMT
View Forum Message <> Reply to Message

When the number of conntracks is reached nf_conntrack_max limit, early_drop()
tries to free one of already used conntracks. If it does not find any conntracks
that may be freed, it leads to transmission errors.

In current implementation the conntracks are searched in one hash bucket only.
It have some drawbacks: if used hash bucket is empty we have not any chances to find something. On the other hand the hash bucket can contain a huge number of conntracks and its check can last a long time.
The proposed patch limits the number of checked conntracks and allows to search conntracks in other hash buckets. As result in any case the search will have the same chances to free one of the conntracks and the check will not lead to long delays.

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
diff --git a/net/netfilter/nf_conntrack_core.c b/net/netfilter/nf_conntrack_core.c
index 7a15e30..0540a88 100644
--- a/net/netfilter/nf_conntrack_core.c
+++ b/net/netfilter/nf_conntrack_core.c
@@ -526,7 +526,7 @@ EXPORT_SYMBOL_GPL(nf_conntrack_tuple_taken);

 /* 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 list_head *chain)
+static int __early_drop(struct list_head *chain, unsigned int *cnt)
 {
  /* Traverse backwards: gives us oldest, which is roughly LRU */
  struct nf_conntrack_tuple_hash *h;
@@ -541,6 +541,8 @@ static int early_drop(struct list_head *chain)
    atomic_inc(&ct->ct_general.use);
    break;
   }
+  if (!--(*cnt))
+   break;
  }
  read_unlock_bh(&nf_conntrack_lock);

@@ -556,6 +558,25 @@ static int early_drop(struct list_head *chain)
  return dropped;
 }

+#define NF_CT_EVICTION_RANGE 8U
+
+static int early_drop(const struct nf_conntrack_tuple *orig)
+{
+ unsigned int i, hash, cnt;
+ int ret = 0;
+
+ hash = hash_conntrack(orig);
+ cnt = NF_CT_EVICTION_RANGE;
+
+ for (i = 0; i < nf_conntrack_htable_size; i++) {
```

```
+  ret = __early_drop(&nf_conntrack_hash[hash], &cnt);
+  if (ret || !cnt)
+   break;
+  hash++; hash %= nf_conntrack_htable_size;
+ }
+ return ret;
+}
+
 static struct nf_conn *
 __nf_conntrack_alloc(const struct nf_conntrack_tuple *orig,
        const struct nf_conntrack_tuple *repl,
@@ -575,9 +596,7 @@ __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(orig)) {
    atomic_dec(&nf_conntrack_count);
    if (net_ratelimit())
     printk(KERN_WARNING
```

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by Patrick McHardy on Wed, 27 Jun 2007 08:52:16 GMT
View Forum Message <> Reply to Message

Vasily Averin wrote:
> When the number of conntracks is reached nf_conntrack_max limit, early_drop()
> tries to free one of already used conntracks. If it does not find any conntracks
> that may be freed, it leads to transmission errors.
> In current implementation the conntracks are searched in one hash bucket only.
> It have some drawbacks: if used hash bucket is empty we have not any chances to
> find something. On the other hand the hash bucket can contain a huge number of
> conntracks and its check can last a long time.
> The proposed patch limits the number of checked conntracks and allows to search
> conntracks in other hash buckets. As result in any case the search will have the
> same chances to free one of the conntracks and the check will not lead to long
> delays.

Thanks Vasily. I have some patches queued to convert all conntrack
hashes to hlists, which conflict with your patches. They need a bit
more work, I'll integrate your changes on top of them once I'm done.

BTW, I played around with your last patch yesterday and it shows
a big improvement when flooding the machine with new connections.

Previously about 5% of the (valid) new connections would get
dropped, with your patch not a single one :)

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by Patrick McHardy on Wed, 27 Jun 2007 12:04:15 GMT
View Forum Message <> Reply to Message

Patrick McHardy wrote:
> Vasily Averin wrote:
>
>>When the number of conntracks is reached nf_conntrack_max limit, early_drop()
>>tries to free one of already used conntracks. If it does not find any conntracks
>>that may be freed, it leads to transmission errors.
>>In current implementation the conntracks are searched in one hash bucket only.
>>It have some drawbacks: if used hash bucket is empty we have not any chances to
>>find something. On the other hand the hash bucket can contain a huge number of
>>conntracks and its check can last a long time.
>>The proposed patch limits the number of checked conntracks and allows to search
>>conntracks in other hash buckets. As result in any case the search will have the
>>same chances to free one of the conntracks and the check will not lead to long
>>delays.
>
>
>
> Thanks Vasily. I have some patches queued to convert all conntrack
> hashes to hlists, which conflict with your patches. They need a bit
> more work, I'll integrate your changes on top of them once I'm done.

I've added this patch to my tree at

http://people.netfilter.org/kaber/nf-2.6.23.git/

I've joined the two loops from your patch since that avoids an
otherwise useless function and doesn't take the lock up to 8
times in a row.

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by vaverin on Wed, 27 Jun 2007 12:29:33 GMT
View Forum Message <> Reply to Message

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.

Thank you,
 Vasily Averin

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
```

