Radically Open Security

Penetration Test Report

The Guardian Project

V 1.0
Diemen, April 30th, 2021
Confidential

Document Properties

Client	The Guardian Project
Title	Penetration Test Report
Targets	Smack-omemo/src/main Smack-omemo-signal/src/main
Version	1.0
Pentesters	Stefan Marsiske, Daan Spitz
Authors	Stefan Marsiske, Marcus Bointon, Patricia Piolon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	August 29th, 2018	Stefan Marsiske	Initial draft
0.2	September 10th, 2018	Marcus Bointon	Review
1.0	April 30th, 2021	Patricia Piolon	Review

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Overdiemerweg 28 1111 PP Diemen The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Radically Open Security B.V. is registered at the trade register of the Dutch chamber of commerce under number 60628081.

Table of Contents

1 Executive Summary

1.1 Introduction

Between August 5, 2018 and August 29, 2018, Radically Open Security B.V. carried out a code audit for The Guardian Project

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of work

The scope of the code audit was limited to the following targets:

- Smack-omemo/src/main
- Smack-omemo-signal/src/main

tagged in git with `v4.3`

1.3 Project objectives

The goal of the audit was to verify the correctness of the Omemo implementation and to check for any traditional vulnerabilities in the code.

1.4 Timeline

The Security Audit took place between August 5, 2018 and August 29, 2018.

1.5 Results In A Nutshell

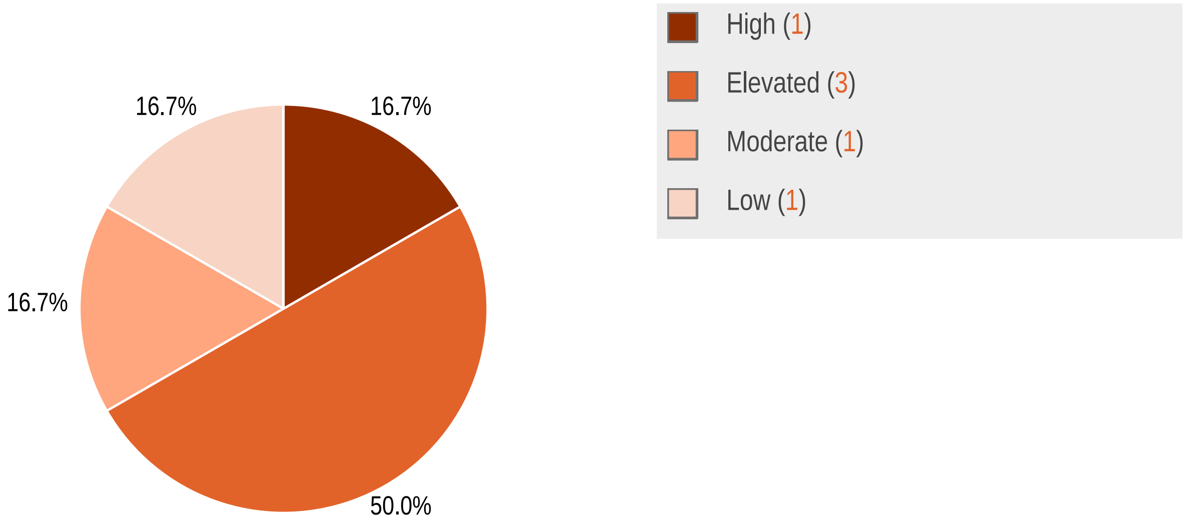
Three cases of compromise of forward security and one denial of service using resource exhaustion of prekeys has been found. The compromise of forward security is an especially serious matter as it reduces the security guarantees of the protocol significantly.

Furthermore, an issue was discovered in the validation of user-supplied usernames, possibly leading to unsafe file operations performed by the default file-backed omemo store. This could possibly lead to system compromise via arbitrary file reading and writing, depending on how the library is used in an application.

