Penetration Test Report

Conversations App for Android

V 1.0
Amsterdam, April 9th, 2025
Public

Document Properties

Client	Daniel Gultsch
Title	Penetration Test Report
Target	Conversations app for Android
Version	1.0
Pentester	András Veres-Szentkirályi
Authors	András Veres-Szentkirályi, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	April 7th, 2025	András Veres-Szentkirályi	Initial draft
0.2	April 9th, 2025	Marcus Bointon	Review
1.0	April 9th, 2025	Marcus Bointon	1.0

Contact

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

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam 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 March 10, 2025 and April 7, 2025, Radically Open Security B.V. carried out a penetration test for Daniel Gultsch.

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 penetration test was limited to the following target:

- Conversations app for Android

The scoped services are broken down as follows:

- Scoping and preparation: 1 days
- Mapping the codebase, SBOM analysis: 1 days
- Cryptography analysis including TLS, SCRAM and SSDP: 1 days
- Android-specific analysis including backups, SQLite and Janus: 1 days
- Reporting: 1 days
- **Total effort: 5 days**

1.3 Project objectives

ROS will perform a penetration test with Daniel Gultsch in order to assess the security of the *Conversations* app for Android. To do so ROS will inspect the attack surface, focusing on network communication, cryptography, and IPC, and guide Daniel Gultsch in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The security audit took place between March 10, 2025 and April 7, 2025.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 Moderate and 1 Low-severity issues.

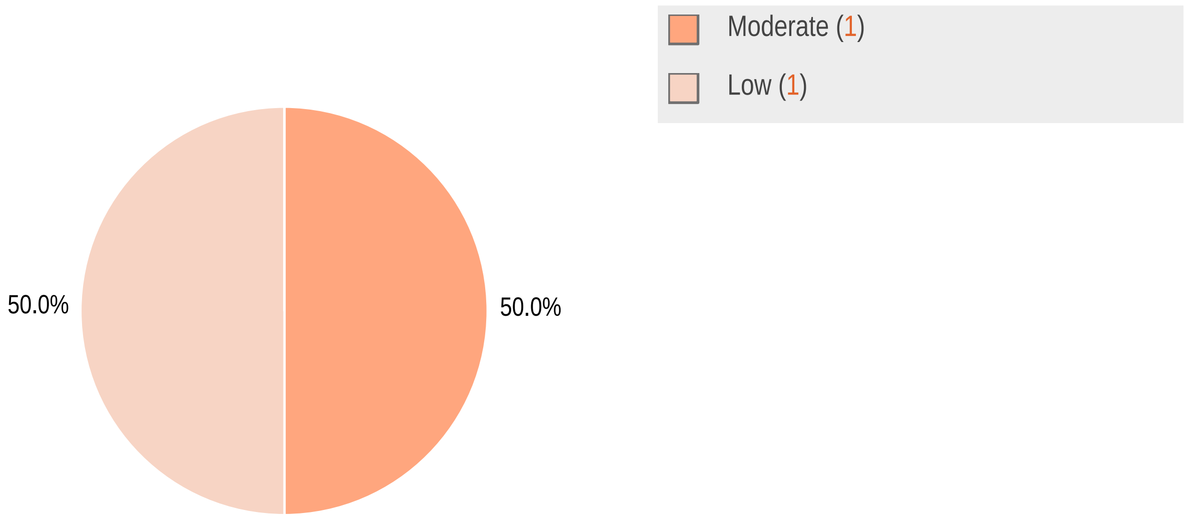
Both issues are related to the validation of certificates used to secure TLS connections. Wildcard certificates could be used to impersonate servers in certain rare cases CLN-005 (page 10), while revoked certificates and those issued without corresponding certificate transparency log entries are accepted by the app CLN-006 (page 11).

By exploiting these issues, an attacker might be able to perform a MITM (machine-in-the-middle) attack against targeted clients in highly specific scenarios.

