strongSwan - Bug #2388

The symmetric key (enc cbc(aes)) can be retrieved in a memory dump after a tunnel was closed

20.07.2017 17:47 - Roman Moschenski

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Tobias Brunner	**Estimated time:**	0.00 hour
Category:	kernel-interface		
Target version:	5.6.0		
Affected version:	5.3.3	**Resolution:**	Fixed

Description

Simple test:
connect 2 hosts (IKEv2) and follow the 4 steps

------host1----------		----------host2----------
1. ipsec up my_connection		
		2. ip xfrm state #save both keys
3. ipsec down my_connection		
4. make RAM dump		
----------		----------

In a created memory dump (whole RAM) both symmetric keys can be found.
Is the sensitive data in charon zeroed after a connection was closed?
Or may be my Linux Kernel (3.16) do not do it?

Associated revisions

Revision e0c78d75 - 07.08.2017 16:55 - Tobias Brunner

prf-plus: Wipe seed and internal buffer

The buffer contains key material we handed out last and the seed can
contain the DH secret.

References #2388.

Revision 1a75514b - 07.08.2017 16:55 - Tobias Brunner

sha2: Write final hash directly to output buffer

This avoids having the last output in internal memory that's not wiped.

References #2388.

Revision 6fadc6a8 - 07.08.2017 16:55 - Tobias Brunner

kernel-netlink: Wipe buffer used to read Netlink messages

When querying SAs the keys will end up in this buffer (the allocated
messages that are returned are already wiped). The kernel also returns
XFRM_MSG_NEWSA as response to XFRM_MSG_ALLOCSPI but we can't distinguish
this here as we only see the response.

References #2388.

History

#1 - 24.07.2017 10:12 - Roman Moschenski

In a core dump of charon both keys can be found (after ipsec down my_connection).

#2 - 24.07.2017 11:15 - Tobias Brunner

- Status changed from New to Feedback

> In a created memory dump (whole RAM) both symmetric keys can be found.

How did you do the memory dumps exactly?

> Is the sensitive data in charon zeroed after a connection was closed?

Yes (the derived CHILD_SA keys are cleared immediately after installation in the kernel, the keymat they are based on obviously only once the IKE_SA is terminated, unless there was a separate DH exchange when the CHILD_SA was created).

> Or may be my Linux Kernel (3.16) do not do it?

As far as I can see, the kernel doesn't (just uses kmemdup() and kfree() to copy and free the memory for aalg and ealg). So if it wasn't overwritten yet with other stuff it might still be there after the SA is terminated.

> In a core dump of charon both keys can be found (after ipsec down my_connection).

What strongSwan version? Sensitive memory is usually overwritten with zeros using the memwipe function (source:src/libstrongswan/utils/utils/memory.h#L118, also indirectly via chunk_clear), which tries to make sure that compilers don't remove calls to it (which are usually the last ones accessing a particular memory location in a function and compilers might think, since it's not used afterwards, they can eliminate the call). There is also a runtime check that tries to verify that this actually works (source:src/libstrongswan/library.c#L272). These particular keys are cleared both in the child-create task (source:src/libcharon/sa/ikev2/tasks/child_create.c#L735) and in the buffer in the kernel-netlink plugin (source:src/libcharon/plugins/kernel_netlink/kernel_netlink_ipsec.c#L1694).

#3 - 24.07.2017 17:14 - Roman Moschenski

The whole RAM dump is made using this simple KO:
https://github.com/504ensicsLabs/LiME

Binary key search, using this tool:
https://github.com/tmbinc/bgrep

My fault -> "In a core dump of charon both keys can be found (after ipsec down my_connection)."
Only one of both keys can be found in the GDB dump of charon:

The core dump of charon is made by:
gdb /usr/lib/ipsec/charon `pidof charon`
(gdb) generate-core-file charon.dump

