strongSwan - Bug #298

Deadlock on charon deinit

25.02.2013 22:47 - Stefan Tomas

Status:	Closed	**Start date:**	25.02.2013
Priority:	Normal	**Due date:**	
Assignee:	Tobias Brunner	**Estimated time:**	0.00 hour
Category:	android		
Target version:	5.0.3		
Affected version:	dr|rc|master	**Resolution:**	Fixed

Description

I have noticed that occasionally Android strongswan client would fail to properly disconnect. It would remain in disconnecting state until process was forcefully killed via OS. Since logs didn't give any clue what might be wrong I started investigating the problem in more detail. I added a lot of logs in the de-init code and this is what I found...

I found that process would become deadlocked if certain operations were taking longer than usual. Deadlock can be easily recreated if a pause is placed in socket_default_socket sender function for example (a sleep(1) in that function would cause certain deadlock at de-init). Deep analysis showed that sender thread was canceled while it held synchronization object locked, which later caused deadlock when destroy function waited to get a lock on that object. But this should not happen if thread was not cancelable in that moment, as it should be. So I focused on threading, mainly thread.c
First I found that there is a bug in thread_cancelability() function (in #else part, when there is no native pthread_cancel()):

```
    sigset_t new, old;

    sigemptyset(&new);
    sigaddset(&new, SIG_CANCEL);
    pthread_sigmask(enable ? SIG_UNBLOCK : SIG_BLOCK, &new, &old);

    return sigismember(&old, SIG_CANCEL) == 0;
```

According to sigismember() man page, sigismember() returns 1 if signum is a member of set, 0 if signum is not a member, and -1 on error.
So the return statement should have been:

```
    return sigismember(&old, SIG_CANCEL) == 1;
```

Alas, this did not fix the deadlock. Moreover, when I added log (__android_log_print) immediately after pthread_sigmask() I was getting a crash?!
Then I looked into SIG_CANCEL definition:

```
#define SIG_CANCEL (SIGRTMIN+7)
```

And in platfrom's asm/signal.h:

```
typedef unsigned long sigset_t;
...
#define SIGRTMIN 32
#define SIGRTMAX _NSIG
```

and signal.h:

```
#ifndef _NSIG
#  define _NSIG  64
#endif
...
static __inline__ int sigaddset(sigset_t *set, int signum)
{
```

```
    unsigned long *local_set = (unsigned long *)set;
    signum--;
    local_set[signum/LONG_BIT] |= 1UL << (signum%LONG_BIT);
    return 0;
}
```

So, after all above is considered, we can see there is a bug in Android NDK. This is because sigset_t is 32-bit, according to definition, but SIGRTMAX indicates that signals up to 64 are supported. But, clearly 32-bit set cannot hold mask for more than 32 different signals. Moreover, signal manipulation functions are all going to cause memory corruption if signal id is larger than 31, and effectively real-time signals are not supported on Android.
This is more or less confirmed here: https://code.google.com/p/android/issues/detail?id=43040
The effect of all this was that client was able to send SIG_CANCEL and threads were indeed being cancelled, but the signal could not have been blocked or unblocked, braking the cancelability logic.
Anyway, after realizing this, changing SIG_CANCEL definition to:

```
#ifdef ANDROID
#define SIG_CANCEL (SIGUSR2)
#else
#define SIG_CANCEL (SIGRTMIN+7)
#endif
```

, and using -DANDROID CFLAG, together with thread_cancelability() fix above, seems that resolved this problem. (note that SIG_USR2 was unused by strongswan, so it appears to be best candidate for SIG_CANCEL definition)

Related issues:

Related to Issue #609: Strongswan fails to disconnect - Android client	**Closed**	**05.06.2014**

Associated revisions

Revision bc07fef0 - 26.02.2013 11:40 - Tobias Brunner

Use SIGUSR2 for SIG_CANCEL on Android

SIGRTMIN is defined as 32 while sigset_t is defined as unsigned long (i.e. holds 32 signals). Hence, the signal could never be blocked. Sending the signal still canceled threads, but sometimes in situations where they shouldn't have been canceled (e.g. while holding a lock).

Fixes #298.

History

#1 - 26.02.2013 11:44 - Tobias Brunner

- Status changed from New to Closed

- Assignee set to Tobias Brunner

- Target version set to 5.0.3

- Affected version changed from 5.0.2 to dr|rc|master

- Resolution set to Fixed

Thanks for the detailed report.

> First I found that there is a bug in thread_cancelability() function (in #else part, when there is no native pthread_cancel()):
> [...]
> return sigismember(&old, SIG_CANCEL) == 0;
> According to sigismember() man page, sigismember() returns 1 if signum is a member of set, 0 if signum is not a member, and -1 on error.

This is not a bug. thread_cancelability() returns the state of the current thread's cancelability, before it got modified. That is, it returns true if the thread could be canceled before calling thread_cancelability() and false otherwise. In case of the signal-based implementation a thread is cancelable if SIG_CANCEL is **not** blocked. That is, the signal **must not** be part of the thread's signal mask. Therefore, the comparison above is correct. true is returned if SIC_CANCEL was not blocked before and false if it was (or if there is an error, that's kind of a bug, but how would we recover from that situation?).

> So, after all above is considered, we can see there is a bug in Android NDK. This is because sigset_t is 32-bit, according to definition, but SIGRTMAX indicates that signals up to 64 are supported. But, clearly 32-bit set cannot hold mask for more than 32 different signals. Moreover, signal manipulation functions are all going to cause memory corruption if signal id is larger than 31, and effectively real-time signals are not supported on Android.

> This is more or less confirmed here: https://code.google.com/p/android/issues/detail?id=43040
> The effect of all this was that client was able to send SIG_CANCEL and threads were indeed being cancelled, but the signal could not have been blocked or unblocked, braking the cancelability logic.

That's very interesting, and unfortunate.

A problem with the workaround could be that Dalvik (the JVM) already uses SIGUSR1 and SIGUSR2 (which, I think, was the reason to base SIG_CANCEL on SIGRTMIN in the first place). Since we send the signal directly to threads that we created ourselves, this might not really be an issue, though.

> Anyway, after realizing this, changing SIG_CANCEL definition to:
> #ifdef ANDROID
> #define SIG_CANCEL (SIGUSR2)
> #else
> #define SIG_CANCEL (SIGRTMIN+7)
> #endif

I suppose that's the only way of action for now. I pushed a fix to master.

#2 - 26.02.2013 13:21 - Stefan Tomas

Agreed, somewhere along the way I forgot that pthread_sigmask() was using sigset_t enabled bit (1) for BLOCKED signals, while I assumed that it was otherwise. Thanks for the update.

#3 - 04.06.2014 16:48 - Le Hoang

Hello, we have been your Android StrongSwan VPN client and have noticed that the app is exhibiting similar behaviour to the one described above. We're using a Samsung Galaxy tab 2 10.1 on Android Version 4.0.4. When attempting to disconnect, the app will just remain in the disconnecting state. This behaviour is not shown on any other of our Android devices including: Samsung Galaxy S5, Nexus 4, Samsung Galaxy S4, Samsung Galaxy S3 and HTC One X. Android Versions ranging from 4.2.2 - 4.4.2.

#4 - 05.06.2014 11:09 - Tobias Brunner

- Related to Issue #609: Strongswan fails to disconnect - Android client added