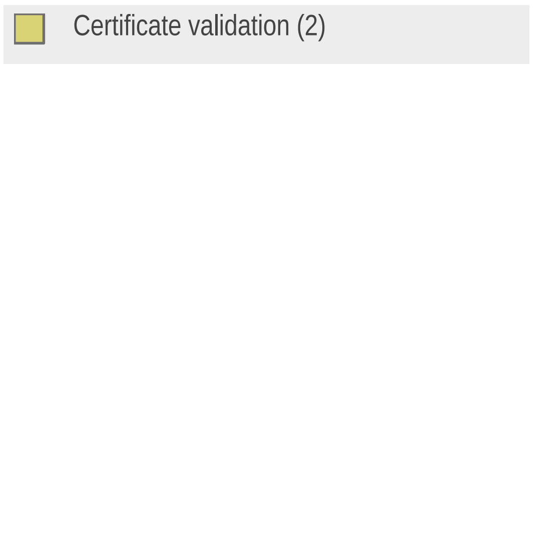
1.6 Summary of Findings

ID	Type	Description	Threat level
CLN-005	Certificate Validation	The custom-built hostname matcher implemented in the class XmppDomainVerifier accepts certain hostnames as valid for wildcard certificates that would be rejected by web browsers and other standards-compliant TLS clients.	Moderate
CLN-006	Certificate Validation	The TLS layer of the XMPP client mostly inherits Java's default configuration, which is a good baseline but misses some opportunities for hardening.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
CLN-005	Certificate Validation	Change the hostname matcher to implement stricter validation that matches the standards specified in the technical description.
CLN-006	Certificate Validation	Check OCSP and/or CRL, along with the presence of SCT to further harden the certificate validator.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

1. **Reconnaissance**
   We attempt to gather as much information as possible about the target. Reconnaissance can take two forms: active and passive. A passive attack is always the best starting point as this would normally defeat intrusion detection systems and other forms of protection afforded to the app or network. This usually involves trying to discover publicly available information by visiting websites, newsgroups, etc. An active form would be more intrusive, could possibly show up in audit logs and might take the form of a social engineering type of attack.
2. **Enumeration**
   We use various fingerprinting tools to determine what hosts are visible on the target network and, more importantly, try to ascertain what services and operating systems they are running. Visible services are researched further to tailor subsequent tests to match.
3. **Scanning**
   Vulnerability scanners are used to scan all discovered hosts for known vulnerabilities or weaknesses. The results are analyzed to determine if there are any vulnerabilities that could be exploited to gain access or enhance privileges to target hosts.
4. **Obtaining Access**
   We use the results of the scans to assist in attempting to obtain access to target systems and services, or to escalate privileges where access has been obtained (either legitimately though provided credentials, or via vulnerabilities). This may be done surreptitiously (for example to try to evade intrusion detection systems or rate limits) or by more aggressive brute-force methods. This step also consist of manually testing the application against the latest (2021) list of OWASP Top 10 risks. The discovered vulnerabilities from scanning and manual testing are moreover used to further elevate access on the application.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **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.

3 Reconnaissance and Fingerprinting

We were able to gain information about the software and infrastructure through the following automated scans. Any relevant scan output will be referred to in the findings.

- MobSF – https://mobsf.live/about
- SemGrep – https://semgrep.dev
- BadSSL – https://badssl.com

4 Findings

We have identified the following issues:

4.1 CLN-005 — Wildcard certificate hostname confusion

Vulnerability ID: CLN-005

Vulnerability type: Certificate Validation

Threat level: Moderate

Description:

The custom-built hostname matcher implemented in the class `XmppDomainVerifier` accepts certain hostnames as valid for wildcard certificates that would be rejected by web browsers and other standards-compliant TLS clients.

Technical description:

As part of the TLS peer verification, the client inspects the server certificate and — amongst other things — it checks whether the domain name(s) within the X.509 structure match the hostname of the server. XMPP treats this in a different way in contrast to most TLS connections; the JID hostname is matched instead of the one used for establishing the underlying TCP connection. *Conversations* implemented a custom matcher in the class `XmppDomainVerifier`, solving the hostname validation in method `matchDomain`. The algorithm within said method treated asterisk (`"*"`) symbols used as wildcards in a way that would also match separators (`"."`) between components of the hostname.

This goes against section 3.1 of RFC 2818:

> Names may contain the wildcard character `*` which is considered to match any **single domain name component** or component fragment. E.g., `*.a.com` matches `foo.a.com` but not `bar.foo.a.com`. `f*.com` matches `foo.com` but not `bar.com`.

…and also section 6.4.3 of RFC 6125:

> If the wildcard character is the only character of the left-most label in the presented identifier, the client SHOULD NOT compare against anything but the left-most label of the reference identifier (e.g., `*.example.com` would match `foo.example.com` but not `bar.foo.example.com` or `example.com`).

