strongSwan - Bug #2399

libipsec - soft timeout of one CHILD_SA results in hard timeout of another

09.08.2017 15:28 - Phil Levin

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Tobias Brunner	**Estimated time:**	0.00 hour
Category:	libipsec		
Target version:	5.6.1		
Affected version:	5.1.1	**Resolution:**	Fixed

Description

summary

I'm seeing an issue with libipsec where the rekeying of one CHILD_SA will result in the hard timeout of a different CHILD_SA.

The problem appears to be the that ipsec_sa_mgr.c uses a cached pointer (ipsec_sa_entry_t *) in the job callback context (ipsec_sa_expired_t) to find the CHILD_SA during both its soft and hard timeout. When the soft timeout triggers, the CHILD_SA is deleted (after rekeying is completed) which frees the ipsec_sa_expired_t pointer.

In the meanwhile,a new CHILD_SA may happen to reuse the previously freed ipsec_sa_expired_t pointer. When the original (now deleted) CHILD_SA job callback finally reaches it's hard timeout, it still has the original cached (ipsec_sa_entry_t *) value, which is now assigned to the new CHILD_SA. This results in the **new** CHILD_SA being hard-expired.

Question: Could this be solved by caching spi, src, and dst in ipsec_sa_expired_t instead of (ipsec_sa_entry_t *). Then use match_entry_by_spi_src_dst for the callback lookup?

details

ipsec_sa_mgr.c uses pointer comparison to find the appropriate ipsec_sa_entry_t struct to apply both a soft and subsequent hard timeout to, in sa_expired()

```
>   if (this->sas->find_first(this->sas, (void*)match_entry_by_ptr,
>                             NULL, expired->entry) == SUCCESS)
```

schedule_expiration() appears to assume that the callback context (ipsec_sa_expired_t *expired) will be valid for soft timeout:

```
>   job = callback_job_create((callback_job_cb_t)sa_expired, expired,
>                             (callback_job_cleanup_t)free, NULL);
>   lib->scheduler->schedule_job(lib->scheduler, (job_t*)job, timeout);
```

...and subsequent hard timeout callback in sa_expired():

```
>       if (hard_offset)
>       {   /* soft limit reached, schedule hard expire */
>           expired->hard_offset = 0;
>           this->mutex->unlock(this->mutex);
>           return JOB_RESCHEDULE(hard_offset);
>       }
```

reproduction

I have a test setup that creates 20 host-to-host IPSec connections each with a

single child_SA. IKE_SA rekeying is disabled and only the connection initiators have child rekeying enabled with the following:

```
  connections.<conn>.children.<child>.rekey_time 20
  connections.<conn>.children.<child>.life_time  40
  connections.<conn>.children.<child>.rand_time   1
```

I added debug code to ipsec_sa_mgr.c (see attached diffs). Here's a key to the debug output below:

```
  entry         ipsec_sa_entry_t *entry
  expired       ipsec_sa_expired_t *expired
  life          lifetime->time.life
  rekey         lifetime->time.rekey
  timeout       lifetime->time.life or lifetime->time.rekey
  hard_offset   expired->hard_offset
```

debug output

A CHILD_SA is created, scheduled and inserted into the ipsec_sa_mgr database in schedule_expiration() and add_sa(). Note that the job callback context (expired=0x7fed08000cc0) has cached the child_sa entry of 0x7fed080130b0:

```
> Aug  8 18:53:16 1502218396.615 10[ESP] >>>SCHEDULE: entry=0x7fed080130b0 spi=0xd8c1ecaa sa=0x7fe
d08011570 expired=0x7fed08000cc0 life=40 rekey=18 timeout=18 hard_offset=22
> Aug  8 18:53:16 1502218396.615 10[ESP] >>>INSERTED: entry=0x7fed080130b0 spi=0xd8c1ecaa sa=0x7fe
d08011570
```

Then the job callback for rekey (soft) timeout fires, leaving 22 seconds remaining for the hard timeout.

```
> Aug  8 18:53:34 1502218414.615 12[ESP] >>>EXPIRED1: entry=0x7fed080130b0 expired=0x7fed08000cc0
hard_offset=22
> Aug  8 18:53:34 1502218414.615 12[ESP] >>>EXPIRED2: entry=0x7fed080130b0 spi=0xd8c1ecaa sa=0x7fe
d08011570 expired=0x7fed08000cc0 hard_offset=22
```

