École Polytechnique Fédérale de Lausanne

Vulnerability Assessment of Swiss Post E-Voting System

by Vladyslav Zubkov

Master Semester Project Report

Prof. Bryan Ford
Project Advisor

Louis-Henri Merino
Project Supervisor

EPFL IC IINFCOM DEDIS
BC 210 (Bâtiment BC)
Station 14
CH-1015 Lausanne

June 8, 2022

Contents

Chapter 1

Introduction

For years, Swiss Post, Federal Chancellery, and independent researchers have been developing, reviewing, and creating legal frameworks for Switzerland's E-Voting system for elections and referendums. The system follows two main goals - being convenient for voters and, at the same time, ensuring the secrecy and integrity of votes. The system, specification, and development processes were made public. Federal Chancellery defined and published a draft of the Ordinance on Electronic Voting to facilitate and organize the goals that the future E-Voting system should follow. Since then, independent researchers have provided additional input to the Swiss Post to make the system more and more secure.

The project's goal is to review the documentation and source code for vulnerabilities and inconsistencies in documentation. However, the author decided to follow the methodology based on the adversarial assumptions within the threat model published by Swiss Post and review the defense mechanisms deployed by Swiss Post. The methodology is explained in chapter 4, and the main findings are presented in chapter 5.

The report first retraces the background of the E-Voting system and previous work done in the DEDIS Laboratory in chapter 2. Then the description of the relevant parts of the system is provided in chapter 3, and the methodology itself with the goals and tools used - in chapter 4. In the end, chapter 5 describes the findings.

In summary, this paper provides the following contributions:

- The author found and described five vulnerabilities, of which four were reported to the Swiss Post E-Voting Program.
- The author raises questions on whether the untrustworthy Voting Server should be an intermediary in communication between trustworthy components.

Chapter 2

Background

This section gives a short overview of the system history and the current state of the Bug Bounty Program.

2.1 Timeline

Since the previous work of Ella Kummer in 2021 [1], Swiss Post continued refactoring the code on all levels, from changing the code on the function level to abandoning components like Secure Logs. In version 0.12, Swiss Post changed the implementation of prime numbers generation, moved shared domain objects to an additional library, added MixnetInitialPayload to the verifier, and aligned the control components with zero-knowledge proofs to the system specification. In version 0.13, deserialization vulnerability has been fixed, dangerous functions were removed from code, fixed multiple minor issues related to cryptographic algorithms, aligned algorithms to one's described in the documentation, and merged both components of Secure Data Manager into one. The latest version is 0.14, which was released in the middle of the analysis, where SecureLogging information has been deprecated and removed because of the changes to the protocol, Start Voting Key length has been increased, and other minor issues were fixed.

2.2 Analysis

The documentation is much better written and more detailed than previously. However, as the system is still in a development mode, many components and algorithms either change or are still not implemented or partially implemented. That is why the analysis cannot be complete and does not aim to be.

2.3 Current Findings

Since the previous work of Ella Kummer, there were at least 32 reports submitted, from which 16 of them were accepted, mainly concerning source code, infrastructure, cryptography, and recent log4j vulnerability-related issues.

Chapter 3

Description of the system

3.1 Ella Kummer work

The best short description of the system was presented by Ella Kummer in her work [1]. That is why the current description mainly builds on top of what Ella Kummer described, and to get a complete picture of the system, one would need to read Chapter `Description of the system` in the previous report.

Additionally, this chapter gives an additional overview of the E-Voting system to understand the goals, tool selection, and analysis presented in this paper.

3.2 Security objectives of the system

Additionally to explanations of Individual Verifiability, Universal Verifiability, and Vote secrecy, the author would add additional objectives of the system as well as add additional information and redefine Vote Secrecy.

3.2.1 Integrity

The system should maintain the trustworthiness, accuracy, and consistency of the data during steps of the E-Voting system.

3.2.2 Confidentiality

The system should not leak sensitive data such as keys, votes, or personal information to unauthorized parties.

3.2.3 Availability

The system should be readily accessible to authorized parties. In terms of the E-Voting System, it also means that vote results should not be delayed.

3.2.4 Vote Secrecy

Vote Secrecy preserves the voter's vote option selection privacy, which means that no component and process should be able to know the vote content, except the last one after doing full decryption of the vote.

3.2.5 Vote Integrity

The system should preserve the vote's state across the system processes. Additionally, the vote should not be damaged and, as a result, discarded.

3.3 High Level System Architecture

In the figure 3.1, the system's high-level architecture and communication between components are presented. All building blocks and actors of the system were defined in the previous report [1].

Additionally, the author would like to redefine or give more information on the following blocks and actors.

3.3.1 Voter

The majority of voters comply with the adversaries and give their access codes for potential attacks. [3]

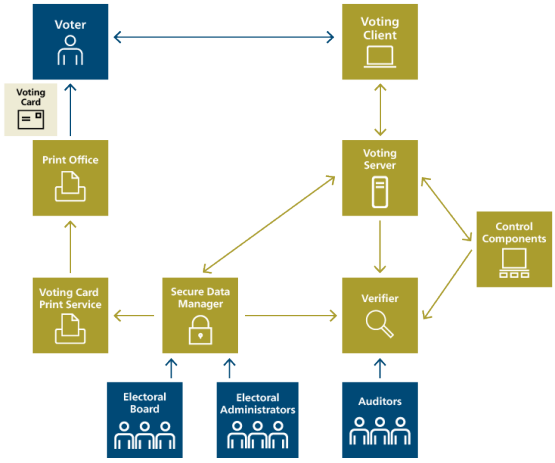


Figure 3.1: Overview of e-voting service [2]

3.3.2 Electoralboard Administrator

Electoral board Administrators are trustworthy only during the setup phase. As a result, according to the comment in one of the reports, they are untrustworthy, but they operate with a trustworthy USB drive and Secure Data Manager component. Furthermore, to prevent possible manipulations from an administrator, operational safeguards are in place, such as multiple-accessor verification[1].

3.3.3 Voting Client (Voting Portal)

Voting Client sometimes is referred to as Voting Portal. It is a front-end system seen as trustworthy for vote secrecy but untrustworthy overall. That means that once and if Voting Client is compromised, it can break vote secrecy of the future votes.

3.3.4 Voting Server

Voting Server is a cluster of 10 microservices and one universal API Gateway that are seen as a back-end for Voting Client, Message Broker user, Data supplier to Control Components, and an intermediary for Secure Data Manager and Control Components. It is not evident from the diagram above where the Message Broker is shown. Still, it is a component between Voting Server and Control Components and is used as a communication channel between them. The voting Server is seen as untrustworthy.

[1] https://www.bk.admin.ch/bk/en/home/politische-rechte/e-voting/sicherheit-beim-e-voting.html

3.3.5 Control Components

4 Control Components are tasked with shuffling the encrypted votes, generating return codes, and decrypting votes at the end of the election. Together, they compose 1 component in the system. Overall there are two types of control components - CCR and CCM, which stand for Return Codes Control Component and Mixing Control Components. In addition, one CCM component is absent in the Control Components part of the system but is present in the offline part of Secure Data Manager. This CCM is needed for the final decryption of the votes.