#on the other side------->>>>>>>>>>>>>>>

```
~$ip xfrm state
src 11.222.0.40 dst 11.222.0.7
    proto esp spi 0xc86ae487 reqid 1 mode transport
    replay-window 32
    auth-trunc hmac(sha256) 0x6b7b935ae7c941c0327003fe4bb7db2de0b98f6813bde6c91e6300d051003209 128
    enc cbc(aes) 0x49b00a3456498055173da4a5bb362ab934f80b1e3dba1c8d61fc2a514fb88de8
    encap type espinudp sport 4500 dport 4500 addr 0.0.0.0
    sel src 0.0.0.0/0 dst 0.0.0.0/0
src 11.222.0.40 dst 11.222.0.7
    proto comp spi 0x0000bed6 reqid 1 mode tunnel
    replay-window 0 flag af-unspec
    comp deflate
src 11.222.0.40 dst 11.222.0.7
    proto 4 spi 0x0bde0028 reqid 0 mode tunnel
    replay-window 0 flag af-unspec
src 11.222.0.7 dst 11.222.0.40
    proto esp spi 0xce78c322 reqid 1 mode transport
    replay-window 32
    auth-trunc hmac(sha256) 0xeaf633064b5b8308f02932052a0ff861419dabf618b55fa96bc13cd5ea69b501 128
    enc cbc(aes) 0x3cbb50e48043181de4915b856d71cd7435958c445b366edc5c8fbd95aecc3b3b
    encap type espinudp sport 4500 dport 4500 addr 0.0.0.0
    sel src 0.0.0.0/0 dst 0.0.0.0/0
src 11.222.0.7 dst 11.222.0.40
    proto comp spi 0x000031a2 reqid 1 mode tunnel
    replay-window 0 flag af-unspec
    comp deflate
```

```
src 11.222.0.7 dst 11.222.0.40
    proto 4 spi 0x0bde0007 reqid 0 mode tunnel
```

#now create GDB coredump and search for the keys

#first key was not found

```
~$ bgrep 49b00a3456498055173da4a5bb362ab934f80b1e3dba1c8d61fc2a514fb88de8 /media/STICK/charon.dump
```

#second key found many times

```
~$ bgrep 3cbb50e48043181de4915b856d71cd7435958c445b366edc5c8fbd95aecc3b3b /media/STICK/charon.dump
/media/STICK/charon.dump: 000f2604
/media/STICK/charon.dump: 00140d94
/media/STICK/charon.dump: 001719b4
/media/STICK/charon.dump: 001911b4
/media/STICK/charon.dump: 001b18c4
/media/STICK/charon.dump: 001b2364
/media/STICK/charon.dump: 001d4774
/media/STICK/charon.dump: 00214f94
/media/STICK/charon.dump: 00236fe4
/media/STICK/charon.dump: 00299304
/media/STICK/charon.dump: 02385224
/media/STICK/charon.dump: 02b85224
/media/STICK/charon.dump: 03b85224
/media/STICK/charon.dump: 04385224
/media/STICK/charon.dump: 05385224
/media/STICK/charon.dump: 05b85224
/media/STICK/charon.dump: 06385224
/media/STICK/charon.dump: 08385224
```

Here is my ipsec.conf, on the side where the key is found:

```
config setup
conn %default
    forceencaps=yes
    ikelifetime=60m
    keylife=20m
    rekeymargin=3m
    keyingtries=1
    keyexchange=ikev2

conn my_connection
    auto=add
    type=tunnel
    leftcert=%smartcard1:3
    leftsendcert=always
    leftsourceip=%config
    reauth=no
    ike=aes256-sha256-modp2048!
    esp=aes256-sha256-modp2048!
    compress=yes
    right=11.222.0.40
    leftid="C=DE, ST=123, L=456, O=My Organization, CN=80276883120000011427-20170424, postalCode=12345, street
=Mystr. 20"
    rightid=%any
    rightsubnet=10.50.4.0/24,10.50.1.0/25,10.50.5.0/24,10.50.1.128/25
    leftsubnet=10.50.1.128/25
    dpdaction=clear
    dpddelay=30
    replay_window=0
```

