Penetration Test Report

Monal IM

V 1.0
Amsterdam, April 15th, 2024
Confidential

Document Properties

Client	Monal IM
Title	Penetration Test Report
Targets	Monal XML parsing code (MLXMLNode) Monal implementation of SASL2, SCRAM and SSDP
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 4th, 2024	András Veres-Szentkirályi	Initial draft
0.2	April 5th, 2024	Marcus Bointon	Review
1.0	April 15th, 2024	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

Table of Contents

1 Executive Summary

1.1 Introduction

Between March 21, 2024 and April 3, 2024, Radically Open Security B.V. carried out a penetration test for Monal IM.

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 targets:

- Monal XML parsing code (MLXMLNode)
- Monal implementation of SASL2, SCRAM and SSDP

The audit specifically targets commit `68efd1702097bf8669f5df1391abc15f142c90ea`.

The scoped services are broken down as follows:

- Scoping and preparation: 1 days
- Security analysis of Monal XML parsing code (MLXMLNode): 1 days
- Security analysis of Monal implementation of SASL2, SCRAM and SSDP: 3 days
- **Total effort: 5 days**

1.3 Project objectives

ROS will perform a penetration test of Monal with Monal IM in order to assess the security of their XML parsing, SASL2, SCRAM, and SSDP implementations. To do so ROS will analyse the target source code and guide Monal IM in attempting to find inconsistencies, deviations from specification, and vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The security audit took place between March 21, 2024 and April 3, 2024.

1.5 Results In A Nutshell

During this crystal-box penetration test we found no security issues.

We only found minor implementation issues that do not have a security relevance, thus are not vulnerabilities, as an attacker cannot trigger them nor benefit from them occurring. These can be found in the non-findings section (page 6).

2 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends. For this specific project, this discusses accuracy of the implementation with respect to the applicable standards, and any inconsistencies we found.

2.1 NF-001 — Analysis of XML parsing and processing

Since Monal is an XMPP client, making heavy use of XML parsing and processing, the first specific threat outlined in the project description was that attackers could send mostly arbitrary XML to another XMPP user, thus attackers could manipulate the XML input of the `MLXMLNode`. We analyzed the source code to see if this resulted in any vulnerabilities, including denial of service or buffer overflows by sending specially crafted XMPP-Stanzas from one client to Monal.

As the project description stated, search patterns use Monal's own query language, through three sinks of `MLXMLNode`: `find`, `findFirst`, and `check`. Since both the second and the third call the first, we performed data flow analysis using all references to these three as sources, and `find` as a sink. Our findings were consistent with the project description – most patterns are static (i.e. known at compile time), and no queries are built explicitly using `[NSString stringWithFormat:@"...", ...]`.

There is a subset of methods within `MLPubSub`, `MLMessageProcessor`, `MLCall`, `MLOMEMO`, `MLPubSubProcessor`, and `XMPPDataForm` that use the built-in implicit `stringWithFormat`-style vararg string formatting, although `MLOMEMO` only uses it for numeric types, which offer fewer avenues for an attacker. However, since many of these include data potentially controlled by an attacker (such as `sdpMid` in `MLCall`), analysis continued to measure the security level of processing such queries.

The method `[MLXMLNode parseQueryEntry]` uses multiple layers of regular expressions to parse the query language, and in the end, only a single part of the input is processed as a format string: the values of `<key=value>` filters. This was in contrast to the project description, which states that names/keys can also have such format string parameters. This should be clarified in the internal/development documentation.

For values, the format string and the list of variadic arguments are passed on to a special implementation of `vasprintf` that calculates the necessary length during a first pass over the format string, and then allocates the exact length needed to fit the result into, kicking off a second pass that actually renders the output into that buffer. The code was modified in a way to handle `%@` placeholders (`NSString` instances), both here and on the call site of `vasprintf` to handle conversion between zero-terminated C-style `char*` strings and `NSString` instances properly by copying content and `free()`-ing `malloc()`'d buffers properly.

During these checks, we found that the method `dataWithHexString` in `HelperTools.m` does not check the return value of `malloc()` but this does not result in a vulnerability, since attackers cannot do much if the system is already in a state where heap allocation fails.

Analysis of the XML parsing and processing codebase did not identify any ways for a denial of service or buffer overflow through XML content received from other clients.

2.2 NF-002 — Analysis of SASL2 implementation

Monal implements SASL2 (XEP 0388), 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. Although we also analyzed the XEP itself, the main focus was on enumerating elements containing keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" from the standard, and interpreting their meaning from RFC 2119 to create a benchmark to measure Monal against. Of course, many of these refer to server-side obligations, which we ignored.

For SASL2, the first relevant keyword is in section 2.1 *SASL mechanism negotiation*. According to the XEP, all servers and clients supporting channel binding **must** implement *SASL Channel-Binding Type Capability* (XEP 0440). Monal not only implements it on the client side, but also enforces server-side support in `xmpp.m` using `kServerDoesNotFollowXep0440Error`.

