Code review of the Data Integration Service v2.8.3.1

CLIENT	**Federal Chancellery**	CLASSIFICATION	**Public**
DATE	**July 26, 2024**	AUTHOR	**Thomas Hofer**
VERSION	**1.0**	DISTRIBUTION	**Federal Chancellery**
GIT COMMIT	**bae10f6**	MODIFICATIONS	
STATUS	**Final**		

OS OBJECTIF SÉCURITÉ
Architecte de la sécurité informatique

Contents

OS

OS Objectif Sécurité SA
Route Cité-Ouest 19 - CH-1196 Gland
+41 22 364 85 70 - info@objectif-securite.ch

1 Introduction

1.1 Context

This report contains our review of a specific software used in the e-voting solution provided by Swiss Post. The review was mandated by the Federal Chancellery as part of the examination process according to OEV Article 10 paragraph 1.

The reviewed software (Data Information Service) is used to create the configuration file of an election event based on the electoral roll and the definition the election.

This report expands on our previous report on an earlier version of the software. The latest version reviewed was version 2.7.1.5, published in July 2023.

The version reviewed in this document is version 2.8.3.1, published on July 1st 2024, having the following commit hash: 79cbf733aa6dcf10b0eae9fa5c553d1f13b739da.

1.2 Execution of the work

Version 2.8.3.1: This version, published on July 1st, 2024, was reviewed between July 2nd and July 5th 2024. The complete source code was retrieved from Swiss Post's gitlab.[1]

We were able to compile the code directly from the source, as well as running and debugging the application.

1.3 Executive summary

The analysis of the results of the tests led us to the following conclusions:

No significant security issue: We found no evidence of code that would enable various attack scenarii that we imagined for the specific threat model of the environment in which the software is executed.

No exploitable vulnerable dependencies: At the time of the review, the documented vulnerabilities in the dependencies were not exploitable in the software's context.

Very minor code quality issues: Some minor quality issues have been introduced since our last review, without any significant impact on maintainability or auditability of the software.

No undocumented or hidden features: The software performs the tasks that are required of it, and no hidden features have been found within the source code.

[1] https://gitlab.com/swisspost-evoting/e-voting/data-integration-service

2 Analysis

This chapter reprises our analysis presented in previous versions of this report, since the software goals and context have not changed. Therefore, the threat analysis presented below applies to the latest version of the software, without changes.

The role of the software is to create a configuration file for an e-voting event. It processes the definition of an election and of the electoral roll and produces a single XML file containing all information necessary for setting up the the election event.

The software is installed on a secured standalone laptop, operated by at least two persons. Data is exchanged by USB keys. The laptop is used for configuring elections. It also contains the software (SDM) which implements a part of the cryptographic protocol of the setup phase.

During the setup phase, the SDM reads the configuration file created by the DIS and produces results which it signs with a key that is saved on the laptop.

The software uses the following assets:

- **Identity of voters:** the electoral roll,
- **Definition of the ballots:** questions, answers, candidates,
- **Signature key:** used for signing the produced xml documents.

The laptop also contains the following assets:

- **SDM software:** The SDM uses the output of the DIS and implements a part of the cryptographic protocol during the setup phase.
- **Codes:** initialisation key, return codes, confirmation and finalisation code, produced by the SDM.

Malicious operation by the software could result in the following classes of attack:

Attack 1: **Adding or removing voters:**
This is mitigated by the manual verification of the number of voters and the reclamations of voter who do not receive a voting card.

Attack 2: **Manipulating the rights of the voters:** The software could manipulate the configuration of the election to allow certain voters to participate to more – or fewer – parts of an election (e.g. cantonal or federal).
This is mitigated by the verification of the number of cast votes and the reclamations of voter who did not receive a voting card or are not able to participate in all expected parts of the election.

Attack 3: **Visible manipulation of voting cards to compromise individual verifiability:** The software could modify the phrasing of the questions or the names of candidates or parties to entice the voter to vote against their will. It could even manipulate the codes if it waits until they have been generated by other software on the laptop.
This is mitigated by the manual review of a sample of cards.

Attack 4: **Manipulating the trusted component:** The software could modify the behaviour of other software that runs on the same laptop or interacts with their data. It could for example try to leak

secret information through the data that is exported from the laptop or sign manipulated data with the keys that are available on the laptop.
This is mitigated by having the source code reviewed and a trusted build ceremony the demonstrates that the deployed software is the software that was reviewed.