Impact:

An attacker

- able to actively manipulate the network traffic between a victim client using *Conversations* and the XMPP server `victimserver.foo.bar.tld` **and**
- in possession of a wildcard certificate valid for `*.bar.tld`

could perform a MITM (machine-in-the-middle) attack where the victim client would accept said certificate as a valid one for `victimserver.foo.bar.tld`, making it possible for the attacker to:

- capture authentication credentials (although impact is lowered if SCRAM is used), and
- impersonate the victim user and/or read their communication (although impact is lowered if E2EE is used).

Since *Conversations* has built-in support for Tor, this can also affect the impact if used:

- If a hidden service (`.onion` address) is used, the attacker model gets much stronger, as a successful MITM attack for such a scenario would also require access to the hidden service private key.
- If the XMPP server itself is a Tor node, the attacker model gets stronger in a similar way to the previous scenario, with the hidden service's private key being replaced with the relay identity's private key.
- If the XMPP server used has no direct connectivity to the Tor network, the opportunity for MITM attacks shifts from the network path between the victim user and the XMPP server to that between the Tor exit node and the XMPP server. This scenario also applies to proxy-like VPN services.

Recommendation:

Change the hostname matcher to implement stricter validation that matches the standards specified in the technical description.

4.2 CLN-006 — Missing TLS client hardening measures

Vulnerability ID: CLN-006

Vulnerability type: Certificate Validation

Threat level: Low

Description:

The TLS layer of the XMPP client mostly inherits Java's default configuration, which is a good baseline but misses some opportunities for hardening.

Technical description:

The configuration of the TLS layer used for XMPP communication is left at Java defaults, which can vary between versions and vendors. Based on our experience with the latest stock Google Android 15 on a Pixel 7 during testing, several of the tests marked as *bad* on `badssl.com` failed (the TLS handshake went through without exceptions), two of which have security relevance in the context of XMPP (besides CLN-005 (page 10)):

- `revoked.badssl.com` – the revocation status was not checked
- `no-sct.badssl.com` – the presence of Signed Certificate Timestamp (SCT) was not checked

Impact:

Since these are hardening measures, exploitation is limited to highly specific scenarios with motivated and sophisticated attackers. Impact is further lowered by the fact that TLS is just one of the layers of protection in transit, as SCRAM, Tor hidden services, and E2EE affect this as well, just like with CLN-005 (page 10).

In case of certificate revocation, attackers compromise the XMPP server and steal the private key that matches the certificate. If the XMPP server operator discovers this, they can ask the CA (Certificate Authority) to revoke the certificate but as the *Conversations* client will not check revocation status (CRL, OCSP), it will accept the certificate as valid up until its original intended expiration date.

In case of SCT, attackers strong-arm or bribe a CA to sign a fraudulent certificate with the XMPP server's FQDN but the attacker's public key. If this gets into the Certificate Transparency logs, this can get noticed quickly. If not, the lack of SCT can be a sign that the certificate was not submitted to the CT system.

Note that OCSP is essentially deprecated and is scheduled to be phased out in favour of CRLs and certificates with shorter lifetimes.

Recommendation:

Check OCSP and/or CRL, along with the presence of SCT to further harden the certificate validator.

5 Non-Findings

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

5.1 NF-009 — SQLite database security analysis

The application writes most of its data into an SQLite database called `history` which includes:

- accounts (JID, password),
- conversations,
- messages (plain text, even for E2EE sessions),
- OMEMO keys.

As mentioned in non-finding NF-008 (page 14), the built-in backup functionality of the Android system is instructed to require client-side encryption, which is especially important since the XML resources that affect the backup process explicitly include the directory `database` (with `history` in it).

Interestingly, the application also provides a way for users to do one-off or recurring backups on the shared storage of Android (`/sdcard`), which contains this same SQLite database – with an additional layer of gzip compression and encryption. This encryption uses 128-bit AES in Galois Counter Mode (GCM), resulting in a modern stream cipher with built-in integrity protection. The IV is included in the file, while the key is derived from the account password using PBKDF2 using 1024 iterations of HMAC with SHA-1. This also requires a salt, which is included in the file header as well – both this and the IV is generated using an appropriate `SecureRandom` CSPRNG.