The next relevant keyword appears in section 2.3 *Initiation*, stating that clients **should** include a `<user-agent/>` element and the contents of its `id` attribute **must** be a UUID v4. Monal implements this properly around line 2946 of `xmpp.m`.

Although sections 2.4 *During Authentication* and 2.5 *Challenges and Responses* state what clients must and must not send, these are discussed mainly in the context of what servers can accept and have no security relevance regarding SASL2.

Section 3 *Security Considerations* states that clients **should not** support SASL `PLAIN`, which aligns with Monal, that only supports SCRAM in the context of SASL2. Two paragraphs later it also states that SASL2 **must** only be used by clients after TLS negotiation; Monal waits properly for `STARTTLS` to finish before starting any form of authentication. The next paragraph states the clients **must not** send authentication requests included in the TLS 0-RTT ("early data") extension. Though Monal does no such thing, again, the only security implication of such a behavior would be triggered on the server side, in the form of replay attacks.

2.3 NF-003 — Analysis of SCRAM implementation

Monal 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 Monal 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`. In `SCRAM.m`, comments within the methods `quote` and `unquote` even state the lack of this `SASLprep` step, however, the RFC does not mandate their usage here, but rather before hashing the password, which happens in method `hashPasswordWithSalt` and uses UTF-8 without any particular normalization. (At least UTF-8 itself matches the requirement in section 3 *SCRAM Algorithm Overview*.) It's also worth noting that the aforementioned method `quote` quotes comma and equals characters in the wrong order, resulting in double-quoting the former. This could be easily fixed by swapping the order of the two lines before the `return` statement.

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. Monal does so in line 167 of `SCRAM.m`.

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 – Monal does so on line 90 of `SCRAM.m`.
- `m`: client **must** cause authentication failure upon its presence – Monal does so on line 103 of `SCRAM.m`.
- `r`: client **must** verify that the initial part of the nonce used in subsequent messages is the same as the nonce it initially specified – Monal does so on lines 99 and 100 of `SCRAM.m`.
- `c`: attribute is **required** – Monal includes it in the second authentication message.
- `i`: **must** be sent by the server along with the user's salt – Monal errors out either because it's missing or it being below 4096 on lines 108 and 118 of `SCRAM.m`.
- 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` – Monal implements this on line 84 of `SCRAM.m`.
- Clients that support mechanism negotiation and channel binding **must** use a `"p" gs2-cbind-flag` when the server offers the PLUS-variant – Monal does so on line 88 of `SCRAM.m`.
- If the client does not support channel binding, then it **must** use an `"n" gs2-cbind-flag` – Monal does so on line 84 of `SCRAM.m`.

Section 7 *Formal Syntax* states that all extensions are optional, i.e., unrecognized attributes not defined in the RFC **must** be ignored – Monal does so by parsing attributes into `NSDictionary` instances, and also helps fuzzing other implementations by including grease in parameter `"g"`. 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"` – Monal implements this in lines 821 through 929 of `MLStream.m`.

2.4 NF-004 — Analysis of SSDP implementation

Monal implements SASL SCRAM Downgrade Protection (XEP 0474), 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. Although we also analyzed the XEP itself, 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 Monal 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 – Monal implements this on lines 70 and 74 of `SCRAM.m`.

Section 6.2 *Client Verifies the Downgrade Protection Hash* stipulates that if the hashes do not match, the client **must** fail the authentication. Monal does so on line 115 of `SCRAM.m`.

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 Monal (on line 115 of `SCRAM.m`) 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 Monal does on line 167 of `SCRAM.m`), 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 computer, issue another certificate.

3 Future Work

- **Regular security assessments**
  Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.
- **Set up automated fuzzing harness**
  Automated fuzzing harnesses can help to test the code thoroughly before deployment, catching issues that might not be immediately apparent with manual testing methods. While this can identify all kinds of bugs, focusing on security, it might also trigger edge cases affecting memory safety and other security aspects. By ensuring that the application functions correctly with a variety of input data, including maliciously crafted ones, the project can avoid potential errors and make the software more robust overall. This does not replace manual testing, but rather complements it to catch low-hanging fruit with minimal effort in the long run. Ideally, it should be integrated into a CI/CD pipeline so that issues can be caught as early as possible.

4 Conclusion

We discovered no security issues during this penetration test.

The Monal source code is mostly sound and the relevant XEPs are well-thought-out. The codebase takes extensive advantage of the memory safety options of Objective-C, using low-level C primitives only where necessary.

We only found minor implementation issues that do not have a security relevance, thus are not vulnerabilities, as an attacker cannot trigger them nor benefit from them occurring. The SASL2, SCRAM, and SSDP implementations, tightly coupled together make it much harder for even well-funded attackers to mount successful MITM attacks against Monal users.

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 Veres-Szentkirályi has CISSP, OSCP, GWAPT, and SISE certifications in addition to an MSc. 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.