3.3.6 Secure Data Manager

SDM, also known as Secure Data Manager, is a component needed for the setup and administration of the election. The SDM consists of an online and offline component with communication between them going through a USB drive. Most of the operations are executed by electoral board administrators. Additionally, Secure Data Manager queries one of the microservices of Voting Server and uses Voting Server as an intermediary who will deliver data or tasks to Control Components. The transmitted data is signed to have a secure channel between Secure Data Manager and Control Components. Additionally, a secure Data manager and USB drive used for data transmission are trustworthy components.

3.4 Malicious Actors

In the diagram 3.2, a non-extensive attack surface of the Swiss Post E-Voting system is presented. During the attack surface modeling, the author made assumptions based on the Swiss Post threat model [3]. Additionally, the Attack Surface would be expanded in the following diagram.

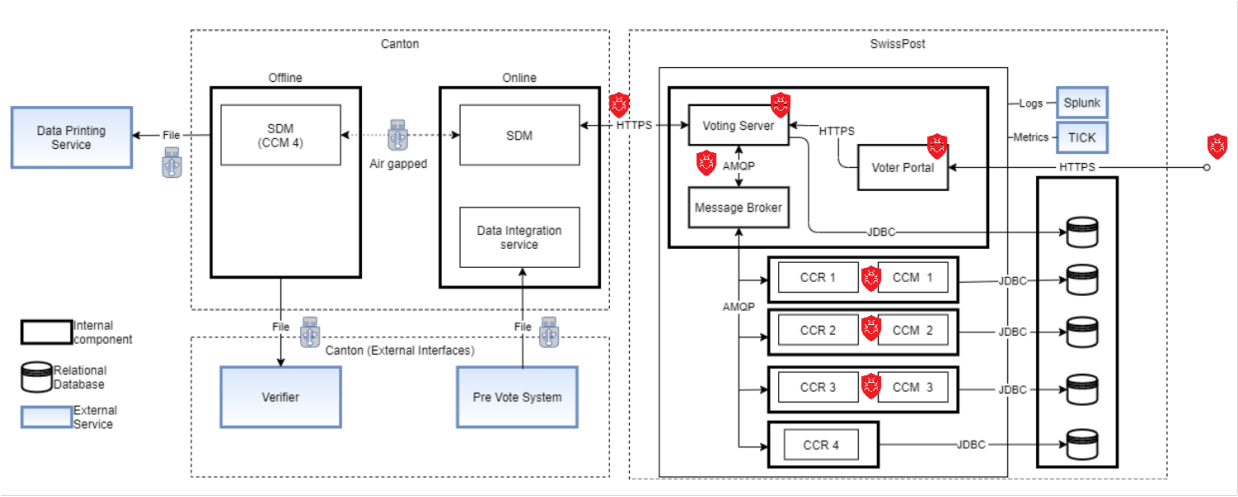


Figure 3.2: Deployment overview with possible adversary locations [2]

3.4.1 Malicious Voter

A malicious Voter is a voter whose intent is to get into internal components of the system during the voting process. It could be either a voter him- or herself or an adversary who has someone else voting keys. A malicious voter can try to access the Voting Client, one or more microservices of the Voting Server, or the whole Voting Server.

3.4.2 Malicious Microservice

A malicious microservice is a microservice that the adversary controls. Microservices run in the Voting Server component in a docker container. An adversary controlling the microservice can try escaping the container and escalate their privilege to root access, thus compromising the whole Voting Server. Therefore, microservices are untrustworthy as they are subcomponents of Voting Server.

3.4.3 Malicious Voting Server

A malicious voting server is a voting server component and all its microservices that an adversary controls. An adversary can try to access and perform lateral movements to online Secure Data Manager, Voting Client, 4 CCR components, and 3 CCM components from this component.

3.4.4 Malicous Administrator

A malicious administrator is such an administrator who was either bribed or forced to do evil actions. Additionally, a malicious administrator could be accompanied by a second person who should have, in theory, authorized their actions. Such an administrator can try to access Voting Server, Control Components, compromise online and offline front-end components of Secure Data Manager.

Chapter 4

Methodology

In this chapter, the author presents vital differences between Ella Kummer's work and ours. Then the author defines initial goals that were a bit changed during the analysis due to a fresh release and a mistake in the initial analysis. In the end, the author presents a selection of tools used to perform analysis with a few tips for configuring and using them, which might be helpful for future research.

4.1 Comparison with a previous work

In the previous analysis of Swiss Post E-Voting performed by Ella Kummer, the key focus was the implementation of hashing and cryptographic algorithms, whether or not they followed the documentation. Additionally, it was checked whether there are no imports of insecure libraries, whether the signatures are authenticated and whether there are no buffer, heap or integer overflows.

After analyzing the previous and other reports found on the Internet, the author concluded that the cryptographic part is checked extensively. Therefore, there is a lower chance of finding issues in these parts of the system. Additionally, the author considered that E-Voting relies heavily on cryptography, which is why the author thought it would be the most guarded and well-implemented component.

That is why the author decided not to go through the front door and focus on the implementation and infrastructure-related components.

4.1.1 Documentation analysis

First, the author read the documentation and looked for strange wording, which might lead developers to implement some parts of the system incorrectly. Then, the author had to determine which components to focus on and why from the documentation.

4.1.2 Static analysis

After that, the author decided to perform static analysis by following a novel bug bounty technique called `Spray and pray`, which meant looking for sinks or sources on the whole e-voting repository without knowing what to check and where and just waiting for something to pop up.

4.1.3 Dynamic analysis

The author decided to perform dynamic analysis by setting up parts of the system and then fuzzing these components or verifying the static analysis findings. However, after spending five full days trying to set up different parts of the system, the author was unsuccessful. As a result, the author talked to Ruben Santamarta, also known as @reversemode, who is the best hunter on the Swiss Post E-Voting Program. Rubben assured the author that he was never able to set up any component, and he mostly did a static analysis.

Then the author opened an issue on GitLab and asked for the setup of a dynamic analysis of the system. Unfortunately, the author got a reply that the documentation yet has no recommendations for DAST[1]. Therefore, the author decided not to do any dynamic analysis.

The author believes that the Swiss Post would only benefit from making the parts of the system easy to set up for DAST analysis. In that way, researchers would be able to test their hypotheses and verify the proof of concepts that would eventually lead to higher-quality reports and bugs that can only be discovered with the help of DAST.

4.1.4 Adversarial scenarios

The author brainstormed possible adversarial scenarios where some system components might be compromised. These scenarios are mainly theoretical and rely on the current threat model, current and past findings, and precedents of the attacks in different systems with a high chance of not being mitigated by the current defenses.

[1] https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/issues/36

4.2 Goals

In this part of the chapter, the author defines the goals that the author drew after analyzing the documentation.

4.2.1 Fortify

Swiss Post uses Fortify product as the primary tool for checking security issues of the source code. Therefore, the author wanted to check how Swiss Post uses Fortify to perform security checks and suggest improvements.

4.2.2 SonarQube

Swiss Post uses SonarQube product as a secondary tool for checking security and looking for code smells and other inconsistencies. The author wants to check how Swiss Post runs it and suggest improvements.