I have just set up openSUSE-13.2 (3.16 Kernel) and have installed strongswan 5.1.3 from an official repositories. I have established an IKEv2 connection using PSK and yes, I can not find the keys... Is it possible that these memory wipe functions you have mentioned, were optimized out, as we have compiled Strongswan?
We are testing with a self compiled Strongswan 5.3.3 with an enabled PKCS11 Plugin. We are using a SmartCard on the side, where the key is found (but I think that does not matter)
Here is how strongswan is compiled:

```
                        ./configure --prefix=${GLOBAL_INSTALL_PATH}
                        --disable-aes
```

```
                        --disable-cmac
                        --disable-constraints
                        --disable-des
                        --disable-dnskey
                        --disable-fips-prf
                        --disable-gmp
                        --disable-hmac
                        --disable-ikev1
                        --disable-load-warning
                        --disable-md5
                        --disable-pgp
                        --disable-pgp
                        --disable-pkcs1
                        --disable-pkcs12
                        --disable-pkcs12
                        --disable-pkcs7
                        --disable-pkcs8
                        --disable-pubkey
                        --disable-pubkey
                        --disable-random
                        --disable-rc2
                        --disable-resolve
                        --disable-revocation
                        --disable-sha1
                        --disable-sha2
                        --disable-sshkey
                        --disable-static
                        --disable-x509
                        --disable-xauth-generic
                        --disable-xauth-generic
                        --disable-xcbc
                        --enable-curl
                        --enable-openssl
                        --enable-pkcs11
                        --host=${CMAKE_CNN_CROSS_ARCH}
                        --libdir=${GLOBAL_INSTALL_PATH}/lib
                        --libexecdir=${GLOBAL_INSTALL_PATH}/lib
                        --localstatedir=/tmp/bla
                        --sbindir=${GLOBAL_INSTALL_PATH}/bin
                        --with-systemdsystemunitdir=${GLOBAL_INSTALL_PATH}
```

#4 - 25.07.2017 10:49 - Tobias Brunner

> #second key found many times

That seems very odd. Each key is in two obvious locations at one point. One is on the heap once the key is derived and the other on the stack when the Netlink/XFRM message is prepared to be sent to the kernel. Both should get wiped (see links in my previous comment). Perhaps the key could also be in some socket buffer (Netlink/XFRM socket, over which we have no direct control), if that's contained in the dump, but I still don't see why there should be that many copies of it.

> Is it possible that these memory wipe functions you have mentioned, were optimized out, as we have compiled Strongswan?

It is, but why would that only affect one of the keys? And as mentioned that key should not be in memory that many times anyway. What architecture and compiler do you use?

I tried reproducing the issue in our testing environment. There we have our leak detective (LD) debugging tool activated, which has two important features regarding this issue: 1) each allocated block of heap memory has a distinct header and trailer, 2) newly allocated memory is initialized with 0xee and freed memory is overwritten with 0xff.

My memory dump contained only one instance of one of the integrity keys (on the initiator that of the inbound SA, on the responder that of the outbound SA) stored somewhere in heap memory. After terminating the SA this memory was apparently freed. The allocated block was for 152 bytes, however, in that block the encryption key was nowhere to be found. Looking at how the keys for CHILD_SAs are derived, I wondered if it could be related to that somehow.

Deriving the CHILD_SA keys is done via KEYMAT = prf+(SK_d, Ni | Nr) (RFC 7296, section 2.17), with prf+ defined as follows:

```
prf+(K,S) = T1 | T2 | T3 | T4 | ...

   where:
   T1 = prf(K, S | 0x01)
```

```
   T2 = prf(K, T1 | S | 0x02)
   T3 = prf(K, T2 | S | 0x03)
   T4 = prf(K, T3 | S | 0x04)
```

So the prf is initialized with SK_d as key (from the IKE key material) and then the seed (Ni | Nr or with DH in a CREATE_CHILD_SA exchange g^ir | Ni | Nr) is used with a counter byte and the previous block in each further call. When deriving the keys the generated key material is used in this order (from the perspective of the peer that initiated the CHILD_SA, vice-versa on the responder): encryption key outbound, integrity key outbound, encryption key inbound, integrity key inbound. This means that the last key (or at least parts of it, depending on the block size of the PRF and the required key material) is stored in the prf+ implementation (for the next potential iteration) and maybe also in the prf implementation (or the hash implementation on which it is based if its an HMAC). The prf+ instance is destroyed immediately after key derivation (however, its internal memory is strangely not wiped). However, the prf instance is only destroyed once the IKE_SA is (its state does change, though, when it is used for something else i.e. set_key() is called and new material is pulled from it).

Now, what are those 152 bytes exactly. It turns out they represent the private data of the SHA-256 hash implementation on which the PRF was based in this test scenario. Part of that is a 64 byte output buffer, that contained that key (was 32 byte long in my case).