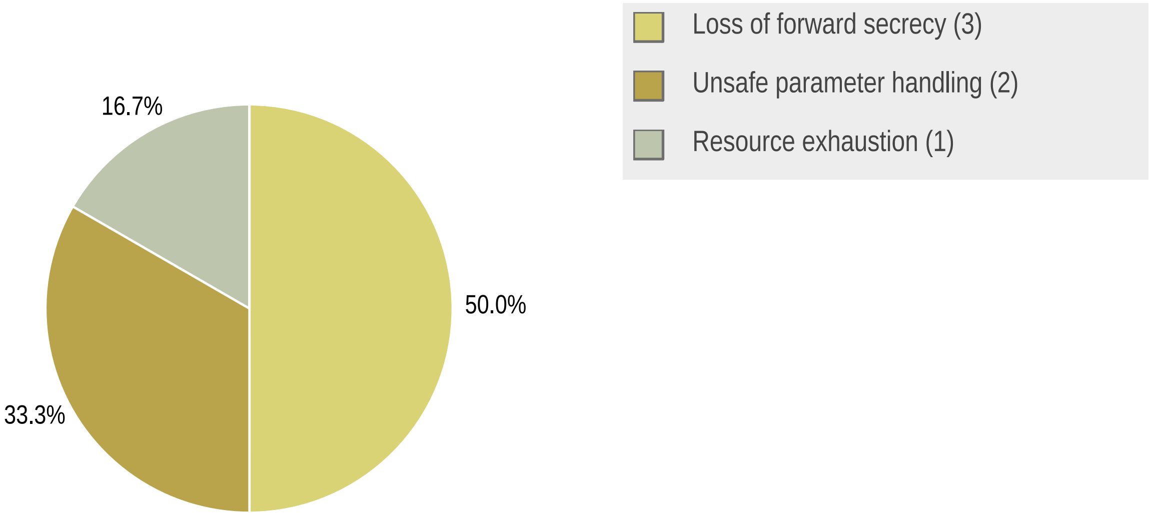
1.6 Summary of Findings

ID	Type	Description	Threat level
GSO-001	Loss of Forward Secrecy	Inactive devices which no longer come online retain old chaining keys, which, if compromised, break confidentiality of all messages from that key onwards.	High
GSO-002	Unsafe Parameter Handling	Inappropriate validation of user-supplied username and password inputs in the smack-omemo component may pass through characters that are unsafe to use in file operations performed by the default file-backed omemo store.	Elevated
GSO-003	Loss of Forward Secrecy	Read-only devices never update their keys and thus forfeit forward security.	Elevated
GSO-004	Loss of Forward Secrecy	Signed prekeys are not refreshed when used in a session initiation, allowing an active attacker to compromise the forward secrecy of a PreKeySignalMessage.	Elevated
GSO-005	Unsafe Parameter Handling	In the FileBasedOmemoStore, the arguments passed to the methods responsible for creating files/directories and reading/writing data to them are not sufficiently checked for malicious characters, leading to possible path traversal attacks.	Moderate
GSO-006	Resource Exhaustion	An attacker can exhaust all pre-keys using MAM stored messages, and prevent legitimate users from initiating a session with the peer.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
GSO-001	Loss of Forward Secrecy	Deactivate inactive devices after a certain amount of time. Allow inactive devices to reactivate only after a `keyTransportElement` ensures fresh keys.
GSO-002	Unsafe Parameter Handling	The functions used for reading and writing data to the omemo store should strip any special characters from parameters used for creating the target path inside the methods themselves. This way no assumptions are made about the safety of the parameters and incorrect implementations or future changes cannot introduce similar security issues. `if (slashIndex > 0) {` inside the `parseDomain` method should be changed to `if (slashIndex >= 0)` so that the first character cannot be a `/`.
GSO-003	Loss of Forward Secrecy	Read-only devices should send regular keyTransportElement messages.
GSO-004	Loss of Forward Secrecy	Rotate all used pre-keys upon usage.
GSO-005	Unsafe Parameter Handling	The functions used for reading and writing data to the omemo store should strip any special characters from parameters used for creating the target path inside the methods themselves. This way, no assumptions are made about the safety of the parameters and incorrect implementations or future changes cannot introduce similar security issues.
GSO-006	Resource Exhaustion	Retain all prekeys until all MAM messages are processed, and initiate a fresh session with the peer to avoid prekey reuse.