While neither of the above approaches constitute vulnerabilities, we have some key observations:

- Using the account password for password-based encryption has certain advantages – however, the strength and long-term availability of the resulting key depends on a number of circumstances.
    - PBKDF2 has the advantage of being widely implemented, and although SHA-1 has its share of weaknesses, in this case, their combined weakness comes from the fact that the process can be massively parallelized on general-purpose GPUs. This could be improved by using Key Derivation Functions specifically designed with such hardware in mind, requiring high RAM usage as well, thwarting parallelization efforts – Argon2 is an optimal choice.
    - The account password might have been generated by a computer (e.g., the built-in registration helper generates 12-character ones from a set of 65 choices, resulting in over 72 bits of entropy), but also by a human for pre-existing accounts (which could result in much less entropy). Another potential issue with this approach is that the user might forget this password, and industry-standard XMPP servers probably have no access to the clear-text value – even the "forgotten password" function only allows the user to set a new one, but not to access the current one. This could be improved by adding an option for setting/changing the backup password and/or making use of a set of "trusted contacts" and a solution like Shamir's secret sharing (SSS).

- Backups made by Android's built-in backup facility have become more secure over the years. In apps that target Android 12 (API level 31) or higher, when a user runs the `adb backup` command, app data is excluded from any other system data that is exported from the device. The attacker model that could read the database is even more constrained by the fact that for cloud backups in Google Drive, backup data can't be read by the user or other apps on the device.
- The core developer of *Conversations* stated that device security is in the hands of the operating system, with the OS being responsible for FDE and only allowing backups after authentication, etc. This is a reasonable stance given sufficiently up-to-date devices and software environment, however — as discussed in non-finding NF-007 (page 15) — the app can be installed on devices running operating systems as old as Android 6.0, which do not have these security enhancements. This could be improved through disabling backups done by Android itself (`android:allowBackup="false"`) and/or encrypting the database with some device-specific value. These come at a cost of availability and (re)implementing functionality offered by the OS.

5.2 NF-008 — Android hardening

The APK is signed using its SHA-256 digest, which can be considered secure for this purpose. Depending on a simple and clearly labeled switch in the user preferences activity, `FLAG_SECURE` is enabled for all windows, making it harder for screen capturing malware to gain access to sensitive information displayed on the app screens. The built-in backup functionality of the Android system is instructed to require client-side encryption.

The resource `res/xml/network_security_config.xml` opts out of the default Android security policy of disallowing cleartext traffic, however, this is required for things like STARTTLS to work. The same resource also includes both system- and user-supplied certificates as trust anchors – however, for XMPP connections, system CAs can also be disabled with another clearly labeled switch in the user preferences activity.

Although not strictly an Android-specific hardening, this section is the most relevant for the discussion of binary-level hardening measures regarding native shared objects in the `lib` subdirectory within the APK. Two such libraries were included, `libconscrypt_jni` and `libjingle_peerconnection_so`, and all well-known hardening measures against memory corruption attacks are enabled:

- NX bit – non-executable memory pages, where possible
- stack canaries – verifies the presence and validity of a secret value on the stack to protect the return address against buffer overflows
- full RELRO – ensures that the GOT (`.got` and `.got.plt` segments) cannot be overwritten by marking it read-only
- fortify – common *libc* (standard C library) functions (in case of these two libraries: `memmove`, `strchr`, `memset`, `memcpy`, `vsnprintf`, `read`, `strlen`, `FD_SET`, `FD_ISSET`, `FD_CLR`) are replaced with fortified equivalents that perform additional bounds checks

5.3 NF-007 — Janus vulnerability

The *Conversations* app defines the minimum SDK level as 23 (Android 6.0), and Android versions below API level 27 (Android 8.1) are susceptible to attacks where the APK file is modified in a way that the system still accepts the package as if its signature was still valid. This is called the Janus vulnerability, and can be found easily using automated tools which tout such vulnerabilities, giving them more weight than their severity deserves.

Since the application uses both v1 and v2 APK signature schemes, the Janus vulnerability cannot be exploited on Android 8 devices, and this way, the app has done everything it can in this regard. The only way to avoid Android 5-7 users being vulnerable to Janus exploits would be to raise the minSDK level and/or remove v1 signatures, which would simply make it impossible for such users to install the app in the first place. Ultimately, it is a decision left to the app developers knowing their audience and making the decision regarding this tradeoff.