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by vaverin on Wed, 27 Jun 2007 13:02:24 GMT
View Forum Message <> Reply to Message

Patrick McHardy wrote:
> Vasily Averin wrote:
>> 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.
> +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;
>   }
```

it is incorrect again: when cnt=0 you should break both cycles.

thank you,
Vasily Averin

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by Patrick McHardy on Wed, 27 Jun 2007 13:18:07 GMT
View Forum Message <> Reply to Message

Vasily Averin wrote:
> Patrick McHardy wrote:
>
> it is incorrect again: when cnt=0 you should break both cycles.

Indeed, thanks. Fixed now. Also changed it to leave the loop
if we found an entry within a chain (we want the last one of
the chain, so we still walk it entirely) and replaced

hash = (hash + 1) % nf_conntrack_htable_size

by

hash = (hash + 1) & (nf_conntrack_htable_size - 1)

since one of my queued patches makes sure that its always
a power of two.

[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 047d8f088a71f30d3042cc7615a7a25aa60a668b
tree c83c4a356caf608fe21a646c24678bef74023270
parent 665d98d03473cab252830129f414e1b38fb2b038
author Patrick McHardy <kaber@trash.net> Wed, 27 Jun 2007 15:17:17 +0200
committer Patrick McHardy <kaber@trash.net> Wed, 27 Jun 2007 15:17:17 +0200

 net/netfilter/nf_conntrack_core.c | 26 ++++++++++++++++++++--------
 1 files changed, 18 insertions(+), 8 deletions(-)

diff --git a/net/netfilter/nf_conntrack_core.c b/net/netfilter/nf_conntrack_core.c
index d7e62ad..2d5c10f 100644
--- a/net/netfilter/nf_conntrack_core.c
+++ b/net/netfilter/nf_conntrack_core.c
@@ -377,22 +377,33 @@ 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)
+    goto stop;
+  }
+  if (ct)
+   break;
+  hash = (hash + 1) & (nf_conntrack_htable_size - 1);
  }
+stop:
  if (ct)
   atomic_inc(&ct->ct_general.use);
  read_unlock_bh(&nf_conntrack_lock);
@@ -425,8 +436,7 @@ 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
```

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by Patrick McHardy on Wed, 27 Jun 2007 13:23:00 GMT
View Forum Message <> Reply to Message

[Dropping a few CCs since I supect its beginning to be annoying :)]

Patrick McHardy wrote:
> Vasily Averin wrote:
>
> Indeed, thanks. Fixed now. Also changed it to leave the loop
> if we found an entry within a chain (we want the last one of
> the chain, so we still walk it entirely) and replaced
>
> hash = (hash + 1) % nf_conntrack_htable_size
>
> by
>
> hash = (hash + 1) & (nf_conntrack_htable_size - 1)
>
> since one of my queued patches makes sure that its always
> a power of two.