I did the same test with leak detective disabled. And even though the prf+ state is not wiped it didn't show up. Even when doing the dump from a break point right after the destruction of it (my guess is that the memory gets overwritten by another thread, as parts of the key did show up).

The question is how significant all this is. First, the key found here is the integrity key (but depending on the block size of the PRF and the key lengths the internal state could probably also contain the encryption key or parts of it. **Edit:** Actually, with AEAD this key is the encryption and integrity key). Second, SK_d, on which the PRF is based, is in memory anyway and Ni/Nr for the initial key derivation are transmitted in public, and since the derived IKE keys are in memory too, Ni and Nr in later CREATE_CHILD_SA exchanges probably can't be considered secrets either (an attacker who wants to decrypt ESP packets this way will probably also have captured the IKE packets). So an attacker could probably derive the complete CHILD_SA key material this way anyway. With a DH exchange in a CREATE_CHILD_SA exchange the situation may be a bit different, so I think it would probably be a good idea to at least wipe the memory in the prf+ implementation when destroying it (the seed, which can contain the DH secret, wasn't wiped either). We could also advance the prf+ first, so the state of the prf is overwritten. I did so in the *2388-prf-wipe* branch.

The problem is that all this doesn't really explain what you are seeing. For which I currently don't have an explanation. I did also experiment with the *openssl* plugin, which you are using. The integrity key did not show up in full there, but latter parts of it did (I guess OpenSSL's HMAC or SHA-256 implementation is a bit different). The patches above remove those traces too. But how the encryption key could end up in memory, and that many times at that, I don't know. Maybe you could run bgrep with -C <bytes, e.g. 64> to get some context around these areas. Also try searching for the second half of the integrity keys to confirm my findings above.

#5 - 01.08.2017 19:05 - Roman Moschenski

- File charon_wipe.txt added

Our strongswan is cross compiled using a cross compiler: x86_64-unknown-linux-gnu
The target is x64.
I have tried a last strongswan version (5.5.3) with a changes you have made in "2388-prf-wipe" branch.
https://github.com/strongswan/strongswan/compare/2388-prf-wipe

I have also modified the memwipe source:src/libstrongswan/utils/utils/memory.h#L118
so that I can see what data is actually gets wiped.
So, let´s connect, here are the keys:

```
src 11.222.0.40 dst 11.222.0.7
    proto esp spi 0xc6b526a4 reqid 4 mode tunnel
    replay-window 32 flag af-unspec
    auth-trunc hmac(sha256) 0xee45a5b636758350155ac35fc02fdfaf6ec34332d89a4f272c2923e964f9c5a1 128
    enc cbc(aes) 0xdd3998c52d02ced9f3e7820417e0572d0a188f36d34b9e8effe5b45b82eed740
    encap type espinudp sport 4500 dport 4500 addr 0.0.0.0
src 11.222.0.7 dst 11.222.0.40
    proto esp spi 0xcfd40b40 reqid 4 mode tunnel
    replay-window 32 flag af-unspec
    auth-trunc hmac(sha256) 0x556069432829658c835cef2a4e1f544bba04e927571d2eeae3f4ff97261608e7 128
    enc cbc(aes) 0x78ad5f7e4e666aa4f9affdbc2ab0f44929d05197b1d6f2fc02f95f8ce801e642
    encap type espinudp sport 4500 dport 4500 addr 0.0.0.0
```

Let us take the second key only: 78ad5f7e4e666aa4f9affdbc2ab0f44929d05197b1d6f2fc02f95f8ce801e642
Now look in the charon_wipe.txt, I have attached.

The whole key (32 bytes) is wiped 1 time:

```
before 000032 : 78ad5f7e4e666aa4f9affdbc2ab0f44929d05197b1d6f2fc02f95f8ce801e642
after  000032 : 00000000000000000000000000000000000000000000000000000000000000000000
```

A bigger chunk 1024 bytes, that contains the key is wiped also 1 time.

But approximately every 3 seconds a chunk of data (584 bytes) is wiped with our key
"78ad5f7e4e666aa4f9affdbc2ab0f44929d05197b1d6f2fc02f95f8ce801e642" inside!!!

