strongSwan - Bug #1014

Possible data race when destroying libcharon (multiple issues in ike-sa-manager, trap-manager and shunt-manager)

30.06.2015 13:56 - Avinoam Meir

Status:	Closed	**Start date:**	30.06.2015
Priority:	Normal	**Due date:**	
Assignee:	Tobias Brunner	**Estimated time:**	0.00 hour
Category:	libcharon		
Target version:	5.3.3		
Affected version:	5.3.2	**Resolution:**	Fixed

Description

I am running a data race detector (https://code.google.com/p/thread-sanitizer/) on StrongSwan and it found a race condition when destroying libcharon.

Thread 1 call stack:

0. get_spi libcharon/sa/ike_sa_manager.c:
1. checkout_new libcharon/sa/ike_sa_manager.c
2. checkout_by_config libcharon/sa/ike_sa_manager.c
3. acquire libcharon/sa/trap_manager.c
4. execute libcharon/processing/jobs/acquire_job.c

The context switch occurs in get_spi in the line:
if (this->rng && this->rng->get_bytes(this->rng, sizeof(spi), (u_int8_t*)&spi))
between the 2 conditions

Thread 2 call stack:

0. destroy libstrongswan/plugins/random/random_rng.c
1. flush libcharon/sa/ike_sa_manager.c:
2. destroy libcharon/daemon.c
3. libcharon_deinit libcharon/daemon.c

I can see 2 options how to avoid it:

1) Move rng->destroy in private_ike_sa_manager_t from flush to destroy
2) In daemon->destroy calling first trap_manger->flush and just then ike_sa_manager->flush

patch attached.

Thanks,
Avinoam

Associated revisions

Revision 390ae7a2 - 14.07.2015 15:25 - Tobias Brunner

ike-sa-manager: Safely access the RNG instance with an rwlock

Threads might still be allocating SPIs (e.g. triggered by an acquire or an inbound message) while the main thread calls flush(). If there is a context switch right after such a thread successfully checked this->rng in get_spi() and the main thread destroys the RNG instance right then, that worker thread will cause a segmentation fault when it continues and attempts to call get_bytes().

Fixes #1014.

Revision 5af74bcb - 27.07.2015 14:00 - Tobias Brunner

Merge branch 'trap-shunt-updates'

Changes how acquires are tracked in the trap manager, which fixes several race conditions. Also fixes races between threads installing trap policies and the main thread trying to flush the trap policies. Similar changes were added to the shunt manager which previously used no locking at all.

Fixes #1014.

History

#1 - 08.07.2015 15:48 - Tobias Brunner

- Tracker changed from Issue to Bug

- Status changed from New to Feedback

- Assignee set to Tobias Brunner

- Target version set to 5.3.3

Thanks for the report.

> 1) Move rng->destroy in private_ike_sa_manager_t from flush to destroy

Unfortunately, this won't work as the plugins (one of which provides the RNG implementation) will already be unloaded when destroy() is called. So this change could result in a segmentation fault.

But accessing this member without a lock is definitely a bad idea. I've pushed a patch to the *ike-sa-manager-rng* branch that wraps it with an rw-lock to allow multiple threads to allocate SPIs concurrently.

> 2) In daemon->destroy calling first trap_manger->flush and just then ike_sa_manager->flush

While this change might help in this particular case, there could be other threads/jobs causing the same issue (e.g. a process_message_job_t that calls checkout_by_message()). And even an acquire job could already be a in checkout_by_config() while the main thread flushes the trap manager and then the IKE_SA manager. So I'd rather just fix the RNG issue.

#2 - 09.07.2015 10:16 - Avinoam Meir

I tested the branch and it resolves the data race.

However there is another data race.

```
Thread 1 call stack:
    #0 complete libcharon/sa/trap_manager.c:395:14
    #1 ike_state_change libcharon/sa/trap_manager.c
    #2 ike_state_change libcharon/bus/bus.c
    #3 set_state third_libcharon/sa/ike_sa.c
    #4 destroy libcharon/sa/ike_sa.c
    #5 entry_destroy libcharon/sa/ike_sa_manager.c
    #6 flush libcharon/sa/ike_sa_manager.c
    #7 destroy libcharon/daemon.c
    #8 libcharon_deinit libcharon/daemon.c

Thread 2 callstack:
    #0 acquire libcharon/sa/trap_manager.c:368:19
    #1 execute libcharon/processing/jobs/acquire_job.c
    #2 process_job libstrongswan/processing/processor.c
    #3 process_jobs libstrongswan/processing/processor.c
```

Thread 2 update the entry while thread 1 read the entry.

It seems that changing libcharon/sa/trap_manager.c:364 from this->lock->read_lock(this->lock); to this->lock->write_lock(this->lock); will solve it.

Thanks,
Avinoam

#3 - 09.07.2015 18:51 - Tobias Brunner

> I tested the branch and it resolves the data race.

Great, thanks for testing.

> However there is another data race.
>
> [...]
>
> Thread 2 update the entry while thread 1 read the entry.

I think I used a read lock on line 369 because in the usual use case (an acquire triggering an IKE_SA/CHILD_SA) it is quite unlikely that complete() is called before the thread returning from initiate() can update the entry (due to the RTT to negotiate the SA) and it is slightly more efficient this way. The lock is basically to make sure the list itself is not modified while looking for the entry.