5.4 NF-004 — Analysis of cryptographic primitives with known vulnerabilities

XEP-0084: User Avatars

XEP-0084 standardizes user avatars in XMPP, and in *Table 1* within section 4.2.1 it explicitly states that SHA-1 should be used to produce a message digest of the avatar image. In section 8 titled *Security Considerations* it states that

> It is possible that output of the SHA-1 hashing algorithm can result in collisions; however, the use of SHA-1 in producing a hash of the avatar data is not security-critical.

Based on this, usage of SHA-1 within the codebase for the purposes of avatar handling is acceptable and bears no risk.

Affected source file(s):

- `src/main/java/eu/siacs/conversations/xmpp/pep/Avatar.java`
- `src/main/java/eu/siacs/conversations/entities/Contact.java` (line 99)
- `src/main/java/eu/siacs/conversations/generator/IqGenerator.java`
- `src/main/java/eu/siacs/conversations/persistance/FileBackend.java`
- `src/main/java/eu/siacs/conversations/services/XmppConnectionService.java`

XEP-0115: Entity Capabilities

XEP-0115 defines an XMPP protocol extension for broadcasting and dynamically discovering client, device, or generic entity capabilities. Section 9 titled *Security Considerations* states in its first subsection 9.1 that

> The SHA-1 hashing algorithm is mandatory to implement. All implementations MUST support SHA-1.

The rest of this subsection mentions that this might change in the future as preimage attacks (discussed in more length in the next subsection of the XEP) become *practically* vulnerable.

Based on this, use of SHA-1 within the codebase for the purposes of capabilities discovery is acceptable and bears no *practical* risk as of this assessment.

Affected source files:

- `src/main/java/eu/siacs/conversations/entities/ServiceDiscoveryResult.java`
- `src/main/java/eu/siacs/conversations/generator/AbstractGenerator.java`

SCRAM

The SCRAM implementation in *Conversations* supports the mechanism `SCRAM-SHA-1`, thus the classes that implement it must use SHA-1 as well. However, since

- 256- and 512-bit SHA-2 mechanisms are supported as well,
- these SHA-2 mechanisms have a higher priority, which is persisted to prevent later downgrades, and
- XEP 0474 prevents undetected downgrade attacks in case of a MITM attack,

the presence of the SHA-1 mechanism carries no intrinsic risk.

Affected source files:

- `src/main/java/eu/siacs/conversations/crypto/sasl/ScramSha1.java`
- `src/main/java/eu/siacs/conversations/crypto/sasl/ScramSha1Plus.java`

XEP-0392: Consistent Color Generation

XEP-0392 standardizes algorithms to consistently generate colors for a consistent user experience across platforms, and it explicitly states that SHA-1 must be used and

> From a security point of view, the exact choice of hash function does not matter here, since it is truncated to 16 bits. At this length, any cryptographic hash function is weak.

Affected source file:

- `src/main/java/eu/siacs/conversations/utils/XEP0392Helper.java`

XEP-0065: SOCKS5 Bytestreams

XEP-0065 standardizes establishing an out-of-band bytestream between any two XMPP users, and it explicitly states that SHA-1 must be used and

> The use of the SHA-1 algorithm to hash the SID, Requester's JID, and Target's JID is not security-critical. Therefore, the known weaknesses of SHA-1 are not of significant concern in this protocol.

Affected source file:

- `src/main/java/eu/siacs/conversations/xmpp/jingle/transports/SocksByteStreamsTransport.java`

XEP-0234: Jingle File Transfer

XEP-0234 standardizes transferring a file from one entity to another, including exchange of information about the file to be transferred. Although the SHA-1 hash of the transferred file is calculated and sent along to the receiving party,

- the SHA-256 hash is also sent in both code paths, and
- no SHA-1 processing is applied to incoming files.

This is in line with the XEP that states

> It is RECOMMENDED for implementations to use the strongest hashing algorithm available to both parties.

Affected source file:

- `src/main/java/eu/siacs/conversations/xmpp/jingle/JingleFileTransferConnection.java`

Unused code

While most of the examples cited above use deprecated cryptographic primitives for a task where their known vulnerabilities have no impact, investigation regarding said impact in some cases revealed that they had no real use within the project codebase anyway.