2 Methodology

2.1 Planning

Our general approach during this code audit was as follows:

1. **Code reading**
   The code has been inspected with rigorous attention to the implementation of the Omemo protocol, with special regard to our previous review of the Omemo protocol and its implementation in the 3rd party `Conversations` application.
2. **Code searching**
   The code was scanned for problematic use of classes and methods that are known to be potential causes of security issues. This includes methods that perform system commands, file or network operations, deserialization, class loading, sql queries and html outputting. Any such functionality was manually reviewed for problematic usage.

2.2 Risk Classification

Throughout the document, vulnerabilities or risks are labeled and categorized as:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.
- **High**
  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**
  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**
  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**
  Low risk of security controls being compromised with measurable negative impacts as a result.

Please note that this risk rating system was taken from the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting.

3 Pentest Technical Summary

3.1 Findings

We have identified the following issues:

3.1.1 GSO-001 — Inactive Devices Compromise Forward Secrecy

Vulnerability ID: GSO-001

Vulnerability type: Loss of Forward Secrecy

Threat level: High

Description:

Inactive devices which no longer come online retain old chaining keys, which, if compromised, break confidentiality of all messages from that key onwards.

Technical description:

Inactive devices that are no longer used and no longer come online should be pruned from the conversation. They keep a copy of an old chain key in their memory, which compromises the forward secrecy of the entire conversation.

Stale (inactive) devices are taken care of by "stale device" handling, however `processSendingMessage()` only disables our own devices, not the devices of others.

Impact:

Forward secrecy is forfeited. All messages derived from the old key can be decrypted if an inactive (and presumably old) device key is compromised.

Recommendation:

Deactivate inactive devices after a certain amount of time. Allow inactive devices to reactivate only after a `keyTransportElement` ensures fresh keys.

3.1.2 GSO-002 — Invalid BareJid validation leads to unauthorized file access

Vulnerability ID: GSO-002

Vulnerability type: Unsafe Parameter Handling

Threat level: Elevated

Description:

Inappropriate validation of user-supplied username and password inputs in the `smack-omemo` component may pass through characters that are unsafe to use in file operations performed by the default file-backed omemo store.

Technical description:

The following snippet `smack-omemo/src/main/java/org/jivesoftware/smackx/omemo/OmemoManager.java:180` contains code that is exploitable:

```
public static synchronized OmemoManager getInstanceFor(XMPPConnection connection) {
    BareJid user;
    if (connection.getUser() != null) {
        user = connection.getUser().asBareJid();
    } else {
        // This might be dangerous
        try {
            user = JidCreate.bareFrom(((AbstractXMPPConnection)
 connection).getConfiguration().getUsername());
        } catch (XmppStringprepException e) {
            throw new AssertionError("Username is not a valid Jid. " +
                    "Use OmemoManager.gerInstanceFor(Connection, deviceId) instead.");
        }
    }
```

If the `XMPPConnection` object is not yet authenticated and therefore does not contain a username, the username will be taken from the configuration. The line creating the jid object using `JidCreate.bareFrom` passes the username from the connection configuration. This can be any CharSequence coming from the outside client code utilizing the library (set by calling `ConnectionConfiguration.setUsernameAndPassword`).

Following the logic in the `bareFrom` method in the `jxmpp` module to the `parseLocalpart` method, we can see that if the username contains a `/` character before a `@` character, the jid will be seen as having only a domain part and will be instantiated as a `DomainPartJid` object.