In your case the concurrent call to complete(), causing the conflicting access, is due to the local destruction of IKE_SA objects (via flush() of the IKE_SA manager) during shutdown. But I suppose this kind of race could theoretically also occur during regular operation. If this is actually a problem depends on whether complete() was called for the IKE_SA on which the CHILD_SA was initiated in acquire(). If it is, Thread 1 might even access the ike_sa member of the entry before Thread 2 updated it, which would cause the pending flag not to get reset (if this happens during shutdown it's not really an issue, but if not, further acquires for the same trap policy will be prevented). This race is not resolved by using a write lock on line 364.

I actually changed the affected code in the *trap-any* branch some time ago (7301167424ca). The changes there were mainly done to track multiple acquires for the same trap policy later (402c5a8c1701). But they also prevent the concurrent access to the entries. But the situation where Thread 1 is faster than Thread 2 is handled similarly there, that is, the acquire_t entry would just remain in the list of pending acquires until the trap manager is flushed. In the meantime further acquires for the same trap policy would be blocked. There is even a scenario that is handled worse, which is a call to flush() of the trap manager while another thread is just about to update the acquire_t instance after the initiate() call (just before it can acquire the mutex). The instance will then already be destroyed when it finally attempts to update it.

I changed the code from the first commit in the *trap-any* branch a bit to account for these scenarios, refer to the commits in the *trap-acquire-tracking* branch (they also add or change the error handling for corner cases).

#4 - 12.07.2015 08:50 - Avinoam Meir

Great thanks for the code changes.

I tested also the trap-manager branch and it is fix the data race I described in #2. thank you.

But... I just now understand that there is segmentation fault during the libcharon_deinit (I hope this is the last one). this fault occurs regardless the change in the trap manager.

```
Thread 1 call stack
  __pthread_rwlock_wrlock
  libcharon/sa/trap_manager.c install
  libcharon/processing/jobs/start_action_job.c execute
  libstrongswan/processing/processor.c process_job
  libstrongswan/processing/processor.c process_jobs
  libstrongswan/threading/thread.c thread_main
```

The thread waits in the lock at line 200 - in master or line 255 in trap_manager branch

```
Thread 2 call stack
  libcharon/sa/trap_manager.c:115 destroy_entry
  libstrongswan/collections/linked_list.c destroy_function
  libcharon/sa/trap_manager.c flush
  libcharon/daemon.c destroy
  libcharon/daemon.c libcharon_deinit
```

There is segmentation fault because entry->child_sa is empty.

Thank you very much,
Avinoam

#5 - 13.07.2015 13:58 - Tobias Brunner

> I tested also the trap-manager branch and it is fix the data race I described in #2.

Thanks for testing.

> But... I just now understand that there is segmentation fault during the libcharon_deinit (I hope this is the last one). this fault occurs regardless the change in the trap manager.
>
> [...]
>
> The thread waits in the lock at line 200 - in master or line 255 in trap_manager branch
>
> [...]

> There is segmentation fault because entry->child_sa is empty.

This is a similar issue to the one I just fixed. The problem is that the entry_t instance created by Thread 1 in install() is concurrently destroyed by Thread 2 in flush() while the lock is not held by Thread 1, so when the latter then attempts to update the entry the pointer will not be valid anymore.

I removed the refcounting and simplified the acquire tracking and implemented something similar for the trap entries (i.e. in flush() skip the entries that have no IKE/CHILD_SA assigned yet). The skipped entries will remain in the lists until destroy() is called, but that should not be a problem as it happens during shutdown. See the additional commits in the *trap-acquire-tracking* branch.

#6 - 13.07.2015 18:54 - Avinoam Meir

I tested the additional commits and everything works, thank you.

#7 - 14.07.2015 17:19 - Tobias Brunner

Thanks again for testing. I've pushed the RNG fix for the IKE_SA manager to master already.

Regarding the other changes, I realized that we can avoid clearing the acquires list in flush() altogether, as its entries have no external dependencies. However, this is different for the entries in the traps list. The whole point of "flushing" them is to do so before the plugins that provide the kernel-interface implementations are unloaded. Otherwise, the CHILD_SA objects assigned to them will not be able to uninstall the trap policies in the kernel.

I also saw that we need something similar for the shunt policy manager, which currently has no flush() method at all, so it fails to properly uninstall shunt policies (this can be seen in the test cases, e.g. at the end of alice's daemon.log in ikev2/shunt-policies-nat-rw). They are only cleared later by starter, which flushes all states and policies when it terminates. There is also no locking at all in shunt_manager_t.

I rebased the *trap-acquire-tracking* to the current master and replaced the last commit with one that prevents the issue in the trap manager mentioned above but still handles the reported race condition correctly. And I added a few commits to fix the shunt manager.

#8 - 20.07.2015 12:44 - Avinoam Meir

I tested the new commits on the trap-manager in trap-acquire-tracking branch, and they work well.
I was not able to test the changes in shunt manager because I use StrongSwan 5.1.1 and there is few more change in shunt manager files.

Thank you.

#9 - 27.07.2015 14:04 - Tobias Brunner

- Status changed from Feedback to Closed

- Resolution set to Fixed

> I tested the new commits on the trap-manager in trap-acquire-tracking branch, and they work well.
> I was not able to test the changes in shunt manager because I use StrongSwan 5.1.1 and there is few more change in shunt manager files.

OK, thanks. I just merged these changes to master.

#10 - 28.08.2015 16:04 - Tobias Brunner

- Subject changed from Possible data race when destroying libcharon to Possible data race when destroying libcharon (multiple issues in ike-sa-manager, trap-manager and shunt-manager)

Files

0001-Avoid-data-race-in-libcharon_deinit.patch	2.05 KB	30.06.2015	Avinoam Meir