strongSwan - Bug #3164

Retransmit rekeying uses remote address before mobike event occurs

02.09.2019 16:13 - Michaël Legagneur

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Tobias Brunner	**Estimated time:**	0.00 hour
Category:	kernel-interface		
Target version:	5.9.1		
Affected version:	5.8.0	**Resolution:**	Fixed

Description

Hello,

We observed an issue when using a client IKEv2 on strongswan 5.8.0 with two interfaces and mobike enable.
The scenario is the following : the client has an instable primary link and triggers mobike on the secondary link during rekeying initiated by the server.
To reproduce the issue, we uses testbed where we dropped packet on the primary link (10.1.3.7 on our logs at the server side).

```
Mon, 2019-09-02 14:31 03[ENC] <backbone|1> generating CREATE_CHILD_SA request 0 [ N(REKEY_SA) N(IP
COMP_SUP) SA No TSi TSr ]
Mon, 2019-09-02 14:31 03[NET] <backbone|1> sending packet: from 10.1.3.20[4500] to 10.1.3.7[4500]
(272 bytes)
Mon, 2019-09-02 14:31 05[IKE] <backbone|1> retransmit 1 of request with message ID 0
Mon, 2019-09-02 14:31 05[NET] <backbone|1> sending packet: from 10.1.3.20[4500] to 10.1.3.7[4500]
(272 bytes)
Mon, 2019-09-02 14:31 13[IKE] <backbone|1> retransmit 2 of request with message ID 0
Mon, 2019-09-02 14:31 13[NET] <backbone|1> sending packet: from 10.1.3.20[4500] to 10.1.3.7[4500]
(272 bytes)
```

During these retransmissions, the current primary link goes down so it requests mobike to use the secondary interface (10.1.3.8).
This mobike is successfully done at the server side, the remote address used is 10.1.3.8.

```
Mon, 2019-09-02 14:31 17[ENC] <backbone|1> parsed INFORMATIONAL request 3 [ N(UPD_SA_ADDR) N(NATD_
S_IP) N(NATD_D_IP) N(COOKIE2) N(ADD_4_ADDR) ]
Mon, 2019-09-02 14:31 17[IKE] <backbone|1> got additional MOBIKE peer address: 192.168.1.1
Mon, 2019-09-02 14:31 17[CHD] <backbone|1> CHILD_SA net{1} state change: REKEYING => UPDATING
..
Mon, 2019-09-02 14:31 17[KNL] <backbone|1> updating SAD entry with SPI cf7cf832 from 10.1.3.7[4500
]..10.1.3.20[4500] to 10.1.3.8[4500]..10.1.3.21[4500]
Mon, 2019-09-02 14:31 17[KNL] <backbone|1> querying SAD entry with SPI c33187b9 for update
Mon, 2019-09-02 14:31 17[KNL] <backbone|1> querying replay state from SAD entry with SPI c33187b9
Mon, 2019-09-02 14:31 17[KNL] <backbone|1> deleting SAD entry with SPI c33187b9
Mon, 2019-09-02 14:31 17[KNL] <backbone|1> deleted SAD entry with SPI c33187b9
Mon, 2019-09-02 14:31 17[KNL] <backbone|1> updating SAD entry with SPI c33187b9 from 10.1.3.20[450
0]..10.1.3.7[4500] to 10.1.3.21[4500]..10.1.3.8[4500]
```