4.2.3 Documentation analysis

This goal is mainly related to checking the documentation and implementation alignment.

Multiple Choice Option selection during voting

While analyzing the documentation, the author came across lucrative wording. It was explained that during the multiple-choice voting, the user has a choice between YES, NO, and EMPTY options, and he cannot choose the wrong option as YES and NO together. Therefore, the author decided to verify whether the user can select YES and EMPTY choices together and send such a vote that might be either discarded or break assumptions of the algorithms operating on this data.

Secure Logs

The E-Voting system sued secure logs that could not be forged by the adversary in one of the algorithms. So the author decided to look at them and determine whether an adversary could manipulate any part of the secure logs.

JS Response Check

As defined in security objectives, the JS Response check is used to verify hashes of files given by the voting server to the voting client. The author wanted to look at the implementation of the checks done, which hash functions are used, and whether it is possible to inject files that would evade the check.

Data Flow

Here the goal is to check the data flow from accepting it on API components and validating it to the sinks. This might be seen as a check from source to sinks. Then the author wants to do the same, but for the reverse flow from sinks to sources and determine whether there are any vulnerabilities. In addition, the author will use a novel technique - `Spray and Pray`. The usage of semgrep and CodeQL is present in this part of the report.

4.2.4 Adversarial scenarios

The goal is to brainstorm the scenarios where one of the untrustworthy components is compromised and determine how an adversary would perform lateral movements and privilege escalation to gain more privileges in the system.

4.3 Tools

The main tools that the author ended up using were CodeQL and semgrep, in addition to the SonarQube. The author also was planning to use Burp Suite for Dynamic Analysis, but because the author could not set up any components, the author did not use it.

4.3.1 CodeQL

CodeQL is a highly customizable semantic code analysis engine developed by Github. It ships with already defined language-specific queries that try to look for sources, sinks, specific vulnerabilities, or attack vector patterns. [2] To set it up, one should follow the GitHub manual[3]. After that, for convenience, the author installed Visual Studio Code and the CodeQL plugin, as there is no such plugin for IntelliJ IDEA.

[2] https://codeql.github.com/
[3] https://codeql.github.com/docs/codeql-cli/getting-started-with-the-codeql-cli/

After successfully configuring the IDE and CodeQL CLI tool, the author used the following command to create language-specific databases which the query uses.

```
codeql database create --language=java --source-root /path/to/source/
    or/subsource/directory DatabaseName
```

Then the CodeQL project should be created as described in the documentation, and the database should be imported into VS Code through the plugin. Then the VS Code is set up for creating custom queries and running the provided ones.

4.3.2 Semgrep

Semgrep is a semantic grep, which could be adopted by the Swiss Post as a replacement for SonarQube. It could be used with default patterns, or new patterns could be added and used to find issues related to security or standards. The installation is straightforward and described on this page[4]. In addition to the CLI tool, the author signed up for the platform and used the platform to store the results across the runs of semgrep.

[4] https://semgrep.dev/getting-started

Chapter 5

Analysis

5.1 Fortify

Analyzing how Swiss Post uses this product is impossible because there is no information on how it is done.

As of July 2022, Fortify is one of the leaders in the industry of security analyzers, and its parts could be configured to meet different goals. It incorporates six security analyzers, namely: Data Flow, Control Flow, Structural, Semantic, Configuration, and buffer[1]. The Data Flow analyzer tries to detect vulnerabilities involving tainted data, while the Control Flow analyzer detects dangerous function invocation patterns. The structure analyzer detects misconfigurations and potentially harmful actions in the structure of the programs and libraries used or created. Semantic analyzer serves the same goal as semgrep tool and may be referred to as intelligent semantic grep to find known flaw patterns in the code. The configuration analyzer should check the deployment configuration files and look for various misconfigurations. Finally, the buffer analyzer is concerned with overflow vulnerabilities.

The analysis of data flow and adversarial scenarios done in this project may signal that the usage of Fortify analyzers is either wrong or insufficient because the author found issues that Configuration, Semantic, and Data Flow analyzers should have caught. To conclude, additional information from the Swiss Post E-Voting team is needed.

[1] https://stackoverflow.com/questions/13051974/how-does-fortify-software-work

5.2 SonarQube

Sonarqube is a tool for developers that aims to help with code quality, spot security vulnerabilities, and manage technical debt. However, security scanning is not the primary value SonarQube brings to the clients but a supplemental one. For example, the Swiss Post E-Voting team states that they ran SonarQube with the built-in Sonar way quality profile. The analysis revealed 0 bugs and 0 vulnerabilities which the public might see as the fact that the system is secure. Still, in reality, it says nothing[2].

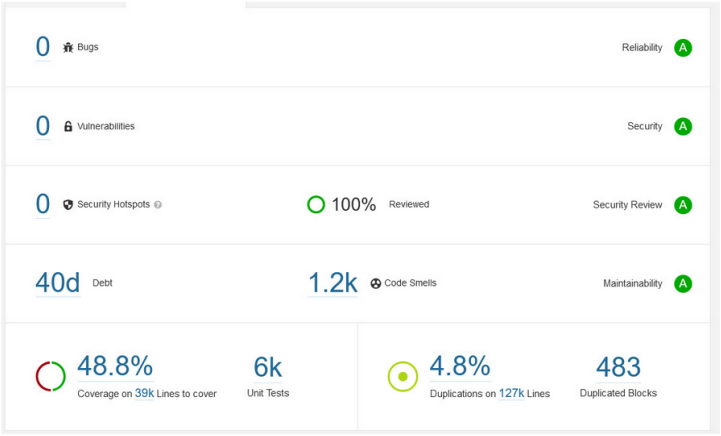


Figure 5.1: Swiss Post SonarQube results

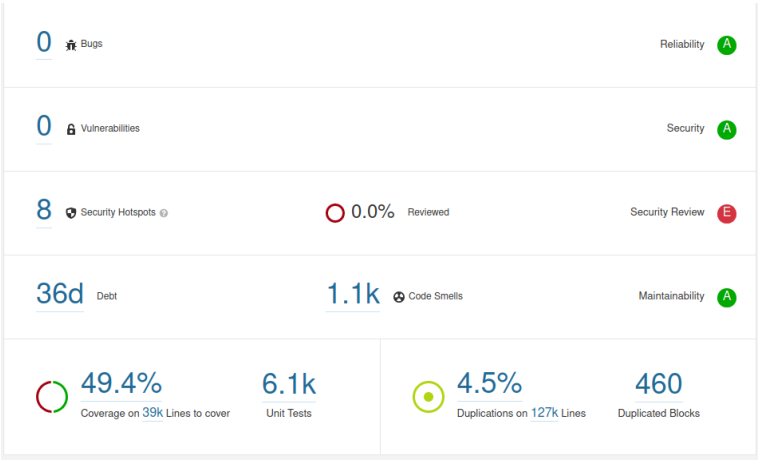


Figure 5.2: Our SonarQube results

The author installed the SonarQube Community version, ran it against the source code with a default profile, and got the output with eight security hotspots. In comparison, Swiss Post had

[2]https://gitlab.com/swisspost-evoting/e-voting/e-voting

0 security hotspots. Additionally, the number of unit tests, code smells, and technical debt has changed. In the diagrams 5.1 and 5.2, you could see these changes.