We have identified two types of attackers:

- **Internal attackers:** An attacker who injects malicious code into the software, before it is delivered to the canton.
- **External attackers:** An attacker who injects code into the software through data that is given to the code. The code could for example be added to the address field of a voter, before the voter registry is imported.

Any malicious action by the operators or the software can be ignored in the analysis of the software because the operators already have the capability to manipulate the voting cards without the help of the software. Additionally, the operators are subject to strong security rules (e.g. 4 eyes principle) as mandated in Number 3 of the OEV Annex.

3 Analysis of the Code

3.1 Analysis specific to the identified attacks

- The code creates the output file based on the inputs it reads. We did not find any code which does not follow this principle and that would manipulate the resulting output file. This seems to exclude manipulations of the content of the cards or of the voting rights of the voters. (Attack 2 and 3)

- We did not find any code that would read or modify other files on the laptop, other than the files needed for its operation. This seems to exclude any attack on the cryptographic protocol. (Attack 4).

- We observed that the inputs of the software are XML files, which are handled by a standard XML parser, relying on well-defined XSD schemas. The use of a standard parser and a well-defined XSD schema should be sufficient in thwarting most injection risks.
  We noted that there is a specific test for the proper escaping of some characters which is executed during compilation. We further confirmed that typical characters that could be used for injections (`!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~`) are properly processed. While this does not guarantee that effective code injection is impossible, it makes it very improbable.

3.2 Generic analysis of the security of the code

- All issues that had been detected in earlier reviews of this software had been corrected.

Problem 1 - Results of `orElseThrow` ignored

In class `ECH015xv4Mapper`, there are two instances of calls to method `orElseThrow` for which the return values are ignored. These calls are simply used as checks that the expected value is present. The value is later computed again when it is needed.

```
Optional.ofNullable(ballotType.getBallotDescription())
    .orElseThrow(() -> new IllegalStateException(
        "Ballot description is missing. A ballot must include a description if no tie-break question title is available."));

final ch.ech.xmlns.ech_0155._4.BallotDescriptionInformationType.BallotDescriptionInfo ballotDescriptionInfo = ballotType.getBallotDescription()
    .getBallotDescriptionInfo().stream()
    .filter(bdi -> bdi.getLanguage().equals(tbqi.getLanguage()))
    .collect(MoreCollectors.onlyElement());
```

While this does not pose any immediate threat to the security or stability of the software, it creates a risk of discrepancy between the check and the usage of the value in future updates.

Recommendation

A variable could be introduced to retrieve the extracted value, rather than extracting it again separately. As such, the lines above could be modified as shown in the following extract.

```
ch.ech.xmlns.ech_0155._4.BallotDescriptionInformationType ballotDescription = Optional.ofNullable(ballotType.getBallotDescription())
    .orElseThrow(() -> new IllegalStateException(
        "Ballot description is missing. A ballot must include a description if no tie-break question title is available."));

final ch.ech.xmlns.ech_0155._4.BallotDescriptionInformationType.BallotDescriptionInfo ballotDescriptionInfo =
    ballotDescription
        .getBallotDescriptionInfo().stream()
```

```
        .filter(bdi -> bdi.getLanguage().equals(tbqi.getLanguage()))
        .collect(MoreCollectors.onlyElement());
```