The server continues to retransmit rekey toward the secondary link, and after client response, the server processes the rekeying with the first remote address (10.1.3.7) instead of the new one, 10.1.3.8.

```
Mon, 2019-09-02 14:31 16[CHD] <backbone|1> CHILD_SA net{2} state change: CREATED => INSTALLING
Mon, 2019-09-02 14:31 16[CHD] <backbone|1>   using AES_CBC for encryption
Mon, 2019-09-02 14:31 16[CHD] <backbone|1>   using HMAC_SHA2_256_128 for integrity
Mon, 2019-09-02 14:31 16[CHD] <backbone|1> adding inbound ESP SA
Mon, 2019-09-02 14:31 16[CHD] <backbone|1>   SPI 0xc30b09eb, src 10.1.3.7 dst 10.1.3.20
Mon, 2019-09-02 14:31 16[KNL] <backbone|1> adding SAD entry with SPI c30b09eb and reqid {1}
Mon, 2019-09-02 14:31 16[KNL] <backbone|1>   using encryption algorithm AES_CBC with key size 128
Mon, 2019-09-02 14:31 16[KNL] <backbone|1>   using integrity algorithm HMAC_SHA2_256_128 with key
size 256
Mon, 2019-09-02 14:31 16[KNL] <backbone|1>   using replay window of 32 packets
Mon, 2019-09-02 14:31 16[KNL] <backbone|1>   HW offload: no
Mon, 2019-09-02 14:31 16[CHD] <backbone|1> adding outbound ESP SA
```

```
Mon, 2019-09-02 14:31 16[CHD] <backbone|1>   SPI 0xc14b0d45, src 10.1.3.20 dst 10.1.3.7
Mon, 2019-09-02 14:31 16[KNL] <backbone|1> adding SAD entry with SPI c14b0d45 and reqid {1}
Mon, 2019-09-02 14:31 16[KNL] <backbone|1>   using encryption algorithm AES_CBC with key size 128
Mon, 2019-09-02 14:31 16[KNL] <backbone|1>   using integrity algorithm HMAC_SHA2_256_128 with key
size 256
Mon, 2019-09-02 14:31 16[KNL] <backbone|1>   using replay window of 0 packets
Mon, 2019-09-02 14:31 16[KNL] <backbone|1>   HW offload: no
Mon, 2019-09-02 14:31 16[KNL] <backbone|1> policy 10.4.0.1/32 === 10.1.4.0/24 in already exists, i
ncreasing refcount
Mon, 2019-09-02 14:31 16[KNL] <backbone|1> updating policy 10.4.0.1/32 === 10.1.4.0/24 in [priorit
y 371327, refcount 2]
Mon, 2019-09-02 14:31 16[KNL] <backbone|1> policy 10.4.0.1/32 === 10.1.4.0/24 fwd already exists,
increasing refcount
Mon, 2019-09-02 14:31 16[KNL] <backbone|1> updating policy 10.4.0.1/32 === 10.1.4.0/24 fwd [priori
ty 371327, refcount 2]
Mon, 2019-09-02 14:31 16[IKE] <backbone|1> inbound CHILD_SA net{2} established with SPIs c30b09eb_
i c14b0d45_o and TS 10.1.4.0/24 === 10.4.0.1/32
Mon, 2019-09-02 14:31 16[CHD] <backbone|1> CHILD_SA net{2} state change: INSTALLING => INSTALLED
Mon, 2019-09-02 14:31 16[KNL] <backbone|1> policy 10.1.4.0/24 === 10.4.0.1/32 out already exists,
increasing refcount
Mon, 2019-09-02 14:31 16[KNL] <backbone|1> updating policy 10.1.4.0/24 === 10.4.0.1/32 out [priori
ty 371327, refcount 2]
Mon, 2019-09-02 14:31 16[IKE] <backbone|1> outbound CHILD_SA net{2} established with SPIs c30b09eb
_i c14b0d45_o and TS 10.1.4.0/24 === 10.4.0.1/32
Mon, 2019-09-02 14:31 16[CHD] <backbone|1> CHILD_SA net{1} state change: REKEYING => REKEYED
```

It seems that it could due that child_create task, the child_sa is created with the remote_address at the build initiator method and didn't update before installing it at the process_i method.

```
METHOD(task_t, build_i, status_t,
    private_child_create_t *this, message_t *message)
{
...
    this->child_sa = child_sa_create(this->ike_sa->get_my_host(this->ike_sa),
            this->ike_sa->get_other_host(this->ike_sa), this->config, this->reqid,
            this->ike_sa->has_condition(this->ike_sa, COND_NAT_ANY),
            this->mark_in, this->mark_out);

    if (!allocate_spi(this))
    {
        DBG1(DBG_IKE, "unable to allocate SPIs from kernel");
        return FAILED;
    }
```

What could be the best fix ? I have updated the local and remote address of child_sa with dedicated method during the process_i method but I am not sure that was the best place.

Thanks,

Michaël

Associated revisions

Revision bce0c5fd - 27.10.2020 16:45 - Tobias Brunner

child-create: Update CHILD_SA IP addresses before installation

We create the child_sa_t object when initiating the CREATE_CHILD_SA request, however, the IP addresses/ports might have changed once we eventually receive the response (potentially to a retransmit sent to a different address). So update them before installing the SA and policies.