The strange behavior might result from either using old images or configuring the SonarQube in a way that skips a specific place. The change in security hotspots is a result of Swiss Post reviewing them, but as the Swiss Post E-Voting team reviews the issue, they determine whether it is a false positive or not. If it is, a `@SuppressWarnings()` directive for SonarQube is left in the source code to signal that SonarQube should not consider the marked issue a security hotspot.

Additionally, SonarQube could be configured, and its own rules could be run using it. To evaluate the extent to which the Swiss Post E-Voting team utilizes the SonarQube, the author would suggest that the Swiss Post reveal the commands and configurations they use to test the system. In the future, security experts would be able to express their views on the amount and configuration of automated security testing that Swiss Post does. Otherwise, Swiss Post E-Voting the whole setup might be seen as security through obscurity.

Swiss Post would only benefit from revealing how they use Fortify and SonarQube because our data flow analysis showed findings that, in theory, should have been eliminated by at least Fortify tools. Overall, the Swiss Post team does an excellent job of catching most of the bugs, but the security processes in the pipeline need additional tuning so that the processes become more efficient. That is why we suggest the Swiss Post share additional information on the CI/CD process, especially the security-related verifications done.

5.3 Documentation

5.3.1 Multiple Choice Option selection

In the first place, the author made a mistake during the documentation analysis and read what the author wanted instead of what the document contained. The author assumed that the E-Voting system has multi-choice voting, where a voter could choose one or more answers for 1 question, and that is why The author decided to look at how the author can force the system into an invalid state.

However, as the author found out later, the author misread or probably did not notice the sentence stating that the voter is expected to select exactly one voting option for the first question and one for the second question.[4]

Then the author quickly went through the implementation and has not found any issues concerning correctnessIds, which are used to enforce the proper selection of options at the same time without revealing the voter's vote content.

5.3.2 Secure Logs

As of version 0.14.0.0, Secure Logs are deprecated, and Swiss Post is not planning to use them in the future[3].

Despite this fact, the author decided to reveal possible manipulations of the Secure Logs, which were identified by Static Analysis only, and have not been checked by Dynamic Analysis, which could have confirmed the finding.

In the figure 5.3, you can see an example of secure logs.

Secure Logs can only be secure against an adversary who does not control the Control Component because if the adversary controls a Component, it can dump the keys from memory and forge its own secure logs. Another finding which is unconfirmed by Dynamic Analysis is that the adversary having control over the writable directory of Secure Logs and having to write permission to the logs, but not having access to the keys and Control Component processes, may try to delete or change the content of 2 different secure logs so that they still could go through the verification. Still, the logs do not refer to the process needed to produce them.

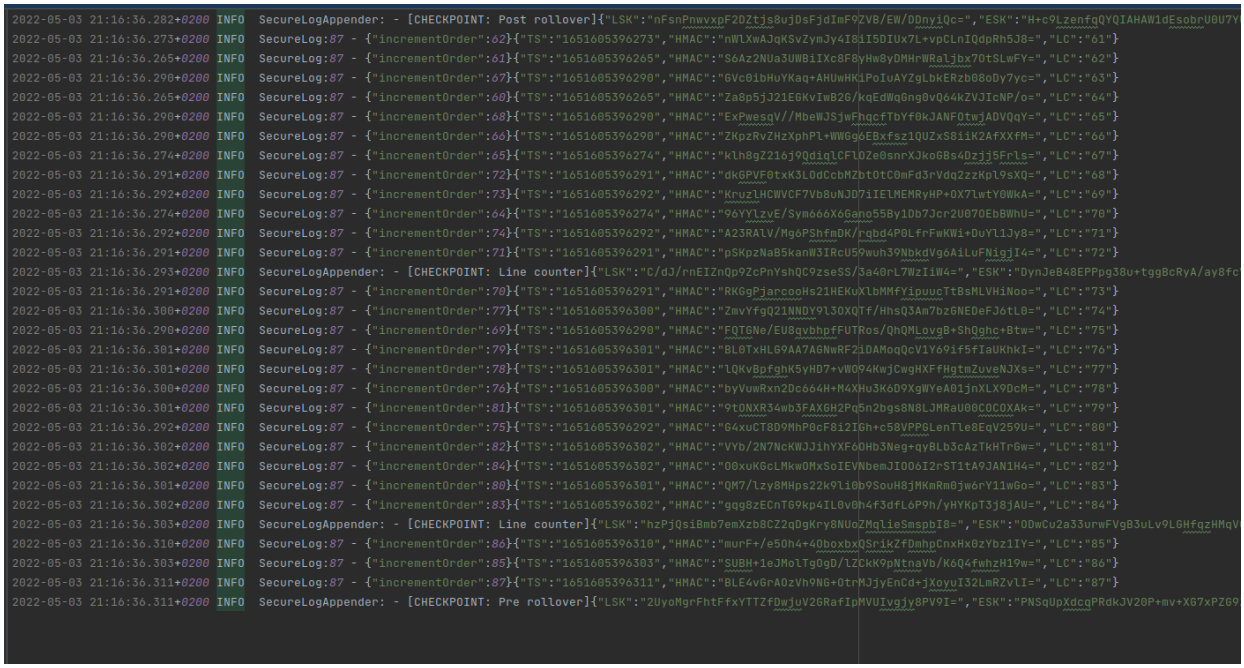


Figure 5.3: Secure Logs output

To be secure from adversaries that do not have the above-mentioned capabilities - The secure Logs component uses HMACs, last session key, encrypted key, timestamp, counter, and

[3]https://gitlab.com/swisspost-evoting/e-voting/e-voting

signatures formatted in JSON format to create a chain of logs connected by keys, HMACs, and signatures so that no adversary without the access to the key could be able to truncate parts of the logs without signaling the verifier that it was manipulated.

5.3.3 JS Response Check

Swiss Post uses JS Response Check in the reverse proxy between Voting Server and Voting Client, Voting Client, and Voter to ensure that the client can correctly encrypt the ballots. The goal here is to examine which technologies are used and whether the hash is collision-resistant, and a malicious Voting Server will not be able to come up with the same hash for a malicious javascript file.

The research is not complete because the implementation of JS Response Check, configuration, and selection of reverse proxy is unknown, as Swiss Post has not shared this information so far. Nevertheless, it could be deduced from the source code that the reverse proxy used is either Apache or Tomcat. Then there is no such term as "JS Response Check" over the Internet, and it is a custom term referred to either a known technique or a home-built one by Swiss Post.

After gaining more knowledge about similar techniques, the author may suggest that Swiss Post uses one of the following techniques:

1. Subresource integrity or SRI is a feature that enables browsers to verify that a specific javascript file was delivered without alterations by setting a hash value to the integrity parameter of a script in HTML and then letting the browser check the values. Additional information could be found at the following link[5].

2. Cloudflare and Code Verify extension - is an example of a technique similar to JS Response Check, where a reverse proxy has the functionality of code verify. This technique works in the following way - Whatsapp (Server) sends hash values for JS resources for a specific version to Cloudflare. Then when a client with Code Verify uses a Whatsapp client, the Code Verify extension queries Cloudflare for the hash values of JS resources for the version of a client used and then checks them. In this case, reverse proxy functions as a Code Verify and Cloudflare simultaneously by using an allowlist of hash values.