Damn it .. it doesn't, it just makes sure it always uses pages
entirely. So I fixed that again.

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by vaverin on Wed, 27 Jun 2007 13:25:58 GMT
View Forum Message <> Reply to Message

Patrick McHardy wrote:
> Vasily Averin wrote:
>> Patrick McHardy wrote:
> -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;

It is incorrect: you should break nested loop here too.

> +   if (--cnt <= 0)
> +   goto stop;
> + }
> + if (ct)
> +   break;
> + hash = (hash + 1) & (nf_conntrack_htable_size - 1);
>   }
> +stop:
>   if (ct)
>   atomic_inc(&ct->ct_general.use);
>   read_unlock_bh(&nf_conntrack_lock);

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by Patrick McHardy on Wed, 27 Jun 2007 13:28:39 GMT
View Forum Message <> Reply to Message

Vasily Averin wrote:
> Patrick McHardy wrote:
>
>>+ 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;
>
>
> It is incorrect: you should break nested loop here too.

No, as I said, we want the last entry of the chain.

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by Patrick McHardy on Wed, 27 Jun 2007 13:35:49 GMT
View Forum Message <> Reply to Message

Patrick McHardy wrote:
> Vasily Averin wrote:
>
>>Patrick McHardy wrote:
>>
>>
>>>+ 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;
>>
>>
>>It is incorrect: you should break nested loop here too.
>
>
>
> No, as I said, we want the last entry of the chain.

Ideally we should do something like this I think (please let it be
correct :)):

+   for (i = 0; i < nf_conntrack_htable_size; i++) {

```
+               entries = 0;
+               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;
+                       entries++;
+               }
+               if (ct)
+                       break;
+               if ((cnt -= entries) <= 0)
+                       break;
+               hash = (hash + 1) % nf_conntrack_htable_size;
        }
```

So we always walk chains up to the end and NF_CT_EVICTION_RANGE is
just a minimum. This ensures we will always get the last entry *and*
we won't scan less entries than currently if someone has a chain
longer than 8 entries.

What do you think?

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by Patrick McHardy on Wed, 27 Jun 2007 13:54:52 GMT
View Forum Message <> Reply to Message

Patrick McHardy wrote:
> Patrick McHardy wrote:
>
> Ideally we should do something like this I think (please let it be
> correct :)):
>
> [...]
> So we always walk chains up to the end and NF_CT_EVICTION_RANGE is
> just a minimum. This ensures we will always get the last entry *and*
> we won't scan less entries than currently if someone has a chain
> longer than 8 entries.
>
> What do you think?

I've added this patch now.

[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 makes it walk the hash until a minimum number of
8 entries are checked.

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

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

---
commit 31889ee2d8f42f84daec97c3b98c47165e358da8
tree 4a1fe840d3056c2e64ab027cc10f92f3843cd710
parent ca0ac66daa3b264702d72282e388f8ba920f9a91
author Patrick McHardy <kaber@trash.net> Wed, 27 Jun 2007 15:54:22 +0200
committer Patrick McHardy <kaber@trash.net> Wed, 27 Jun 2007 15:54:22 +0200

 net/netfilter/nf_conntrack_core.c | 24 ++++++++++++++++++--------
 1 files changed, 16 insertions(+), 8 deletions(-)

diff --git a/net/netfilter/nf_conntrack_core.c b/net/netfilter/nf_conntrack_core.c
index ed44a09..ef3f747 100644
--- a/net/netfilter/nf_conntrack_core.c
+++ b/net/netfilter/nf_conntrack_core.c
@@ -377,21 +377,30 @@ 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 = 0;