All these chunks differ in only one byte.

```
                  !!   here is our key
...480200001000000066...78ad5f7e4e666aa4f9affdbc2ab0f44929d05197b1d6f2fc02f95f8ce801e642...
...480200001000000068...78ad5f7e4e666aa4f9affdbc2ab0f44929d05197b1d6f2fc02f95f8ce801e642...
...48020000100000006a...78ad5f7e4e666aa4f9affdbc2ab0f44929d05197b1d6f2fc02f95f8ce801e642...
                  !!
```

After a one hour I had 500 key matches in my charon_wipe.txt

Any ideas?

Here is the binary data around the key in one of many key occurrences found in charon´s gdb core dump:

```
bgrep -C 128 78ad5f7e4e666aa4f9affdbc2ab0f44929d05197b1d6f2fc02f95f8ce801e642 /tmp/charon.core
/tmp/charon.core: 000cfc6c
...
/tmp/charon.core: 000f154c
...
/tmp/charon.core: 0012397c
/tmp/charon.core: 0013271c
/tmp/charon.core: 0017781c
/tmp/charon.core: 0019811c
/tmp/charon.core: 001ba21c
/tmp/charon.core: 001ba4cc
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x80\x00\x00\x00U`iC()e\x8c\x8
3\\xef*N\x1fTK\xba\x04\xe9'W\x1d.\xea\xe3\xf4\xff\x97&\x16\x08\xe7h\x00\x0
2\x00cbc(aes)\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00x\xad_~Nfj\xa4\xf9\xaf\xfd\xbc*\xb0\xf4I)
\xd0Q\x97\xb1\xd6\xf2\xfc\x02\xf9_\x8c\xe8\x01\xe6B\x1c\x00\x04\x00\xc03\x
0a\xd7\xb9\x7f\x00\x00q\x08\x00\x00\x00\x00\x00\x00x\x05\x00\x9c\xb9\x7f\x00\x00x\x05\x00\x9c\xb9\x7f\x00\x00\
xf0Q\x00\x9c\xb9\x7f\x00\x00\xf0Q\x00\x9c\xb9\x7f\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
/tmp/charon.core: 001d75bc
/tmp/charon.core: 0021960c
/tmp/charon.core: 0021984c
/tmp/charon.core: 0021d80c
/tmp/charon.core: 0029d79c
/tmp/charon.core: 002be2dc
/tmp/charon.core: 013937ec
/tmp/charon.core: 01b937ec
/tmp/charon.core: 023937ec
/tmp/charon.core: 033937ec
/tmp/charon.core: 03b937ec
/tmp/charon.core: 053937ec
/tmp/charon.core: 063937ec
/tmp/charon.core: 07b937ec
```

#6 - 02.08.2017 11:57 - Tobias Brunner

> The whole key (32 bytes) is wiped 1 time:

That's the allocated key, wiped after installing in the kernel.

> A bigger chunk 1024 bytes, that contains the key is wiped also 1 time.

That's probably the Netlink message used to install the SA.

> But approximately every 3 seconds a chunk of data (584 bytes) is wiped with our key "78ad5f7e4e666aa4f9affdbc2ab0f44929d05197b1d6f2fc02f95f8ce801e642" inside!!!

These might be caused by query_sa() calls. To get the number of transmitted packets and bytes the installed SAs are queried using XFRM_MSG_GETSA, which returns the complete SA including the keys. How often this is called depends on status queries (e.g. ipsec statusall).

> After a one hour I had 500 key matches in my charon_wipe.txt

But they are apparently wiped (unless memwipe() doesn't work as intended without your changes).

I added another two patches to the *2388-prf-wipe* branch. The SHA-2 implementation now writes the final hash directly to the output buffer, avoiding a local copy of it in internal memory. I also found a potential issue in the *kernel-netlink* plugin that's related to query_sa, as returned messages are first written to a buffer on the stack. You could try if that makes a difference.