The best variant of these options is the last one, where the hash values are checked against an allowlist of hashes. Still, then additional issues with configuring an allowlist and a possibility of finding a collision or a wrong implementation of a hash function might arise, but because the author does not have the needed information, the author can only suggest. The author asked Swiss Post about JS Response Check[4], but at this moment, the author does not have a reply.

[4] https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/issues/36

5.4 Data Flow

The author used all the techniques described in the goals in this section. Overall, the ultimate goal is to break confidentiality, availability, or integrity of the system or part of the system, possibly breaking vote secrecy and vote integrity if possible by the achieved privileges after exploitation.

Using semgrep, the author got 216 issues of different severity for version 0.13.0.0 and 116 issues of varying severity for version 0.14.0.0, which signals that the Security Posture improves at least for this metric. Each alert was investigated and categorized as issues already known by Swiss Post, false positives, and those that needed to be explored. The author used such categories because semgrep is a semantic grep and cannot take into account all custom checks that Swiss Post may be doing, and that is why each alert needs might fall into one of the three groups. Then the author used CodeQL to find the data flow from sinks to sources and vice versa. For Voting Server, the author had 531 flows from sinks to sources using untrusted data, while for SDM, the author had 265 flows. Voter portal and voting-client-js together had 472 flows from sink to source. So overall, it is 1268 flows that need to be checked. That is why the author decided to focus only on promising ones, where the sinks are the most dangerous, and find and check mechanisms that do sanitization and validation of data, plus look at other flows at scale.

5.4.1 Voting Server API mappings and input validation

The author started by looking at the flows from source to sinks. SDM, Voting Server, and Control Components parts are written in Java and use the Spring Boot framework to structure and server REST API for external and internal use.

As said before, Voting Server consists of 10 microservices. Each microservice has its REST API used as an interface to communicate with it. To create a single interface between microservices and external systems, Swiss Post created another component - API Gateway, whose primary goal is to combine APIs of microservices into one and create custom functions that incorporate calls to a few microservices APIs that all together do one job. The framework has the following decorators that signal that the following function will be called once the data is received at a particular path. These decorators are:

1. `@GET` or `@POST` - which describes the HTTP method of the API endpoint
2. `@Path` - which describes the path and dynamic variables
3. `@Consumes` - which describes the content type of the received packet
4. `@Produces` - which describes the content type of a response packet.

The author can understand which data flows and where by looking for these decorators. Then once the data is assigned to the variables, Swiss Post uses some validation of input by utilizing

`validateUUID(value)` function that validates whether a received value is indeed in a UUID format. This function is often called in all the APIs that consume a variable with UUID, which is heavily used in the E-Voting System. The following figure shows the code of the `validateUUID` function.

```
public final class UUIDValidations {
    @VisibleForTesting
    static final int UUID_LENGTH = 32;
    @VisibleForTesting
    static final String UUID_ALPHABET = "0123456789abcdef";
    private static final String UUID_REGEX = String.format("^[%s]{%d}$", "0123456789abcdef", 32);
    private static final Pattern UUID_PATTERN;

    private UUIDValidations() {
    }

    public static String validateUUID(String toValidate) {
        Preconditions.checkNotNull(toValidate);
        if (!UUID_PATTERN.matcher(toValidate).matches()) {
            throw new FailedValidationException(String.format("The given string does not comply with the required UUID format. [string: %s, format: %s].", toValidate, UUID_REGEX));
        } else {
            return toValidate;
        }
    }

    static {
        UUID_PATTERN = Pattern.compile(UUID_REGEX);
    }
}
```

Figure 5.4: validateUUID function code

The function is safe and uses regex to validate the input. It checks whether the input length equals 32 and the alphabet contains only permitted characters such as 0123456789abcdef.

That means that all UUID values validated by this function pose no threat to the system, except for brute force or indirect object references, which heavily rely on brute force in general. The brute force might be eliminated by the Firewalls and WAFs that Swiss Post should deploy. Still, it is currently unknown whether it is indeed so because there are no published configurations of WAFs and Firewalls.

Other variables are not validated explicitly. That is why it seemed that there was no verification and validation at first glance, but in fact, there is, and it is described in the following subchapter.

5.4.2 Voting Server sanitization

As described in the previous part, there is the first level of validation and sanitization at the Voting Server. When API Gateway receives a request, the request goes through the first level of sanitization, which is done by default by a custom `SanitizerDataHttpServletRequestWrapper` class at the Framework level.

The class wraps the HTTP request with the logic of sanitizing data, validating whether it is sanitized and extracting clean data or sending a signal as an error about the supplied incorrect data.

There are content-specific checks for JSON, CSV, and just strings.

```java
public ServletInputStream getInputStream() throws IOException {
    LOGGER.debug(SEPARATOR_LINE);
    LOGGER.debug("Input stream {}", req.getContentType());
    try {
        String header = req.getHeader(CONTENT_TYPE);
        ServletInputStream inputStream = req.getInputStream();

        String inputString;
        if (header != null) {
            switch (header) {
            case MediaType.APPLICATION_JSON + WITH_CHARSET_UTF_8:
            case MediaType.APPLICATION_JSON:
                inputString = IOUtils.toString(inputStream, StandardCharsets.UTF_8);
                String sanitizedJSON = JsonSanitizer.sanitize(inputString);
                checkIfDataIsSanitized(header, inputString, sanitizedJSON);
                return convertToServletInputStream(inputString);
            case TEXT_CSV:
                inputString = IOUtils.toString(inputStream, StandardCharsets.UTF_8);
                checkIfCSVIsSanitized(inputString);
                return convertToServletInputStream(inputString);
            default:
                inputString = IOUtils.toString(inputStream, StandardCharsets.UTF_8);
                String sanitizedIS = sanitize(inputString);
                checkIfDataIsSanitized(header, inputString, sanitizedIS);
                return convertToServletInputStream(inputString);
            }
        } else {
            inputString = IOUtils.toString(inputStream, StandardCharsets.UTF_8);
            String sanitizedIS = sanitize(inputString);
            checkIfDataIsSanitized(NO_HEADER, inputString, sanitizedIS);
            return convertToServletInputStream(inputString);
        }
    } catch (IOException e) {
        LOGGER.error("Error converting input stream to string in sanitizer data filter...");
        throw e;
    }
}
```

Figure 5.5: Sanitization based on content type

In the case of a JSON, `JsonSanitizer` from Google is used, and then a custom-developed `sanitize` function. In the case of a CSV, a custom-developed `checkIfCSVIsSanitized` function is used, which splits the CSV by new lines, and replaces all escaped backward slashes with an empty value. Then `sanitize` function is used as if it were just a string value in the first place.