  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;
+   cnt++;
+  }
+  if (ct || cnt >= NF_CT_EVICTION_RANGE)
+   break;
+  hash = (hash + 1) % nf_conntrack_htable_size;
  }
  if (ct)
   atomic_inc(&ct->ct_general.use);
@@ -425,8 +434,7 @@ 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
```

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by Rusty Russell on Mon, 02 Jul 2007 19:56:39 GMT
View Forum Message <> Reply to Message

On Wed, 2007-06-27 at 15:54 +0200, Patrick McHardy wrote:
> [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 makes it walk the hash until a minimum number of
> 8 entries are checked.
>
> Based on patch by Vasily Averin <vvs@sw.ru>.
>
> Signed-off-by: Patrick McHardy <kaber@trash.net>
>
> ---
> commit 31889ee2d8f42f84daec97c3b98c47165e358da8
> tree 4a1fe840d3056c2e64ab027cc10f92f3843cd710

> parent ca0ac66daa3b264702d72282e388f8ba920f9a91
> author Patrick McHardy <kaber@trash.net> Wed, 27 Jun 2007 15:54:22 +0200
> committer Patrick McHardy <kaber@trash.net> Wed, 27 Jun 2007 15:54:22 +0200
>
> net/netfilter/nf_conntrack_core.c | 24 ++++++++++++++++++--------
> 1 files changed, 16 insertions(+), 8 deletions(-)
>
> diff --git a/net/netfilter/nf_conntrack_core.c b/net/netfilter/nf_conntrack_core.c
> index ed44a09..ef3f747 100644
> --- a/net/netfilter/nf_conntrack_core.c
> +++ b/net/netfilter/nf_conntrack_core.c
> @@ -377,21 +377,30 @@ 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 = 0;
>
>   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;
> +   cnt++;
> + }
> + if (ct || cnt >= NF_CT_EVICTION_RANGE)
> +   break;
> + hash = (hash + 1) % nf_conntrack_htable_size;
>   }
>   if (ct)
>   atomic_inc(&ct->ct_general.use);

This looks good. The randomness in the hash means we no longer need the
"hit the same hash bucket" heuristic to avoid hashbombing.

I still wonder if we should batch up the drops a little while we're
doing all this work? Should reduce stress under serious flood load.

Thanks,
Rusty.

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by Martin Josefsson on Tue, 03 Jul 2007 06:39:53 GMT
View Forum Message <> Reply to Message

On Tue, 3 Jul 2007, Rusty Russell wrote:

> This looks good. The randomness in the hash means we no longer need the
> "hit the same hash bucket" heuristic to avoid hashbombing.
>
> I still wonder if we should batch up the drops a little while we're
> doing all this work? Should reduce stress under serious flood load.

Yes we should really do that, going searching for something to evict
for each new connection attempt is really painful and in this
overload situation we need all the cpu we can get.

/Martin

Subject: Re: [NETFILTER] early_drop() imrovement (v4)
Posted by Patrick McHardy on Tue, 03 Jul 2007 11:42:54 GMT
View Forum Message <> Reply to Message

Rusty Russell wrote:
> On Wed, 2007-06-27 at 15:54 +0200, Patrick McHardy wrote:
>
>>[NETFILTER]: nf_conntrack: early_drop improvement
>
> This looks good. The randomness in the hash means we no longer need the
> "hit the same hash bucket" heuristic to avoid hashbombing.
>
> I still wonder if we should batch up the drops a little while we're
> doing all this work? Should reduce stress under serious flood load.

Good point, I didn't think of that. Its a bit tricky though, we can't

destroy them while holding nf_conntrack_lock, so we'd either have to release and re-grab it for every conntrack we want to kill or write lock it from the beginning, clean them from the lists immediately and put them on a temporary destruction queue. Or split death_by_timeout so it can deal with callers already holding nf_conntrack_lock ..

I'll see if I can come up with a halfway decent looking patch :)