If the local address changed too and depending on the kernel implementation, the temporary SA created to allocate the inbound SPI might remain as it can't be updated. This could cause issues if e.g. the address switches back before that SA expired (the updated inbound SA conflicts with the temporary one), or if that happens close together

and the expire (having to wait for the address update) causes the updated SA to get deleted.

Fixes #3164.

History

#1 - 13.10.2020 14:39 - Frédéric Martinsons

Hello there,

are there any news about this issue ? I mean, is the patch suggested is the correct one to be applied ?

#2 - 15.10.2020 15:44 - Tobias Brunner

- Tracker changed from Issue to Bug

- Category changed from libcharon to kernel-interface

- Status changed from New to Feedback

- Target version set to 5.9.1

> I mean, is the patch suggested is the correct one to be applied ?

There are some potential issues. The IP addresses that are passed to the constructor are used when allocating an SPI for the inbound SA from the kernel (this has to be done before sending the CREATE_CHILD_SA request as it is contained in the proposals). This usually creates a temporary SA in the kernel to keep track of it. If the addresses are later updated before installing/updating the inbound SA, the kernel might not be able to find the temporary SA. The tuple SPI, destination address and protocol is usually used for this lookup, so for the inbound SA that should work if the local address did not change (as in the example above). But the MOBIKE client could also switch to a different server address if multiple are available (and if a kernel uses both addresses to identify SAs that would not work either). This might not necessarily be a problem, it could just create a "duplicate" SA with different addresses and the same SPI. The temporary SA will eventually expire but if the addresses are switched back until that happens, there would be a conflict (the newly updated SA would have the same address and SPI as the existing temporary one). The expiration also causes an event in the kernel and the daemon, which might or might not be an issue, depending on the addresses that are in use at that time of the event (in the worst case the expire could cause the SA to get deleted).

With some additional changes, we could probably call child_sa_t::update() with the new addresses before installing the SAs and policies, which does nothing if the addresses did not change. Not all kernel interfaces support updating addresses on SAs (in which case the addresses are currently not updated on the child_sa_t object) and it might not work at all for temporary SAs that were created implicitly when an SPI was allocated (there might not even be any such temporary SAs). So there is probably no point in attempting to actually update the inbound SA (the outbound SA won't be updated until installed). I pushed some patches to this effect to the *3164-mobike-childsa* branch.

#3 - 21.10.2020 15:26 - Frédéric Martinsons

Tobias Brunner wrote:

> There are some potential issues. The IP addresses that are passed to the constructor are used when allocating an SPI for the inbound SA from the kernel (this has to be done before sending the CREATE_CHILD_SA request as it is contained in the proposals). This usually creates a temporary SA in the kernel to keep track of it. If the addresses are later updated before installing/updating the inbound SA, the kernel might not be able to find the temporary SA. The tuple SPI, destination address and protocol is usually used for this lookup, so for the inbound SA that should work if the local address did not change (as in the example above). But the MOBIKE client could also switch to a different server address if multiple are available (and if a kernel uses both addresses to identify SAs that would not work either). This might not necessarily be a problem, it could just create a "duplicate" SA with different addresses and the same SPI. The temporary SA will eventually expire but if the addresses are switched back until that happens, there would be a conflict (the newly updated SA would have the same address and SPI as the existing temporary one). The expiration also causes an event in the kernel and the daemon, which might or might not be an issue, depending on the addresses that are in use at that time of the event (in the worst case the expire could cause the SA to get deleted).
>
> With some additional changes, we could probably call child_sa_t::update() with the new addresses before installing the SAs and policies, which does nothing if the addresses did not change. Not all kernel interfaces support updating addresses on SAs (in which case the addresses are currently not updated on the child_sa_t object) and it might not work at all for temporary SAs that were created implicitly when an SPI was allocated (there might not even be any such temporary SAs). So there is probably no point in attempting to actually update the inbound SA (the outbound SA won't be updated until installed). I pushed some patches to this effect to the *3164-mobike-childsa* branch.

Thank you very much for the detailed answer and the commits you provided.

#4 - 27.10.2020 16:45 - Tobias Brunner

- Status changed from Feedback to Closed

- Assignee set to Tobias Brunner

- Resolution set to Fixed

Files

charon_debug.log	16.4 KB	02.09.2019	Michaël Legagneur
swanctl-client.conf	401 Bytes	02.09.2019	Michaël Legagneur
swanctl-server.conf	612 Bytes	02.09.2019	Michaël Legagneur