The sanitize function has four stages:

1. `Normalizer.normalize` call normalizes an input so that validation couldn't be evaded.
2. `ESAPI` call validates input against a proposed regex found in the ESAPI.properties file.
3. Replacement of `\0` values - deletes `0x00` bytes inside the string so that the parsers would work correctly.
4. `Jsoup.clean` call - eliminates HTML tags, which helps to prevent Cross-Site Scripting attacks.

```java
private static String sanitize(String value) {
    if (value == null) {
        return null;
    }

    String debugStatement = "Original value: " + value;
    LOGGER.debug(debugStatement);

    String sanitizedData = value;

    // Normalize
    sanitizedData = Normalizer.normalize(sanitizedData, Normalizer.Form.NFKC);

    // Use the ESAPI library to avoid encoded attacks.
    sanitizedData = ESAPI.encoder().canonicalize(sanitizedData);

    // Avoid null characters
    sanitizedData = StringUtils.replace(sanitizedData,  searchString: "\0",  replacement: "");

    // Clean out HTML
    sanitizedData = Jsoup.clean(sanitizedData, Safelist.none());

    debugStatement = "New value: " + sanitizedData;
    LOGGER.debug(debugStatement);
    LOGGER.debug(SEPARATOR_LINE);

    return sanitizedData;

}
```

Figure 5.6: sanitize function listing

```
Validator.ConfigurationFile=validation.properties
# Validators used by ESAPI
Validator.AccountName=^[a-zA-Z0-9]{3,20}$
Validator.SystemCommand=^[a-zA-Z\\-\\/]{1,64}$
Validator.RoleName=^[a-z]{1,20}$
#the word TEST below should be changed to your application
#name - only relative URL's are supported
Validator.Redirect=^\\/test.*$
# Global HTTP Validation Rules
# Values with Base64 encoded data (e.g. encrypted state) will need at least [a-zA-Z0-9\/+=]
Validator.HTTPScheme=^(http|https)$
Validator.HTTPServerName=^[a-zA-Z0-9_.\\-]*$
Validator.HTTPParameterName=^[a-zA-Z0-9_]{1,32}$
Validator.HTTPParameterValue=^[a-zA-Z0-9.\\-\\/+=@_ ]*$
Validator.HTTPCookieName=^[a-zA-Z0-9\\-_]{1,32}$
Validator.HTTPCookieValue=^[a-zA-Z0-9\\-\\/+=_ ]*$
Validator.HTTPHeaderName=^[a-zA-Z0-9\\-_]{1,32}$
Validator.HTTPHeaderValue=^[a-zA-Z0-9()\\-=\\*\\.\\?;,+\\/:&_ ]*$
Validator.HTTPContextPath=^\\/?[a-zA-Z0-9.\\-\\/_]*$
Validator.HTTPServletPath=^[a-zA-Z0-9.\\-\\/_]*$
Validator.HTTPPath=^[a-zA-Z0-9.\\-_]*$
Validator.HTTPQueryString=^[a-zA-Z0-9()\\-=\\*\\.\\?;,+\\/:&_ %]*$
Validator.HTTPURI=^[a-zA-Z0-9()\\-=\\*\\.\\?;,+\\/:&_ ]*$
Validator.HTTPURL=^.*$
Validator.HTTPJSESSIONID=^[A-Z0-9]{10,30}$
# Validation of file related input
Validator.FileName=^[a-zA-Z0-9!@#$%^&{}\\[\\]()_+\\-=,.~'` ]{1,255}$
Validator.DirectoryName=^[a-zA-Z0-9:/\\\\!@#$%^&{}\\[\\]()_+\\-=,.~'` ]{1,255}$
# Validation of dates. Controls whether or not 'lenient' dates are accepted.
# See DataFormat.setLenient(boolean flag) for further details.
Validator.AcceptLenientDates=false
```

Figure 5.7: ESAPI configuration file

All stages, except a second, are valid and do their job the right way. However, the ESAPI stage is flawed because of the regex used in the validation.

The author found one of the possible Path Traversal attacks because of the flawed sanitization. The attack itself will not work because of the bad inject place, but if it were injected at the beginning of a subdirectory, then an adversary would be able to execute a Path Traversal attack against Voting Server. The source code is shown in the following two figures.

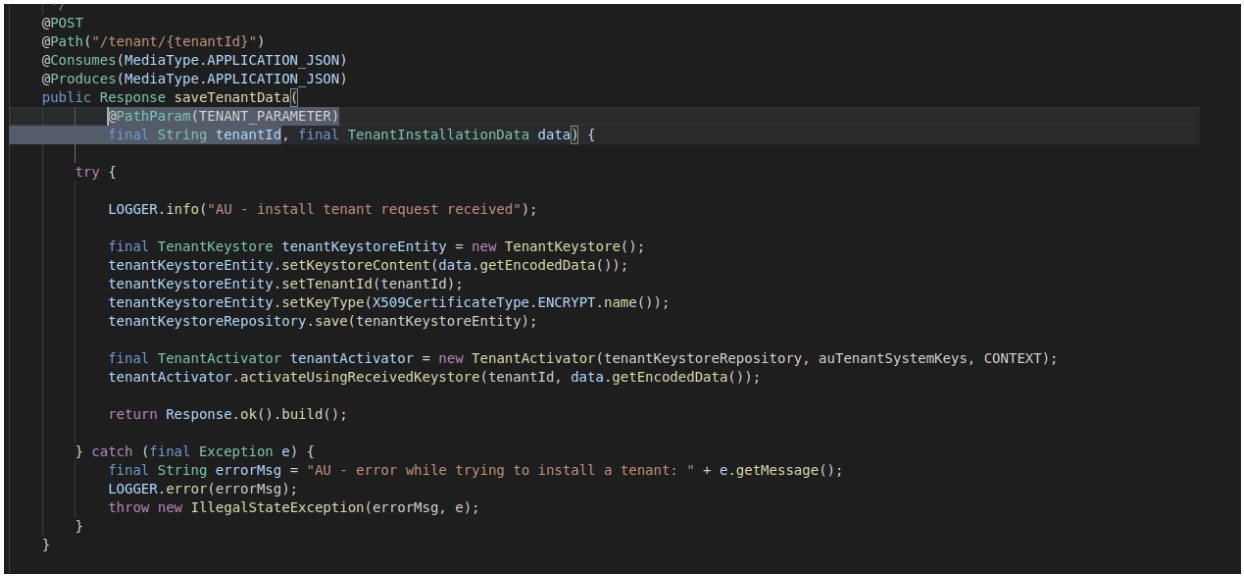


Figure 5.8: Lack of additional input sanitization in the following API mapping code

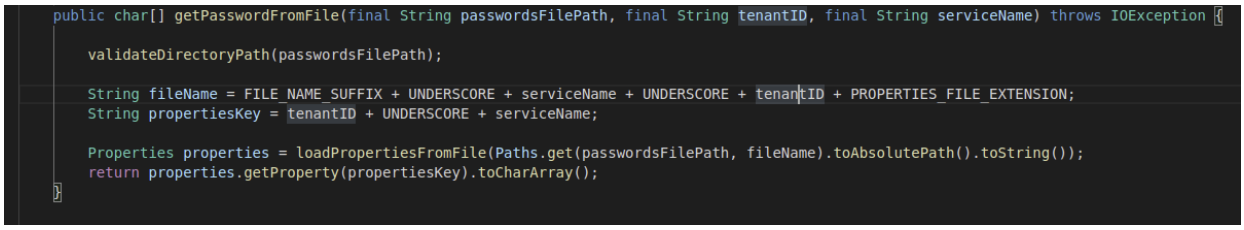


Figure 5.9: Usage of unsanitized variable in definition of path

In the first figure, the malicious path could be received as `tenantId` variable, which is then used in the 2nd figure as `tenandID` variable inside a path to a file. If a filename consisted only of a `tenantId` or it was in the beginning, then this endpoint would be vulnerable to Path Traversal.

