strongSwan - Bug #2437

StrongSwan hangs on exit when using opensc-pkcs11.so PKCS#11 library...

04.10.2017 10:41 - Luka Logar

Status:	Closed	**Start date:**	04.10.2017
Priority:	Normal	**Due date:**	
Assignee:	Tobias Brunner	**Estimated time:**	0.00 hour
Category:	libstrongswan		
Target version:	5.6.1		
Affected version:	5.6.0	**Resolution:**	Fixed

Description

Hi,

strongSwan hangs on exit when *opensc-pkcs11.so* PKCS#11 library is used. I think this happens because thread with a blocked C_WaitForSlotEvent() call is forcefully destroyed before C_WaitForSlotEvent() gets cancelled (by calling C_Finalize() from the main thread). As a consequence, an internal mutex inside *pcsc-lite* library is held locked and a call to C_Finalize() hangs.
Klaus Richter reported similar issue on the *strongSwan-dev* mailing list (with another PKCS#11 library), so this is possibly true also for other PKCS#11 libraries.

The fix is to call C_Finalize() from the job cancel function instead of forcefully terminating the thread/job.

Associated revisions

Revision 6ce7ae24 - 02.11.2017 10:15 - Tobias Brunner

pkcs11: Call C_Finalize() to cancel jobs waiting in C_WaitForSlotEvent()

This is not ideal as the call to C_Finalize() should be the last one via
the PKCS#11 API. Since the order in which jobs are canceled is undefined
we can't be sure there is no other thread still using the library (it could
even be the canceled job that still handles a previous slot event).
According to PKCS#11 the behavior of C_Finalize() is undefined while other
threads still make calls over the API.

However, canceling the thread, as done previously, could also be problematic
as PKCS#11 libraries could hold locks while in the C_WaitForSlotEvent() call,
which might not get released properly when the thread is just canceled,
and which then might cause later calls to other API functions to block.

Fixes #2437.

History

#1 - 06.10.2017 15:25 - Tobias Brunner

- Category set to libstrongswan

- Status changed from New to Feedback

- Assignee set to Tobias Brunner

- Target version set to 5.6.1

> I think this happens because thread with a blocked C_WaitForSlotEvent() call is forcefully destroyed before C_WaitForSlotEvent() gets cancelled (by calling C_Finalize() from the main thread).

If the PKCS#11 library is not implemented in a way that a thread waiting in C_WaitForSlotEvent may safely be canceled (e.g. via pthread_cancel) this could probably happen. Ideally, the library sets up its own blocking call in a way that any necessary cleanup is performed when the calling thread is canceled (e.g. by releasing locks via pthread_cleanup_push). But I guess the PKCS#11 standard does not mandate libraries are implemented this way (it might also depend on whether OS locking is used or not).

> The fix is to call C_Finalize() from the job cancel function instead of forcefully terminating the thread/job.

If wonder whether it's a problem if this is called multiple times (it does get called again once the pkcs11_library_t object is destroyed).

Also, calling C_Finalize should actually be the last call made via the API, not sure if that's guaranteed. The PKCS#11 standard says:

> ...the behavior of C_Finalize is nevertheless undefined if it is called by an application while other threads of the application are making Cryptoki calls. The exception to this exceptional behavior of C_Finalize occurs when a thread calls C_Finalize while another of the pkcs11 application's threads is blocking on Cryptoki's C_WaitForSlotEvent function.

So while the C_WaitForSlotEvent case is explicitly exempted, there could be other threads/jobs still working with keys on the token or using it for hashing or as RNG etc. (the order in which jobs are canceled is undefined), which might be problematic. It could even be the canceled job handling a previous slot event and calling the registered callback that still interacts with the API.

There does not seem to be any other way to get C_WaitForSlotEvent to return, though, so I pushed a patch to the *2437-pkcs11-finalize* branch (not sure if it's necessary to avoid calling C_Finalize twice, I guess it will usually just be a no-op if called again).

#2 - 10.10.2017 15:03 - Luka Logar

I've tried your patch with 3 different PKCS#11 libraries (all on Ubuntu 16.04.03):

- *eToken* card using *libeTPkcs11.so*,
- *SC-HSM card* using *opensc-pkcs11.so* and
- *Gemalto .NET 2.0 card* using *libgtop11dotnet.so*

eToken hangs on exit if patch is not applied. It also doesn't care if C_Finalize() is called once or twice and exits nicely with patch applied.

It seems that stock Ubuntu *opensc* does not have C_WaitForSlotEvent() implemented as strongSwan returns

```
module 'opensc' does not support hot-plugging, cancelled
```

and is thus unaffected by this bug/patch.

opensc (and *pcsc-lite* and *ccid*) compiled from source hang on exit if patch is not applied and exit fine with patch applied. Also it doesn't matter if C_Finalize() is called once or twice.

Gemalto's PKSC#11 library however works fine even without the patch, but crashes if C_Finalize() is called only once and exits normally if C_Finalize() is called twice...

#3 - 10.10.2017 15:25 - Tobias Brunner

> I've tried your patch with 3 different PKCS#11 libraries (all on Ubuntu 16.04.03):
>
> - *eToken* card using *libeTPkcs11.so*,
> - *SC-HSM card* using *opensc-pkcs11.so* and
> - *Gemalto .NET 2.0 card* using *libgtop11dotnet.so*

Thanks a lot for the tests.

> Gemalto's PKSC#11 library however works fine even without the patch, but crashes if C_Finalize() is called only once and exits normally if C_Finalize() is called twice...

Hm, interesting. Maybe because I set the function pointer to NULL? (Perhaps the library internally calls C_Finalize via that pointer, just to make sure in case the application didn't call it.) Anyway, I guess we can avoid that and just go with calling it twice. I've updated the commit.

#4 - 02.11.2017 10:16 - Tobias Brunner

- Status changed from Feedback to Closed

- Resolution set to Fixed

Files

pkcs11-fix-hang-on-exit.patch	636 Bytes	04.10.2017	Luka Logar