lurch/OMEMO Security Assessment

Vault Labs, HardenedVault Limited

Table of Contents

Executive Summary

This report describes the results of the security assessment of lurch and its supportive libraries (axc and libomemo) made by gkdr from 2017.

Introduction of lurch

lurch is a pidgin plugin implementing a widespread revision of the XEP 0384 "OMEMO" encryption protocol.

OMEMO is based on the axolotl protocol. In lurch, axolotl is implemented in libsignal-protocol-c. axc is introduced to wrapper top-level functions and provides a sqlite-based implementation of the required key material storage of libsignal-protocol-c.

libomemo is introduced to compose and parse XML structures dedicated for OMEMO, and in the mean time perform encryption and decryption of the message body. An additional sqlite-based storage is introduced for device ids and group chat configures.

OMEMO is designed to minimize the intervention from its user, so lurch automatically handles XMPP pub-sub events for device lists and bundles to initialize session, with no users' intervention and notification, except for errors.

Assessment of axc

1) (fixed) axc_context_destroy_all() does not release the pointer passed to it.

The projects using axc (e.g. lurch) assume axc_context_destroy_all() will free the ctx passed to it, but axc_context_destroy_all() actually only cleans up its members, without freeing itself, which will cause severe memory leak.

2) (fixed) SIGNAL_UNREF() is used to release instances of session_cipher.

libsignal-protocol-c has three kind of complex types or "classes":
* reference-counting types derived from signal_type_base, which holds its reference counter, and a pointer to a function to release the instance (destroyer), with SIGNAL_REF() and SIGNAL_UNREF() to manipulate the reference counter. If the counter becomes zero within SIGNAL_UNREF(), the destroyer is called.

* context types, which often has pointers to the opaque signal_context and signal_protocol_store_context as its member.

* copyable types, which usually holds values easy to copy, so reference-counting is not needed.

session_cipher itself belongs to the context type, so its instances should not be released with the macro SIGNAL_UNREF(), otherwise, SIGNAL_UNREF() may treat its first member, which is a pointer, as the reference counter and decrement it, as this pointer is nearly impossible to be one, the instance of session_cipher passed to SIGNAL_UNREF() cannot be correctly released, causing a weird memory leak.

3) Identity key storage omits the corresponding device id.
As axolotl protocol suggests, different clients under the same account are distinguished by their device_id, so what should be uniquely stored and queried should be the tuple of (addr_p->name, addr_p->device_id, identity_key), not (addr_p->name, identity_key), but when storing an identity key, axc omits its corresponding device id, making the correspondance of an identity key and its device id being broken, and breaking the axolotl protocol, because it is impossible to tell whether a client changes its identity key while keeping its device id, in which case the changed key should be considered untrusted.

Assessment of libomemo

1) Currently there is no way to generate a valid KeyTransportMessage.

KeyTransportMessage is an omemo-encrypted message without elements for iv or payload, only with encrypted "key" blobs, but valid recipient JID and message id (sequence number of an XMPP message) should be set. However, there is no valid way to generate such message: A KeyTransportMessage could theoretically be generated by encrypting an ordinary empty XMPP message (without a body node but having valid recipient JID and message id), but in order to finally generate a valid encrypted omemo element, both member message_node_p and payload_node_p should be present in an omemo_message structure, otherwise libomemo will refuse to process by returning OMEMO_ERR_MALFORMED_XML. Simply allowing omemo_message_export_encrypted() to accept an omemo_message without

payload_node_p is not enough, because a valid "message" node is still required, but omemo_message_prepare_encryption() could only accept a plain XMPP message with a body (otherwise it will refuse to further process by returning OMEMO_ERR_MALFORMED_XML),

```
msg_node_p = mxmlLoadString((void *) 0, outgoing_message, MXML_OPAQUE_CALLBACK);
if (!msg_node_p) {
ret_val = OMEMO_ERR_MALFORMED_XML;
goto cleanup;
}
msg_p->message_node_p = msg_node_p;

body_node_p = mxmlFindPath(msg_node_p, BODY_NODE_NAME);
if (!body_node_p) {
ret_val = OMEMO_ERR_MALFORMED_XML;
goto cleanup;
}
```

and an omemo_message obtained from omemo_message_create() does not have a valid message_node_p, with no way to manually add one, thus cannot be further processed into a valid omemo-encrypted message.