As a result, Swiss Post does not have explicit checks for each type of variable used in the system. Instead, they rely on a single validation point for most of the variables, which allows a

more extensive set of characters than needed. As a result, it might lead to vulnerabilities in the future. The issue has been reported to Swiss Post, but there has been no reply so far.

5.4.3 Docker containers

E-Voting System uses Kubernetes to deploy parts of the system run in the Swiss Post facility. Such parts as Voting Server, portal, and control components are deployed as Docker containers for additional isolation in the context of security. To write directives for each docker container - Dockerfile files are used.

In general, Docker containers should be deployed from users with low privileges and to enforce that a `USER` directive is used because, by default, user containers are run as root. If an adversary gets a foothold in a container that is run as root, it can get root privileges on the host machine and easily compromise the host.[6][7]

The author found out that the voter portal container did not use `USER` directive and, in that way, was violating the least privilege principle. The Voter Portal is presumed to be untrustworthy except for voter secrecy[4]. Thus, votes' secrecy may be at stake if this issue on the Dockerfile leads to successful root access on the Voter Portal host server. Vulnerable Dockerfile could be found at this link[5]

The author reported a vulnerability to the Swiss Post. As expected, the issue was accepted and assigned a CVSS score of 3.3.

5.4.4 Message Broker communication

The voting Server Component uses MessageBroker to communicate with the Control Components. The same communication channel is used when Secure Data Manager needs to communicate with Control Components. The goal here is to verify whether an adversary with access to the Network can intercept and change the messages sent to SDM.

```
@Produces
@ApplicationScoped
public MessagingService messagingService() {
    return new MessagingServiceImpl.Builder().setHostName(System.getenv( name: "CC_MB_HOSTNAME")).setPort(Integer.parseInt(System.getenv( name: "CC_MB_PORT")))
            .setVirtualHost(System.getenv( name: "CC_MB_VHOST")).setUsername(System.getenv( name: "CC_MB_USER")).setPassword(System.getenv( name: "CC_MB_PASSWORD"))
            .setSenderPoolSize(Runtime.getRuntime().availableProcessors()).useSSL().build();
}
```

Figure 5.10: Connection creation to RabbitMQ

[5]https://gitlab.com/swisspost-evoting/e-voting/e-voting/-/blob/master/voter-portal/Dockerfile

The figure 5.10 shows the configuration of Message Broker to use SSL and the extraction of username and password from system variables.

During the analysis, the author found out that Message Broker is tightly coupled with Voting Server and might be seen as the same component as Voting Server, even though it is neither untrustworthy nor trustworthy according to the threat model.

A Voting Server has to have credentials to connect to RabbitMQ queues and send messages. The author checked whether these credentials are hard coded and found that they are not. The voting Server uses system variables to store the password and username to the Message Broker. This means that in case of a compromise of the Voting Server, an adversary will be able at least to connect and send messages to the queues for Control Components. In the worst case, an adversary will be able to control the whole RabbitMQ instance. It depends on the Voting Server user account's privileges in the RabbitMQ instance.

5.4.5 SDM communication with CCR and Voting Server

As Voting Server is an untrustworthy party and to have a secure channel between both components - SDM signs the payload sent to Control Components. The Control Component successfully verifies the payload and then sends a response to the Voting Server, which sends it back to Secure Data Manager. All the connections are sequential. The Secure Data Manager does not verify that the data is indeed from the Control Component and does not check for any signatures. Then the data is used to update files on Secure Data Manager, update VotingCards and perform other actions with data from an untrusted source.

In addition to the described channel, Secure Data Manager sends several requests for Voting Server and changes its state or overwrites files based on the responses without validating and sanitizing the data from an untrustworthy party.

The logic of connection to Voting Server is coded in `RestClientService.java` file, where a REST client can be created with and without `RestClientInterceptor` class. In case it is created with the `RestClientInterceptor` class, then all HTTP requests and responses will be intercepted by this class, and a custom logic will be applied. The custom logic is presented in `intercept(Chain chain)` function, shown in figure . The SDM performs the signing of the request after executing `chain.request()`, which means that the SDM intercepted a request. Then after signing and updating the request, SDM executes `chain.proceed(newRequest)`, which will send an updated request and get a response. The data returned by the response is not being validated, authenticated, or sanitized, and the signature is not being verified. In the case of not using the `RestClientInterceptor`, then the request is sent without signature as-is.

In the figure 5.11, a listing of `intercept(Chain chain)` is presented:

```
SwissPost-Bot
@Override
public okhttp3.Response intercept(Chain chain) throws IOException {

    Request originalRequest = chain.request();

    // Ask for a new signature
    String signature;
    try {
        signature = getEncodedSignatureString(originalRequest, nodeIdentifier);

    } catch (GeneralCryptoLibException e) {
        LOG.error("GeneralCryptoLibException");
        throw new IOException(e);
    }

    okhttp3.Response returnedResponse;

    Request.Builder requestBuilder = originalRequest.newBuilder().header(HEADER_SIGNATURE, signature)
            .header(HEADER_ORIGINATOR, nodeIdentifier);

    Request newRequest = requestBuilder.build();

    if (LOG.isDebugEnabled()) {
        LOG.debug("REQUEST HEADERS: {}", newRequest.headers());
    }

    returnedResponse = chain.proceed(newRequest);

    if (LOG.isDebugEnabled()) {
        LOG.debug("RESPONSE HEADERS: {}", returnedResponse.headers());
    }
    return returnedResponse;
}
```

Figure 5.11: intercept function of RestClientInterceptor class

BallotBoxDownloadService

In this service, SDM uses `orchestratorRestAdapter` variable, which uses `Interceptor` class. An adversary can influence the output of the following two functions: `getInitialPayloads` and `getMixingPayloads`. And as a result can influence `downloadPayloads` function, which influences a request to download a ballot box identified by the corresponding election event and its id as in the following path - `/electionevent/{electionEventId}/ballotbox/{ballotBoxId}`. Thus a malicious Voting Server can influence what the Secure Data Manager downloads for future decryption. It can duplicate ballot boxes and omit or corrupt them.

