Preliminary response to the NGI Zero sponsored RadicallyOpenSecurity LibEuFin security audit in Q3'2024

Antoine d'Aligny

November 26, 2024

1 Abstract

This is the response to the penetration test audit report Radically Open Security created for LibEuFin in Q3'2024.

2 Management Summary

We thank Radically Open Security for their thorough report and analysis. Their recommendations have been valuable in enhancing the robustness of sensitive parts of our code, particularly in improving our authentication design. While we have addressed and resolved all serious identified issues, there are two minor recommendations that remain unimplemented at this time. We are, however, open to revisiting and addressing them in the future. However, at this time addressing them is not expected to be worth the cost (in terms of added complexity and dependencies).

In terms of impact, TLR-004 was probably the most severe issue and we are extremely happy that this was caught prior to actual use. TLR-005, TLR-007 and TLR-008 were reasonable hardening suggestions. TLR-001, TLR-002 and TLR-003 were mostly theoretical issues given the context in which they arise. TLR-012 is not actually a security issue. Nevertheless, all of the above have been addressed in the latest version. The minor information disclosure of TLR-006 is intentional by design (to improve usability and diagnostics) and thus addressing it was rejected by the maintainers.

3 Issues

3.1 TLR-001 — XML external entity injection

This issue was easily fixed by setting the appropriated feature flags to our XML parser:

```
DocumentBuilderFactory.newInstance().apply {
    // Enable secure processing
    setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true)
    // Disable all external access
```

```
    setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "")
    setAttribute(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "")
}
```

Although this security issue is of great concern when analyzing untrusted entries, we only analyze files generated by a trusted bank. It is therefore unlikely to be exploited in practice.

Fixed in fd77974c95f07882bf566840b3cb345a08d33923.

3.2 TLR-002 — Missing TLS client hardening measures

We now enable certificate revocation checks using system & security features flags:

```
System.setProperty("com.sun.net.ssl.checkRevocation", "true");
System.setProperty("com.sun.security.enableCRLDP", "true");
Security.setProperty("ocsp.enable", "true");
```

We now correctly error on `revoked.badssl.com` and `no-sct.badssl.com` .

As for certificate transparency, it is not yet natively supported by the JVM (JDK-8171275). There are open source projects for this, such as com.appmattus.certificatetransparency, but we are reluctant to add another third-party library just for this.

Partially fixed in 7f1ac03d8abcdf8012616cec2b41bd9cf7c3e434. We will enable certificate transparency when it is natively supported by the JVM in 9355.

3.3 TLR-003 — XML signature check bypass

We choose to check for the attribute `authenticate="true"` during parsing instead of `XMLUtil.verifyEbicsDocume` to return the validated nodes, as we need to access nodes that are not signed but are encrypted using a signed key per the EBICS protocol.

Fixed in 567bf82e855829652b2250f83acd1f8fcf82e37c.

3.4 TLR-004 — Insufficient entropy used for credential generation

This issue was fixed by using a cryptographically secure random number generator for admin account password and token values. We also renamed the utils function to `secureRand` to distinguish it more easily from the faster but unsecure `rand`.

```
/** Thread local cryptographically strong random number generator */
val SECURE_RNG = ThreadLocal.withInitial { SecureRandom() }

fun ByteArray.secureRand(): ByteArray {
    SECURE_RNG.get().nextBytes(this)
    return this
}
```

Fixed in 4e594a248ea62129129c7aec693e39e27d394acd.

3.5 TLR-005 — Lack of password and lockout policy

We have added minimum and maximum password lengths in accordance with NIST 800-63B in 2e2101a2dacbaf985fe0a57bbce7a14064c5704b.

We have enforced token authentication for most endpoints, to facilitate rate limiting and the use of more powerful hashes in the future, in bc35aba3fbac892029ea17ee7ee4d1b4949650c0.

We have not implemented the password blocklist yet, as we're not sure we want to implement it and how to do it. Calling a third-party API is expensive and makes testing without an internet connection complicated, and we certainly want the administrator to be able to customize this feature. We have an opened issue on this subject: 9356.

We have chosen not to block an account after too many password authentication attempts, as this would allow anyone who knows the username of an account, which could be public, to block that account. We have chosen to build on our existing 2FA logic by requiring 2FA for token creation and blocking accounts that fail 2FA too often. Thus, the only way to block an account is to know both the username and the password. Further information is available in the design document DD57.

We enforced 2FA for token creation in f7f37e16b2ca0bbedda3aa9f6cc84975e1ddb2ff.

We lock account after too many 2FA failures in 5f53f42899821e4fb7e353ae5ec6b76981ee5dd9.

3.6 TLR-006 — Username oracle

We believe that the distinction between username not found and username found is important for clients to be able to diagnose errors (typo in username or wrong password). Moreover, for an attacker, there would most likely always remain a timing side-channel to distinguish the two cases. So we think UX trumps information disclosure here.

We are open to adding an option not to disclose the difference in the future if customers request it.

3.7 TLR-007 — CLI asks for password as command line argument

We now accept both command line argument or password prompt, as we need a non-TTY alternative for script tests.

Fixed in 55bd4e47f8e2b4fdb800927c71b83edb47432cb6.

3.8 TLR-008 — Web frontend implements insecure session management

We now correctly remove the session token after logout, use a session duration of 30 minutes and refresh the session in the refresh window.

Fixed in 9a7dee809ec56bdc2aa4b33c425e3f5970692bc8.

3.9 NF-009 — Authorization testing

We try to share a common authentication logic for all endpoints, and this logic is thoroughly tested. We are pleased that no authentication problems have been found.

The Core Bank API is locked down as it is used by an authenticated web interface, while the Bank Integration API is used by wallets that are not authenticated, which explains why some withdrawal endpoints are not. We have ensured that these unauthenticated endpoints cannot be used to cause damage.

3.10 NF-010 — SQL injection testing

We access PostgreSQL using the standard JDBC API and bind all our variables. Although some queries are dynamically generated, we always use hard-coded snippets. We are pleased to note that no place where user data is incorporated into the query in raw form was found.

3.11 NF-011 — File operations testing

We take no files/paths from untrusted entries, so we're not surprised that no path traversals were found.

3.12 TLR-012 — Lack of authorization

This endpoint is not authenticated in the Bank Integration API, as it is designed to be used by the wallet that is not connected to the bank account. Access to this endpoint requires knowing or guessing a secure random ID. We accept the risk that a malicious user might abort a withdrawal if he glances at the victim's screen at the right time, as this can't result in loss of money or leakage of personal information. At worst, an attacker could annoy his victim forcing him to make another withdrawal.

We added authentication for consistency, even though there is also an unauthenticated endpoint.

Fixed in d727095c1bdffc9700ca657bc8bd88af44929949.