2) device id should be stored in another place.
In order to implement the device list functionality of omemo protocol, libomemo introduces an additional sqlite-based storage for device id, but as stated above, device ids should be stored with corresponding identity keys via axc. A dedicated function could be added to query all device ids tied to a user name (here is JID), forming the needed device list.

3) Using free() in place of g_free()
libomemo uses base64 operations of glib, so these results are allocated via g_malloc() and should be released via g_free() instead of free().

Assessment of lurch

1) (fixed) various misuse of free() and g_free()
g_free() of glib and free() of libc is not always interchangeable, especially if glib is built with ENABLE_MEM_PROFILE or ENABLE_MEM_CHECK defined, as pointed out, so free() a g_malloc()-ed pointer MAY have grave result.

2) (mostly fixed) a lot of memory leak.
The return value of xmlnode_to_str() and xmlnode_get_data() should be released with g_free(), otherwise they are leaked. In fact, xmlnode does not store an serialized string form inside, and stores data as chunks, so both the serialized string form and intact data are assembled on the fly when needed, and need releasing after used.

3) There is no way to generate a valid KeyTransportMessage.
lurch_pep_bundle_for_keytransport() is introduced to handle some handshaking problem with KeyTransportMessages, but as stated above, there is no way to generate a valid KeyTransportMessage, so this function is actually malfunctional.

4) Memory leak during XML packet processing.
In the sending queue of libpurple, because they are sent via void jabber_send(JabberStream *js, xmlnode *packet) and released by the caller, one could set (*msg_stanza_pp) to NULL, or modify (**msg_stanza_pp) in place, but one should neither release (*msg_stanza_pp) (cause double free) nor point it to another xmlnode. (there will be nowhere to release this xmlnode, since after jabber_send() returns, the pointer value passed to its second argument packet

remain unchanged, no matter how packet is changed inside the context of jabber_send().)
In the receiving queue, on the other hand, they come from void jabber_process_packet(JabberStream *js, xmlnode **packet), in this case, one could modify (**msg_stanza_pp) in place, set (*msg_stanza_pp) to NULL, or point it to another xmlnode (thus the new xmlnode will be released by the caller of jabber_process_packet()), but if one is going to point (*msg_stanza_pp) to another place, they are responsible to release the original xmlnode in your callback first, otherwise it will be leaked.

Recommendations

axc

1) Store identity keys with their corresponding device ids and add a method to export a device list for a user name.

libomemo

1) Compose a way to generate valid KeyTransportMessages.
The easiest way is to make omemo_message_export_encrypted() work with an omemo_message without payload_node_p, and make omemo_message_prepare_encryption() work with a plain message without a body.

2) Let axc handle device id storage, as described above.

3) Check mistakenly uses of free(), never use free() on g_malloc()-ed pointers (e.g. results of base64 operations of glib), use g_free() instead.

lurch

1) Check mistakenly uses of free(), never use free() on g_malloc()-ed pointers, use g_free() instead.

2) Carefully study the source code of libpurple, and check whether an exported value returned as pointer needs releasing, and release it if needed.

3) How a message is passed between plugins of libpurple is not well documented, so its source code should be carefully studied, in order not to introduce memory leak.

Conclusion

These assessments are done during the work of lurch1317, which is based on lurch. Most fixes of detected problems are fed back to the upstream community, and some are accepted. The rejected fixes are included in our work.

Epilogue

We’re living in a world with full of wonder and danger. Your enemy might be somewhere lurking in the shadow. Know your enemy becomes more important than ever now. If this is just tip of iceberg, you should do your own math how much do you pay or willing to pay for security and privacy:

HOME BOUNTIES FAQ SUBMIT EVENTS CONTACT

Pidgin RCE

Status:	Active
Target:	Pidgin
Bounty:	Up to $100,000
Start Date:	1 June 2021
End Date:	31 August 2021

Pidgin RCE

We are looking for remote code execution exploits affecting the latest version of Pidgin on Windows and/or Linux. The exploit should work with default installations and should not require any user interaction other than reading a message.

Submit Now

Crypto is a neutral technology just like the natural existence (“For he makes his sun rise on the evil and on the good, and sends rain on the just and on the unjust.” -- Matt 5:45) but it’s the crucial part to protect your privacy. Crypto is hard and the crypto audit is harder. Hope this public report can help the FOSS (Free and Open source) crypto community take the design and implementation seriously. Last but not least, HardenedVault is providing audit service. Feel free to contact us via info@hardenedvault.net