The code that parses this domain part contains the actual bug:

```
public static String parseDomain(String jid) {
  if (jid == null) return null;

  int atIndex = jid.indexOf('@');
  int slashIndex = jid.indexOf('/');
  if (slashIndex > 0) {
    // 'local@domain.foo/resource' and 'local@domain.foo/res@otherres' case
    if (slashIndex > atIndex) {
      return jid.substring(atIndex + 1, slashIndex);
    // 'domain.foo/res@otherres' case
    } else {
      return jid.substring(0, slashIndex);
    }
  } else {
    return jid.substring(atIndex + 1);
  }
}
```

The `slashIndex > 0` check is supposed to check if there are any slashes in the username, but it skips the first character at index 0, which can therefore still contain a `/` character undetected. The code in the `else` branch then returns the part after the first `@` character. This means that if the first two characters are `/@`, any following characters will be returned and accepted into the domain part of the `DomainpartJid`.

So if the username is `/@../../../../../../../etc/passwd`, `../../../../../../../etc/passwd` will be stored as the jid. This jid is later used unchanged as part of the path for retrieving the right `defaultDeviceId` from the file-backed omemo store.

To demonstrate files being written outside of the omemo store root, the following PoC code was created and tested:

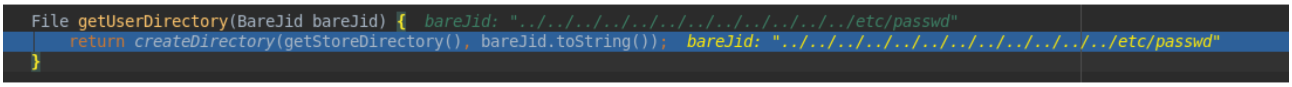
```
OmemoConfiguration.setFileBasedOmemoStoreDefaultPath(new File("/tmp/store"));
Builder<?, ?> builder = DummyConnectionNotAuthed.getDummyConfigurationBuilder();
builder.setUsernameAndPassword("/@../../../../../../../../../../../../../tmp/h4x", "");
DummyConnectionNotAuthed connection = new DummyConnectionNotAuthed(builder.build());

OmemoManager a = OmemoManager.getInstanceFor(connection);
```

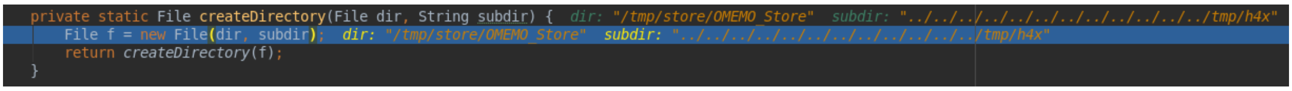
After running this PoC as a unit test, the following files can be observed outside of the omemo store root (`/tmp/store`):

```
[user@pentest]$ ls -l /tmp/h4x
total 4
-rw-rw-r-- 1 user user 4 Aug  9 07:49 defaultDeviceId
```