- `src/main/java/eu/siacs/conversations/entities/DownloadableFile.java` – the generated SHA-1 hash is not used, code will be removed
- `src/main/java/eu/siacs/conversations/utils/CryptoHelper.java`
  - `extractCertificateInformation` stores an SHA-1 hash in `Bundle`, but both callers of the function ignore this value
  - `get...Fingerprint...()` functions use SHA-1, but only for non-security functions

- `src/main/java/im/conversations/android/xmpp/EntityCapabilities.java` – `hash()` is not called within the project
- `src/main/java/im/conversations/android/xmpp/EntityCapabilities2.java` – the only place `hash()` is ever called within the project is right above, hardcoded as SHA-256
- `src/quicksy/java/eu/siacs/conversations/entities/Entry.java` – the only place `statusQuo(List<Entry>)` is ever called within the project is in the other `statusQuo(...)` right above, which in turn is unused.

5.5 NF-003 — Analysis of SCRAM implementation

Conversations implements SCRAM (RFC 5802), so we analyzed the relevant parts of the codebase to identify any vulnerabilities that could enable server impersonation attacks (via MITM), including downgrade attacks and password extraction. The main focus was on enumerating elements containing keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" and interpreting their meaning from RFC 2119 to create a benchmark to measure *Conversations* against. Of course, many of these refer to server-side obligations, which we ignored.

For SCRAM, the first relevant keyword is in section 2.2 *Notation*. According to the RFC, implementations **must** either implement `SASLprep` or reject non-US-ASCII Unicode codepoints in the `str` parameter of `Normalize`. Method `saslPrep` on line 122 of `CryptoHelper.java` implemented this in accordance with section 4 of RFC 3454. Escaping is performed by method `saslEscape` on lines 103-118 of the same file, and it correctly uses a single-pass approach, avoiding even the possibility of mixing up the correct order of a multi-pass replacer.

Section 5 *SCRAM Authentication Exchange* states that the client authenticates the server by computing the `ServerSignature` and comparing it to the value sent by the server. If the two are different, the client **must** consider the authentication exchange unsuccessful. *Conversations* does so in line 293 of `ScramMechanism.java`.

Subsection 5.1 *SCRAM Attributes* has a number of the target keywords, one for almost every attribute:

- `n`: client **must** include it in its first message to the server – *Conversations* does so on lines 75 and 76 of `ScramMechanism.java`.
- `m`: client **must** cause an authentication failure upon its presence – *Conversations* does so on line 172 of `ScramMechanism.java`.
- `r`: client **must** verify that the initial part of the nonce used in subsequent messages is the same as the nonce it initially specified – *Conversations* does so on line 194 of `ScramMechanism.java`.
- `c`: attribute is **required** -- *Conversations* includes it in the second authentication message (`clientFinalMessageWithoutProof`).
- `i`: **must** be sent by the server along with the user's salt – *Conversation* errors out either because it's missing or it being below zero on line 191 of `ScramMechanism.java`.
- As-yet unspecified mandatory and optional extensions -- see `m` above.

Section 6 *Channel binding* also lists several requirements in its bullet points:

- If the client supports channel binding and the server does not appear to, then the client **must not** use an `"n" gs2-cbind-flag` – Conversations implements this on line 62 of `ScramMechanism.java`.
- Clients that support mechanism negotiation and channel binding **must** use a `"p" gs2-cbind-flag` when the server offers the PLUS-variant – *Conversations* does so on lines 66-69 of `ScramMechanism.java`.
- If the client does not support channel binding, then it **must** use an `"n" gs2-cbind-flag` – *Conversations* implements this on line 62 of `ScramMechanism.java`.

Section 7 *Formal Syntax* states that all extensions are optional, i.e., unrecognized attributes not defined in the RFC **must** be ignored – *Conversations* does so by parsing attributes into `ImmutableMap` instances. Two definitions later, it also states that `cbind-data` **must** be present for `gs2-cbind-flag` of `"p"` and **must** be absent for `"y"` or `"n"` – *Conversations* implements this in lines 23 through 64 of `ChannelBindingMechanism.java`.

5.6 NF-002 — Analysis of SSDP implementation

We analyzed the XEP 0474 implementation to identify any vulnerabilities that could enable server impersonation attacks (via MITM), including downgrade attacks and password extraction. The main focus was on enumerating elements containing keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" and interpreting their meaning from RFC 2119 to create a benchmark to measure *Conversations* against. Of course, many of these refer to server-side obligations, which we ignored.