Problem 2 - High cognitive complexity

In class `ContestService`, the additional description line for the candidates is currently computed as per the following code, which is flagged by SonarQube as having excessive complexity.

```
electionInformationType.getCandidate()
    .forEach(candidateType -> {
        if ((DisplayCandidateLineStyle.DEFAULT.equals(defaultDisplayCandidateLineStyle)
                && noSpecificDisplayCandidateLineStyle) || DisplayCandidateLineStyle.DEFAULT.equals(
                    specificDisplayCandidateLineStyle)) {
            // if the defaultDisplayCandidateLineStyle is DEFAULT and no specific displayCandidateLineStyle is defined,
            // or the specific displayCandidateLineStyle is DEFAULT, then the default displayCandidateLineStyle is used.
            addDefaultDisplayCandidateLinesStyle(candidateType);
        } else if ((DisplayCandidateLineStyle.WITH_PARTY_AFFILIATION.equals(defaultDisplayCandidateLineStyle)
                && noSpecificDisplayCandidateLineStyle) || DisplayCandidateLineStyle.WITH_PARTY_AFFILIATION.equals(
                    specificDisplayCandidateLineStyle)) {
            // if the defaultDisplayCandidateLineStyle is WITH_PARTY_AFFILIATION and no specific displayCandidateLineStyle is defined,
            // or the specific displayCandidateLineStyle is WITH_PARTY_AFFILIATION, then the displayCandidateLineStyle with party affiliation is used.
            addWithPartyAffiliationDisplayCandidateLinesStyle(candidateType);
        } else if ((DisplayCandidateLineStyle.WITH_PARTY_AFFILIATION_EXTRA_LINE.equals(defaultDisplayCandidateLineStyle)
                && noSpecificDisplayCandidateLineStyle) || DisplayCandidateLineStyle.WITH_PARTY_AFFILIATION_EXTRA_LINE.equals(
                    specificDisplayCandidateLineStyle)) {
            // if the defaultDisplayCandidateLineStyle is WITH_PARTY_AFFILIATION_EXTRA_LINE and no specific displayCandidateLineStyle is defined,
            // or the specific displayCandidateLineStyle is WITH_PARTY_AFFILIATION_EXTRA_LINE, i
            // then the displayCandidateLineStyle with party affiliation extra line is used.
            addWithPartyAffiliationExtraLineDisplayCandidateLinesStyle(candidateType);
        } else if ((DisplayCandidateLineStyle.WITH_PARTY_AFFILIATION_EXTRA_LINE_COMMA_SEPARATED.equals(defaultDisplayCandidateLineStyle)
                && noSpecificDisplayCandidateLineStyle) || DisplayCandidateLineStyle.WITH_PARTY_AFFILIATION_EXTRA_LINE_COMMA_SEPARATED.equals(
                    specificDisplayCandidateLineStyle)) {
            // if the defaultDisplayCandidateLineStyle is WITH_PARTY_AFFILIATION_EXTRA_LINE_COMMA_SEPARATED
            //      and no specific displayCandidateLineStyle is defined,
            // or the specific displayCandidateLineStyle is WITH_PARTY_AFFILIATION_EXTRA_LINE_COMMA_SEPARATED,
            // then the displayCandidateLineStyle with party affiliation extra line comma separated is used.
            addWithPartyAffiliationExtraLineCommaSeparatedDisplayCandidateLinesStyle(candidateType);
        } else if ((DisplayCandidateLineStyle.DEFAULT_COMMA_SEPARATED.equals(defaultDisplayCandidateLineStyle)
                && noSpecificDisplayCandidateLineStyle) || DisplayCandidateLineStyle.DEFAULT_COMMA_SEPARATED.equals(
                    specificDisplayCandidateLineStyle)) {
            // if the defaultDisplayCandidateLineStyle is DEFAULT_COMMA_SEPARATED
            //      and no specific displayCandidateLineStyle is defined,
            // or the specific displayCandidateLineStyle is DEFAULT_COMMA_SEPARATED,
            // then the default comma separated displayCandidateLineStyle is used.
            addDefaultCommaSeparatedDisplayCandidateLinesStyle(candidateType);
        } else {
            throw new IllegalStateException(String.format(
                "Unknown displayCandidateLineStyle. [displayCandidateLineStyle: %s, electionId: %s, candidateId: %s]",
                    defaultDisplayCandidateLineStyle, electionIdentification, candidateType.getCandidateIdentification()));
        }
    });
```

Indeed, the logic is not very easy to follow at first glance, despite the comments.

Recommendation

By extracting an `effectiveDisplayStyle` first, we can then simply use a single switch statement on that value afterwards, as shown below. To further improve readability the code has been extracted to a separate method.

```
private Consumer<CandidateType> addDisplayCandidateLineStyle(
    DisplayCandidateLineStyle defaultDisplayCandidateLineStyle,
    boolean noSpecificDisplayCandidateLineStyle,
    DisplayCandidateLineStyle specificDisplayCandidateLineStyle,
    String electionIdentification) {
        return candidateType -> {
            final DisplayCandidateLineStyle effectiveDisplayStyle =
                noSpecificDisplayCandidateLineStyle ? defaultDisplayCandidateLineStyle : specificDisplayCandidateLineStyle;
            switch (effectiveDisplayStyle) {
                    case DEFAULT -> addDefaultDisplayCandidateLinesStyle(candidateType);
                    case WITH_PARTY_AFFILIATION -> addWithPartyAffiliationDisplayCandidateLinesStyle(candidateType);
                    case WITH_PARTY_AFFILIATION_EXTRA_LINE ->
                                addWithPartyAffiliationExtraLineDisplayCandidateLinesStyle(candidateType);
                    case WITH_PARTY_AFFILIATION_EXTRA_LINE_COMMA_SEPARATED ->
                                addWithPartyAffiliationExtraLineCommaSeparatedDisplayCandidateLinesStyle(candidateType);
                    case DEFAULT_COMMA_SEPARATED -> addDefaultCommaSeparatedDisplayCandidateLinesStyle(candidateType);
                    default -> throw new IllegalStateException(String.format(
                                "Unknown displayCandidateLineStyle. [displayCandidateLineStyle: %s, electionId: %s, candidateId: %s]",
                                defaultDisplayCandidateLineStyle, electionIdentification, candidateType.getCandidateIdentification()));
            }
        };
}
```