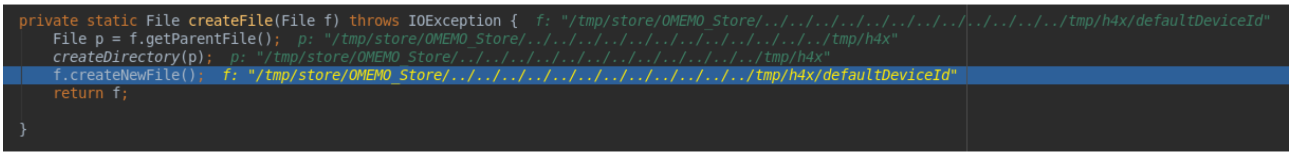
The following screenshots show a debugging session annotating the local variables that are being passed forward to the file operation methods:



```
File getUserDirectory(BareJid bareJid) {  bareJid: "../../../../../../../../../../../../etc/passwd"
    return createDirectory(getStoreDirectory(), bareJid.toString());  bareJid: "../../../../../../../../../../../../etc/passwd"
}
```



```
private static File createDirectory(File dir, String subdir) {  dir: "/tmp/store/OMEMO_Store"  subdir: "../../../../../../../../../../../../tmp/h4x"
    File f = new File(dir, subdir);  dir: "/tmp/store/OMEMO_Store"  subdir: "../../../../../../../../../../../../tmp/h4x"
    return createDirectory(f);
}
```



```
private static File createFile(File f) throws IOException {  f: "/tmp/store/OMEMO_Store/../../../../../../../../../../../../tmp/h4x/defaultDeviceId"
    File p = f.getParentFile();  p: "/tmp/store/OMEMO_Store/../../../../../../../../../../../../tmp/h4x"
    createDirectory(p);  p: "/tmp/store/OMEMO_Store/../../../../../../../../../../../../tmp/h4x"
    f.createNewFile();  f: "/tmp/store/OMEMO_Store/../../../../../../../../../../../../tmp/h4x/defaultDeviceId"
    return f;

}
```

Impact:

This bug could lead to different attack vectors depending on how exactly the library is integrated into an application. In any case, the result is that arbitrary folders can be created outside of the omemo store root, and files can be written to those directories. Further potential attacks could be implemented against the system combining other load and store operations.

Recommendation:

The functions used for reading and writing data to the omemo store should strip any special characters from parameters used for creating the target path inside the methods themselves. This way no assumptions are made about the safety of the parameters and incorrect implementations or future changes cannot introduce similar security issues.

`if (slashIndex > 0) {` inside the `parseDomain` method should be changed to `if (slashIndex >= 0)` so that the first character cannot be a `/`.

3.1.3 GSO-003 — Read-Only Devices Compromise Forward Secrecy

Vulnerability ID: GSO-003

Vulnerability type: Loss of Forward Secrecy

Threat level: Elevated

Description:

Read-only devices never update their keys and thus forfeit forward security.

Technical description:

Read-only devices will forward their Signal chaining key, but messages are never sent from these devices so the Signal root key will never be ratcheted forward. Such a device compromises the future secrecy of the entire conversation: if the receiving chaining key of such a device is compromised, the rest of the conversation from that point onwards is compromised.

Impact:

If a chaining key is compromised, the confidentiality of all future messages is compromised.

Recommendation:

Read-only devices should send regular keyTransportElement messages.

3.1.4 GSO-004 — Signed Pre-Keys Should Be Updated When Used

Vulnerability ID: GSO-004

Vulnerability type: Loss of Forward Secrecy

Threat level: Elevated

Description:

Signed prekeys are not refreshed when used in a session initiation, allowing an active attacker to compromise the forward secrecy of a `PreKeySignalMessage`.

Technical description:

From the previous protocol audit:

The lifetime of (signed) prekeys should be mentioned in the standard. Signed prekeys should be changed regularly in order to achieve forward secrecy. This should at least be done after every time the user receives a PreKeySignalMessage that uses the latest signed prekey, but it can be done more often (based on time) to ensure the forward secrecy of dropped messages.

Although signed pre-keys are regularly updated, they are not rotated when someone uses one.

```
changeSignedPreKey -- regenerates changed prekeys every 7-14 days
called from
     rotateSignedPreKey
     regenerate
     publishBundle
     removeOldSignedPreKeys -- keeps the last n (configurable via MaxNumberOfStoredSignedPreKeys)
 old prekeys
```

Impact:

Forward secrecy can be forfeited, especially in combination when no one-time pre-key is sent along.

Recommendation:

Rotate all used pre-keys upon usage.