For SSDP, the first relevant keyword is in section 6.1 *Server Sends Downgrade Protection Hash*. According to the XEP, all sorting operations **must** be performed using `i;octet` collation as specified in Section 9.3 of RFC 4790 – *Conversations* implements this on lines 20 and 21 of `DowngradeProtection.java`.

Section 6.2 *Client Verifies the Downgrade Protection Hash* stipulates that if the hashes do not match, the client **must** fail the authentication. *Conversations* does this on line 214 of `ScramMechanism.java`.

Based on our analysis, the SSDP XEP covers all possible MITM scenarios. If both the server and the client support XEP 0474, either:

- the attacker does not modify the hash, but manipulates the server-advertised SASL mechanisms and/or channel-bindings – this results in a hash mismatch, which compliant clients such as *Conversations* (on line 214 of `ScramMechanism.java`) detect and abort the connection, warning the user,
- the attacker modifies the hash to match the manipulated list of SASL mechanisms and/or channel-bindings – this results in a SCRAM mismatch on the server side, since the client proof will be based on the hash the client observed as it came from the attacker, or
- the attacker does not modify either the hash or its aforementioned inputs – this makes it possible for both the client and the server to include channel-binding information in the SCRAM attributes, again resulting in a SCRAM mismatch (which *Conversations* does on line 293 of `ScramMechanism.java`) as (in the stronger case of `tls-unique` or `tls-exporter`) the two TLS sessions (the one used by the client and the one used by the server) will differ, or (in the weaker case of `tls-server-end-point`) the TLS certificates might differ if the attacker did not take the private key of the server, but rather made a CA, trusted by the client device, issue another certificate.

5.7 NF-001 — SBOM analysis

For the purpose of SBOM analysis, we exported a CycloneDX-formatted JSON BOM from the Gradle files using the CycloneDX Gradle Plugin. The output was processed using Trivy and OSV-Scanner to identify publicly known vulnerabilities that might affect *Conversations* code through its dependencies. Both tools identified the same two vulnerabilities that affect `com.google.guava:guava` version `31.1-jre`, and both were fixed in version `32.0.0-android`. Below is an excerpt from `libs/annotation-processor/build.gradle` that includes the affected line.

```
dependencies {

    implementation project(':libs:annotation')

    annotationProcessor 'com.google.auto.service:auto-service:1.0.1'
    api 'com.google.auto.service:auto-service-annotations:1.0.1'
    implementation 'com.google.guava:guava:31.1-jre'
```

However, based on manual inspection, neither of the two vulnerabilities affect the current *Conversations* codebase.

In case of CVE-2020-8908, `createTempDir` is not used anywhere in the codebase, while Guava itself makes no use of it internally.

```
% rg --stats createTempDir

0 matches
0 matched lines
0 files contained matches
2339 files searched
```

In case of CVE-2023-2976, `FileBackedOutputStream` is not used anywhere in the codebase, while Guava itself makes no use of it internally.

```
% rg --stats FileBackedOutputStream

0 matches
0 matched lines
0 files contained matches
2339 files searched
```

Based on the above, no upgrade is necessary right now from a security perspective, though our general advice is to keep all dependencies up to date.

6 Future Work

- **Perform a similar assessment on dependencies that haven't had one so far**
  Since this assessment considered dependencies out of scope, we recommend checking whether they have had any recent security assessments. To ensure supply chain security, components without proper assessments should receive one, prioritizing those that interact directly with untrusted content.
- **Set up automated code analysis for the issues uncovered and potential future ones**
  Automated code analysis and unit tests could make sure that the issues uncovered in this report are not reintroduced with future merges or developments. This could also include those described in non-findings to keep those areas in check, such as protocol state machines enforcing invariants and keeping to valid state transitions, TLS certificates matching the names properly, etc.
- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is a process that must be continuously evaluated and improved; this penetration test is just a single snapshot. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security.

7 Conclusion

We discovered 1 Moderate and 1 Low-severity issues during this penetration test.

The findings show that the codebase is mostly solid, and the findings affect rare circumstances outside the attacker's control.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

András Veres-Szentkirályi	András has CISSP, OSCP, GWAPT and SISE certifications in addition to an M.Sc. in Computer Engineering, and has been working in IT Security since 2009.
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 dougwoods (https://www.flickr.com/photos/deerwooduk/682390157/), "Cat on laptop", Image styling by Patricia Piolon, https://creativecommons.org/licenses/by-sa/2.0/legalcode.