The above rekey eventually causes the rekeyed CHILD_SA to be deleted and removed in del_sa():

```
> Aug  8 18:53:34 1502218414.694 15[ESP] >>>REMOVED2: entry=0x7fed080130b0 spi=0xd8c1ecaa sa=0x7fe
d08011570
```

The above frees the old pointer (entry=0x7fed080130b0). A new CHILD_SA (spi=0xacc9ede3) is created and inserted, but the memory alloctor hands it the recently freed entry pointer (entry=0x7fed080130b0):

```
> Aug  8 18:53:52 1502218432.725 10[ESP] >>>SCHEDULE: entry=0x7fed080130b0 spi=0xacc9ede3 sa=0x7fe
d080196c0 expired=0x7fed080090b0 life=40 rekey=18 timeout=18 hard_offset=22
> Aug  8 18:53:52 1502218432.725 10[ESP] >>>INSERTED: entry=0x7fed080130b0 spi=0xacc9ede3 sa=0x7fe
d080196c0
```

In the meanwhile, the hard timeout from expired=0x7fed08000cc0 finally fires. But it's callback context still has the original entry pointer (entry=0x7fed080130b0) which was reassigned assigned to the **new** CHILD_SA (spi=0xacc9ede3). The hard_offset is from the original entry pointer and is now "0" which causes the **new** CHILD_SA to see a hard timeout:

```
> Aug  8 18:53:56 1502218436.615 08[ESP] >>>EXPIRED1: entry=0x7fed080130b0 expired=0x7fed08000cc0
hard_offset=0
> Aug  8 18:53:56 1502218436.615 08[ESP] >>>EXPIRED2: entry=0x7fed080130b0 spi=0xacc9ede3 sa=0x7fe
d080196c0 expired=0x7fed08000cc0 hard_offset=0
```

This deletes the **new** (spi=0xacc9ede3) CHILD_SA:

```
> Aug  8 18:53:56 1502218436.615 08[ESP] >>>REMOVED1: entry=0x7fed080130b0 spi=0xacc9ede3 sa=0x7fe
d080196c0 expired=0x7fed08000cc0 hard_offset0
```

Associated revisions

Revision 6e861947 - 18.09.2017 10:51 - Tobias Brunner

libipsec: Make sure to expire the right SA

If an IPsec SA is actually replaced with a rekeying its entry in the manager is freed. That means that when the hard expire is triggered a new entry might be found at the cached pointer location. So we have to make sure we trigger the expire only if we found the right SA.

We could use SPI and addresses for the lookup, but this here requires a bit less memory and is just a small change. Another option would be to somehow cancel the queued job, but our scheduler doesn't allow that at the moment.

Fixes #2399.

History

#1 - 10.08.2017 14:35 - Phil Levin

P.S. Thanks in advance!

#2 - 14.08.2017 16:14 - Tobias Brunner

- Tracker changed from Issue to Bug

- Category set to libipsec

- Status changed from New to Feedback

- Assignee set to Tobias Brunner

- Target version set to 5.6.1

> The problem appears to be the that ipsec_sa_mgr.c uses a cached pointer (ipsec_sa_entry_t *) in the job callback context (ipsec_sa_expired_t) to find the CHILD_SA during both its soft and hard timeout. When the soft timeout triggers, the CHILD_SA is deleted (after rekeying is completed) which frees the ipsec_sa_expired_t pointer.
>
> In the meanwhile,a new CHILD_SA may happen to reuse the previously freed ipsec_sa_expired_t pointer. When the original (now deleted) CHILD_SA job callback finally reaches it's hard timeout, it still has the original cached (ipsec_sa_entry_t *) value, which is now assigned to the new CHILD_SA. This results in the **new** CHILD_SA being hard-expired.

Thanks for the detailed report. Hm, yeah, that code doesn't really make sense. It would require either canceling the event when the SA is deleted (which our scheduler currently doesn't support), or using a different/additional identifier.

> **Question: Could this be solved by caching spi, src, and dst in ipsec_sa_expired_t instead of (ipsec_sa_entry_t *). Then use match_entry_by_spi_src_dst for the callback lookup?**

I guess so. Another option is to store the SPI of the SA on the ipsec_sa_expired_t struct and compare that after retrieving the entry via pointer (requires a bit less memory than two cloned addresses or even one). Patch is in the *2399-libipsec-expire-right-sa* branch.

#3 - 18.09.2017 11:06 - Tobias Brunner

- Status changed from Feedback to Closed

- Resolution set to Fixed

Files

debug.patch	2.58 KB	09.08.2017	Phil Levin