3.1.5 GSO-005 — Unsafe parameters passed to file operation methods

Vulnerability ID: GSO-005

Vulnerability type: Unsafe Parameter Handling

Threat level: Moderate

Description:

In the `FileBasedOmemoStore`, the arguments passed to the methods responsible for creating files/directories and reading/writing data to them are not sufficiently checked for malicious characters, leading to possible path traversal attacks.

Technical description:

The methods affected include the following:

- `writeBytes`
- `readBytes`
- `writeIntegers`
- `readIntegers`
- `writeInt`
- `readInt`
- `writeLong`
- `readLong`
- all the methods of the `fileHierarchy` class

An example of this vulnerability can be found in `OmemoManager.java -> deviceListUpdateListener -> getOmemoStoreBackend().mergeCachedDeviceList() -> storeCachedDeviceList()`. Another `BareJid` contact parameter is used to get the file to write to, and the data to write is taken from status updates of active devices in the `deviceListUpdateListener`.

Impact:

This bug could potentially allow an attacker to write arbitrary data to arbitrary files, leading to serious security issues. Potential further attacks could be implemented against the system combining other load and store operations, depending on the exact implementation of the application.

Recommendation:

The functions used for reading and writing data to the omemo store should strip any special characters from parameters used for creating the target path inside the methods themselves. This way, no assumptions are made about the safety of the parameters and incorrect implementations or future changes cannot introduce similar security issues.

3.1.6 GSO-006 — Exhaustion of Pre-Keys Using MAM Messages.

Vulnerability ID: GSO-006

Vulnerability type: Resource Exhaustion

Threat level: Low

Description:

An attacker can exhaust all pre-keys using MAM stored messages, and prevent legitimate users from initiating a session with the peer.

Technical description:

`decryptMamQueryResult()` does not take care of `PreKeySignalMessage` by retaining them until all MAM messages are processed, also no attempt is made to initiate a fresh session back to the initiator.

Impact:

Legitimate users might be prevented from initializing a session with a victim.

Recommendation:

Retain all prekeys until all MAM messages are processed, and initiate a fresh session with the peer to avoid prekey reuse.

3.2 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

3.2.1 NF-001 — Search the code for dangerous java methods

The smack-omemo and smack-omemo-signal code was searched for potentially dangerous method calls that can lead to security issues when passed user-controlled input. No such usage was found.

The following methods were included in this search:

- `java.lang.ClassLoader.defineClass`
- `java.net.URLClassLoader`
- `java.beans.Instrospector.getBeanInfo`
- `java.io.File.delete`
- `java.io.File.renameTo`
- `java.io.File.listFiles`
- `java.io.File.list`
- `java.io.FileReader`
- `java.io.FileWriter`
- `java.io.RandomAccessFile`
- `System.setProperty`
- `System.getProperties`
- `System.getProperty`
- `System.load`
- `System.loadLibrary`
- `Runtime.exec`
- `ProcessBuilder (constructor)`
- `java.awt.Robot.keyPress/keyRelease`
- `java.awt.Robot.mouseMove/mousePress/mouseRelease`
- `java.lang.Class.getDeclaredMethod`
- `java.lang.Class.getDeclaredField`
- `java.lang.reflection.Method.invoke`
- `java.lang.reflection.Field.set`
- `java.lang.reflection.Field.get`
- `javax.script.ScriptEngine.eval`
- `Runtime.getRuntime().exec`
- `XMLdecoder`

- `XStream`
- `fromXML`
- `ObjectInputSteam`
- `readObject`
- `readObjectNodData`
- `readResolve`
- `readExternal`
- `ObjectInputStream.readUnshared`

3.2.2 NF-002 — Investigate serialization functionality of the CachedDeviceList class

Due to time-constraints, this functionality was not thoroughly checked for security issues. When combined with the potential problems in the file based omemo store read/write operations as described in the unsafe file operations (page 15) section, this could lead to dangerous scenarios in which user-controlled input is deserialized.

Nonetheless, if those file operations are properly secured this will likely not pose a security threat. Further retesting of implemented security fixes will have to verify this, as described in the future work (page 20) section.