#7 - 04.08.2017 17:40 - Roman Moschenski

- File charon_wipe.zip added

In our test environment we had "ipsec statusall" running every 3 seconds, so I have switched it off for this test.
Here are new results with your last commit in the 2388-prf-wipe branch. (changes in sha2_hasher.c has no effect, as we use openssl)

```
#on the other side
ip xfrm state

src 11.222.0.40 dst 11.222.0.7
proto esp spi 0xc21e9949 reqid 18 mode tunnel
replay-window 32 flag af-unspec
auth-trunc hmac(sha256) 0x9a4c3b60d33247d703236fa3fac96030bbe5de374b2ef7cd5f634a10749dfd5d 128
enc cbc(aes) 0xe8a9bc3728d2897478403c19a23326f9eee2310c0acd736ce42df9a8b5bf2ca8
encap type espinudp sport 4500 dport 4500 addr 0.0.0.0
src 11.222.0.7 dst 11.222.0.40
proto esp spi 0xcfdeb675 reqid 18 mode tunnel
replay-window 32 flag af-unspec
auth-trunc hmac(sha256) 0x147cab5fbfcbe1f8ae5cee71fff9c26e52195fa5371d7bbb96c63bedabd57867 128
enc cbc(aes) 0x6fd0e90e966c1efb67938be09a58a317d65c7cde2e16a469ad3067dcb69864f1
encap type espinudp sport 4500 dport 4500 addr 0.0.0.0
```

Both keys could still be found in a gdb core dump file of charon(after a connection was closed):

```
bgrep e8a9bc3728d2897478403c19a23326f9eee2310c0acd736ce42df9a8b5bf2ca8 /tmp/charon.core
/tmp/charon.core: 001b0434
bgrep 6fd0e90e966c1efb67938be09a58a317d65c7cde2e16a469ad3067dcb69864f1 /tmp/charon.core
/tmp/charon.core: 00149224
/tmp/charon.core: 0018d404
/tmp/charon.core: 001f20e4
/tmp/charon.core: 00232824
/tmp/charon.core: 00295564
/tmp/charon.core: 0033ad44
```

But it looks better than before, obviously the memwipe here makes a difference:

```
src/libcharon/plugins/kernel_netlink/kernel_netlink_shared.c
@@ -284,6 +284,7 @@ static bool read_and_queue(private_netlink_socket_t *this, bool block)
             hdr = NLMSG_NEXT(hdr, len);
         }
     }
+    memwipe(buf, this->buflen);
     return FALSE;
 }
```

Even though "ipsec statusall" does not run, there is still some periodic task that let a key appear every 30 seconds(chunks of 584 and 4096 bytes). After 10 minutes we have 40 wipes for this key(e8a9bc3728d2897478403c19a23326f9eee2310c0acd736ce42df9a8b5bf2ca8, found 1 time in the core dump).

```
15:42:42
plugins/libstrongswan-kernel-netlink.so(+0x38d9) [0x7f173c7408d9]
1024 Bytes to wipe : ...e8a9bc3728d2897478403c19a23326f9eee2310c0acd736ce42df9a8b5bf2ca8...

15:42:42
libcharon.so.0(+0x6c571) [0x7f174463f571]
32 Bytes to wipe : e8a9bc3728d2897478403c19a23326f9eee2310c0acd736ce42df9a8b5bf2ca8

----------repeats------------
15:43:12
plugins/libstrongswan-kernel-netlink.so(+0x13e7d) [0x7f173c750e7d]
4096 Bytes to wipe : ...e8a9bc3728d2897478403c19a23326f9eee2310c0acd736ce42df9a8b5bf2ca8...

15:43:12
plugins/libstrongswan-kernel-netlink.so(+0x38d9)[0x7f173c7408d9]
584 Bytes to wipe : ...e8a9bc3728d2897478403c19a23326f9eee2310c0acd736ce42df9a8b5bf2ca8...

15:43:42
plugins/libstrongswan-kernel-netlink.so(+0x13e7d)[0x7f173c750e7d]
4096 Bytes to wipe : ...e8a9bc3728d2897478403c19a23326f9eee2310c0acd736ce42df9a8b5bf2ca8...
```

```
15:43:42
plugins/libstrongswan-kernel-netlink.so(+0x38d9) [0x7f173c7408d9]
584 Bytes to wipe: ...e8a9bc3728d2897478403c19a23326f9eee2310c0acd736ce42df9a8b5bf2ca8...
---------repeats------------
...
```

Another key(6fd0e90e966c1efb67938be09a58a317d65c7cde2e16a469ad3067dcb69864f1, found 6 times in the core dump) has 82 wipes in 10 minutes (also chunks of 584 and 4096 bytes), but every 20 and every 30 seconds.
The debug file charon_wipe.txt is atached.
What are these periodic jobs?

Looks like a one more memwipe is still missing on some place...

Could you please point me to a place in a Linux kernel, where the keys are copied into a user space, when charon queries installed SAs using XFRM_MSG_GETSA? I will try to remove key copying, and test whether I can still found any keys in charon dump.