Problem 3 - Ambiguous error catching in test

In test class `ECH015xv4MapperTest`, line 488, a call is made to `assertThrows`, where the provided argument has two separate elements that could throw similar errors.

```
@Test
void testBallotDescriptionInfoWithMissingDescriptions() {
    final BallotDescriptionInformationType source = createEchBallotDescriptionInformationTypeWithMissingDescriptions();
    final IllegalStateException exception = assertThrows(IllegalStateException.class, () -> mapper.convert(source, createEchBallotWithMissingQuestionInfo()));
    assertEquals("Ballot question info is not available in expected language. [language: rm]", exception.getMessage());
}
```

Both the call to `mapper.convert` and the call to `createEchBallotWithMissingQuestionInfo` could throw an `IllegalStateException`.

Recommendation

Since the method under test is the `mapper.convert` method, the code could be refactored as follows, so that only the errors thrown by calls to that method would be caught and handled by the assertion.

```
@Test
void testBallotDescriptionInfoWithMissingDescriptions() {
    final BallotDescriptionInformationType source = createEchBallotDescriptionInformationTypeWithMissingDescriptions();
    final BallotType echBallotWithMissingQuestionInfo = createEchBallotWithMissingQuestionInfo();
    final IllegalStateException exception = assertThrows(IllegalStateException.class, () -> mapper.convert(source, echBallotWithMissingQuestionInfo));
    assertEquals("Ballot question info is not available in expected language. [language: rm]", exception.getMessage());
}
```

Problem 4 - Unnecessary LinkedList creation

In the `AuthorizationService` class, line 106, the result of the `values()` call is then converted into a LinkedList before being added to the `AuthorizationType`.

```
result.getAuthorizationObject().addAll(new LinkedList<>(voter.getDomainOfInfluenceInfos().stream()
    .filter(doiInfo -> usedDOIids.contains(doiInfo.getDomainOfInfluence().getLocalId())).map(doiInfo -> map(voter, doiInfo))
    .collect(Collectors.toMap(
        ao -> ao.getDomainOfInfluence().getDomainOfInfluenceIdentification() + "|" +
                ao.getCountingCircle().getCountingCircleIdentification(),
        Function.identity(),
        (ao1, ao2) -> {
            LOGGER.warn("A voter with multiple identical domainOfInfluenceInfo found. Taking only the first one. [voterId: {}].",
            voter.getVoterIdentification());
            return ao1;
        }
        )).values()));
```

However, since the `addAll` method requires a `Collection` as an input, which is the same type the `values()` call returns, that cast is redundant and wastes an object creation.

Recommendation

The `LinkedList` constructor call can simply be removed.

Problem 5 - Error in capitalization of variable name

The Swiss Post have chosen to use camelCase for their variables and fields, by convention. However, in class `ContestService`, one field is named `byPassCheckConsistencyDates`.

Recommendation

Since *bypass* should be single word in this context, the field should be named `bypassCheckConsistencyDates` instead.

Problem 6 - Vulnerable dependency - xmlunit-core

The software depends on the library `xmlunit-core`, version 2.9.1. The latest released version at the time of writing this report is 2.10. Version 2.9.1 contains a critical vulnerability.

However, since this library is only used for testing, as it is pulled through the `spring-boot-starter-test` dependency, no XSLT transformations are performed, the vulnerability is not exploitable in the context of the software.

4 Conclusions

The source code that we reviewed seems to faithfully create the configuration file from its inputs. The trusted build ceremony guarantees that the software is made from the analysed source code.

We identified 4 types of attacks that could be mounted in the specific setup in which the application is used. They can be easily excluded by reviewing the code. We thus do not see any explicit danger in using the software.

Some of the changes introduced in this version bring minor code quality issues. However, none of the issues identified present a risk in the context of this software.

The only vulnerable dependency that was found in the project is only used during development, never deployed with the software. Furthermore, the vulnerability lies within a feature that is not used either in testing or in the deployed version. As such, it does not appear to pose any risk.