The following line contains code that could warrant further inspection:

- `java/org/jivesoftware/smackx/omemo/internal/CachedDeviceList.java:35: public class CachedDeviceList implements Serializable {`

3.2.3 NF-003 — Investigate potential xml attacks against used parsers

Due to time-constraints and the complexity involved in verifying some of the other issues encountered, no time was spent investigating the XML parsing components of the library for attacks such as external entity injection and denial of service.

During a potential retest to verify any security fixes implemented, some time should be dedicated to investigating these potential issues. See the future work (page 20) section.

The following files contain code that could warrant further inspection:

- `smack-omemo-main/java/org/jivesoftware/smackx/omemo/element/OmemoElement.java`
- `smack-omemo-main/java/org/jivesoftware/smackx/omemo/element/OmemoBundleElement.java`

- `smack-omemo-main/java/org/jivesoftware/smackx/omemo/element/OmemoVAxolotlElement.java`
- `smack-omemo-main/java/org/jivesoftware/smackx/omemo/element/OmemoBundleVAxolotlElement.java`
- `smack-omemo-main/java/org/jivesoftware/smackx/omemo/element/OmemoDeviceListElement.java`
- `smack-omemo-main/java/org/jivesoftware/smackx/omemo/provider/OmemoVAxolotlProvider.java`
- `smack-omemo-main/java/org/jivesoftware/smackx/omemo/provider/OmemoBundleVAxolotlProvider.java`
- `smack-omemo-main/java/org/jivesoftware/smackx/omemo/provider/OmemoDeviceListVAxolotlProvider.java`
- `smack-omemo-main/java/org/jivesoftware/smackx/omemo/OmemoInitializer.java`

4 Future Work

- **Retest of implemented security fixes**

  If deemed appropriate, a retest of security fixes implemented in response to the findings described in this document can verify that all attack vectors have been closed off.
- **Further investigation of Serialization**

  Further investigation of the serialization functionality as used in the Smack library could potentially uncover additional security issues.
- **Further investigation of XML parsing**

  Further investigation of the XML parsers used in the library could potentially uncover additional security issues.
- **Address shortcomings of the Omemo protocol**

  The Omemo protocol could do with a review, but most importantly, cryptographic solutions should be explored for securing sessions against infiltrations by adding surreptitious devices to peers who blindly accept new devices.
- **Consider checking the security of key material**

  Investigate how the compromise of a device affects the confidentiality of session keys.

5 Conclusion

The cryptographic issues found are mostly related to the quality of the Omemo protocol specification and not the concrete implementation. Some things can be fixed at the implementation level, but we suggest updating the Omemo specification to help implementers avoid common pitfalls.

The issue relating to unsafe file operations could in certain specific cases be used to compromise parts of the system running the library code. This can be trivially fixed by adding effective validations, as suggested in the recommendations.

Appendix 1 Testing team

Stefan Marsiske	Stefan runs workshops on radare2, embedded hardware, lock-picking, soldering, gnuradio/SDR, reverse-engineering, and crypto topics. In 2015 he scored in the top 10 of the Conference on Cryptographic Hardware and Embedded Systems Challenge. He has run training courses on OPSEC for journalists and NGOs.
Daan Spitz	Daan is a security researcher and developer who believes in offensive security. He has a special interest in the fields of reverse engineering, system emulation/fuzzing, vulnerability discovery and exploit development and has several years of experience in performing infrastructure and application-level security audits. He occasionally plays CTF with the Eindbazen team, which he enjoys a lot, and has been helping to organize and build the CTF event for the Hack in the Box security conference in Amsterdam since 2016.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.

Front page image by Slava (https://secure.flickr.com/photos/slava/496607907/), "Mango HaX0ring",
Image styling by Patricia Piolon, https://creativecommons.org/licenses/by-sa/2.0/legalcode.