#8 - 04.08.2017 18:32 - Tobias Brunner

> What are these periodic jobs?

The 4096 bytes buffer is probably the receive buffer that's now wiped. The 584 bytes buffer the actual received message.

The only calls to query_sa() occur when child_sa_t::update_usebytes() is called, which happens if get_usestats() is called with bytes and packets arguments (there is also a fallback case, see below). There are four reasons for such a call: ipsec statusall, swanctl --list-sas, RADIUS accounting, and the log message when a CHILD_SA is deleted, which contains the transmitted bytes. The latter will happen e.g. after a rekeying. So depending on the rekeying times you might see some periodic wipes even if you don't do any status queries. But the keys there would obviously be different each time, so it doesn't seem to be the reason here.

However, there is another case where update_usebytes() is called, which is as a fallback if update_usetime() (via query_policy()) could not determine a use time for the CHILD_SA. The problem is that if you never used the SA the last use time will be reported as 0, which will then be interpreted as a failure causing a fallback call to update_usebytes() (which is actually useless on Linux as we can't use it to determine the last use time of a CHILD_SA). Therefore, DPDs and/or NAT keepalives, which query the last use time, will cause such a periodic call until the SA (or rather the respective policy) is used once. So try sending at least one packet in both directions to see if that makes a difference (or disable DPDs and NAT keepalives).

> Looks like a one more memwipe is still missing on some place...

No clue, unless it's somehow related to your version of OpenSSL (i.e. some internal memory there is not wiped even after the corresponding objects are destroyed).

> Could you please point me to a place in a Linux kernel, where the keys are copied into a user space, when charon queries installed SAs using XFRM_MSG_GETSA? I will try to remove key copying, and test if I can still found any key in charon.

That's in copy_to_user_state_extra() in net/xfrm/xfrm_user.c.

#9 - 09.08.2017 13:59 - Roman Moschenski

> No clue, unless it's somehow related to your version of OpenSSL

I have tried to compile Strongswan without OpenSSL. There was no difference, the keys are still in charon dump.

> So try sending at least one packet in both directions to see if that makes a difference.

Yes, that makes a difference, with a traffic in a tunnel, there are no keys appeared in charon.

My conclusion is the same as your's:

> Perhaps the key could also be in some socket buffer (Netlink/XFRM socket, over which we have no direct control)

as long as charon calls query_sa() (no trafic on the beginning, or "ipsec statusall") the key ghosts will appear in a charon process dump in some freed socket buffers.

Interesting, that you was not able to reproduce this issue. You can try to establish a connection and after that let "ipsec statusall" run 100 times and look if there are some key ghosts...

But for me this issue is closed, to eliminate all key ghosts in a RAM after a tunnel was closed, we have to wipe also a socket buffers, that is not a Strongswan jurisdiction...
Thanks a lot for your support.

#10 - 14.08.2017 11:42 - Tobias Brunner

> Interesting, that you was not able to reproduce this issue.

I didn't do any status queries in my tests.

> You can try to establish a connection and after that let "ipsec statusall" run 100 times and look if there are some key ghosts...

I tried, but in our testing environment (with the referenced patches in strongSwan applied) I don't see any keys in the memory dump.

> But for me this issue is closed, to eliminate all key ghosts in a RAM after a tunnel was closed, we have to wipe also a socket buffers, that is not a Strongswan jurisdiction...

OK, so you do that in the kernel?

#11 - 14.08.2017 11:55 - Roman Moschenski

> OK, so you do that in the kernel?

Yes, I have removed a copying of **xfrm_algo_auth** and **xfrm_algo** into user space, so that the key material never leaves the kernel space. Additionally I have added a memory wipe in **xfrm_state_gc_destroy** before kfree called. After that I was not able to find any key ghosts in a RAM anymore.

#12 - 14.08.2017 13:16 - Tobias Brunner

- Tracker changed from Issue to Bug

- Category set to kernel-interface

- Status changed from Feedback to Closed

- Assignee set to Tobias Brunner

- Target version set to 5.6.0

- Resolution set to Fixed

Files

charon_wipe.txt	4.62 MB	01.08.2017	Roman Moschenski
charon_wipe.zip	192 KB	04.08.2017	Roman Moschenski