In the figures 5.12, 5.13 and 5.14, listings of `BallotBoxDownloadService`, `downloadPayloads`, and `getMixingPayloads` of BallotBoxDownloadService class are presented:

```java
@PostMapping(value = "/electionevent/{electionEventId}/ballotbox/{ballotBoxId}")
@ApiOperation(value = "Download ballot box", notes = "Service to download a given ballot box.")
@ApiResponses(value = { @ApiResponse(code = 403, message = "Forbidden") })
public ResponseEntity<IdleState> downloadBallotBox(
        @ApiParam(value = "String", required = true)
        @PathVariable
        final String electionEventId,
        @ApiParam(value = "String", required = true)
        @PathVariable
        final String ballotBoxId) {

    validateUUID(electionEventId);
    validateUUID(ballotBoxId);

    // checks firstly if the ballot box id is already doing an operation
    final IdleState idleState = new IdleState();
    if (idleStatusService.getIdLock(ballotBoxId)) {
        try {
            // Download the mixing payloads.
            ballotBoxDownloadService.downloadPayloads(electionEventId, ballotBoxId);
            // Download the raw ballot box.
            ballotBoxDownloadService.download(electionEventId, ballotBoxId);
            // Mark the ballot box as downloaded.
            ballotBoxDownloadService.updateBallotBoxStatus(ballotBoxId);
            return new ResponseEntity<>(idleState, HttpStatus.OK);
        } catch (final BallotBoxDownloadException e) {
            LOGGER.error("Error downloading ballot box ", e);
            return new ResponseEntity<>(idleState, HttpStatus.FORBIDDEN);
        } finally {
            idleStatusService.freeIdLock(ballotBoxId);
        }
    } else {
        idleState.setIdle(true);
        return new ResponseEntity<>(idleState, HttpStatus.OK);
    }
}
```

Figure 5.12: BallotBoxDownloadService listing

```java
4 usages  SwissPost-Bot
public void downloadPayloads(final String electionEventId, final String ballotBoxId) throws BallotBoxDownloadException {
    checkNotNull(electionEventId);
    checkNotNull(ballotBoxId);
    validateUUID(electionEventId);
    validateUUID(ballotBoxId);

    LOGGER.info("Requesting payloads for ballot box {}...", ballotBoxId);
    final JsonNode ballotBox = getBallotBoxJson(ballotBoxId, electionEventId);

    // Check that the full ballot box has been mixed.
    checkBallotBoxStatusForDownload(ballotBox, ballotBoxId);
    final String ballotId = ballotBox.get(JsonConstants.BALLOT).get(JsonConstants.ID).textValue();

    // Get payloads, then persist them
    final ResponseBody initialPayload = getInitialPayloads(electionEventId, ballotBoxId);
    persistBallotBoxInitialPayload(electionEventId, ballotId, ballotBoxId, initialPayload);

    final ResponseBody shufflePayloads = getMixingPayloads(electionEventId, ballotBoxId);
    persistBallotBoxShufflePayloads(electionEventId, ballotId, ballotBoxId, shufflePayloads);
}
```

Figure 5.13: downloadPayloads function of BallotBoxDownloadService class

Figure 5.14: getMixingPayloads of BallotBoxDownloadService class

BallotBoxMixingService

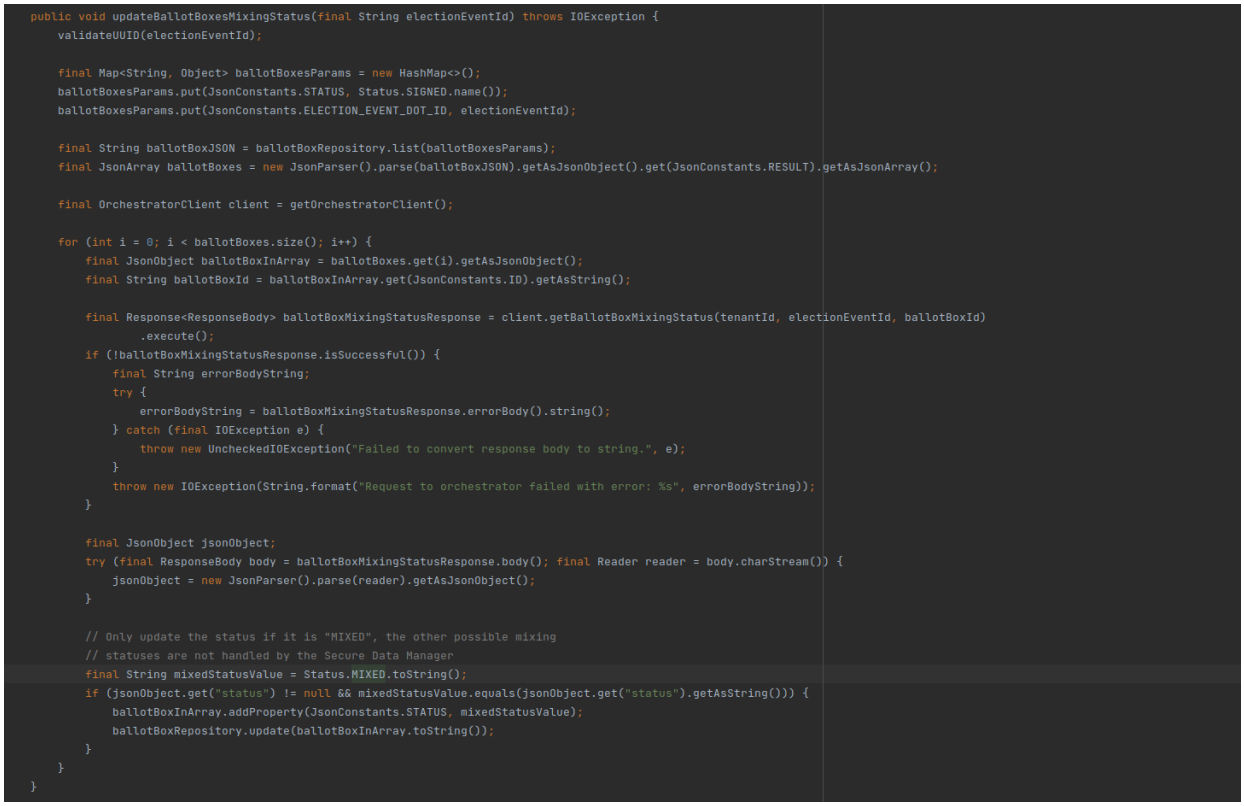
An adversary can respond to the following functions in this service: `mixBallotBoxes` and `updateBallotBoxesMixingStatus`. An attacker can trick Secure Data Manager into thinking that Ballot was sent to the mixing while it has not been in `mixBallotBoxes` function and then trick Secure Data Manager into thinking that a specific ballot was mixed by responding with malicious data to `updateBallotBoxesMixingStatus`, where JSON object will have a status parameter. It will be equal to `MIXED` or `"14"`.

In the figures 5.15 and 5.16, listings of `mixBallotBoxes` and

`updateBallotBoxesMixingStatus` functions of `BallotBoxMixingService` class are presented:

```java
public String mixBallotBoxes(final String electionEventId, final List<String> ballotBoxIds) {
    validateUUID(electionEventId);
    checkNotNull(ballotBoxIds);
    checkArgument(!ballotBoxIds.isEmpty(), "Ballot box ids to mix must not be empty.");
    ballotBoxIds.forEach(UUIDValidations::validateUUID);

    final Response<ResponseBody> response;
    try {
        response = getOrchestratorClient().mixBallotBoxes(tenantId, electionEventId, ballotBoxIds).execute();
    } catch (final IOException e) {
        throw new UncheckedIOException("Failed to communicate with orchestrator.", e);
    }

    if (!response.isSuccessful()) {
        final String errorBodyString;
        try {
            errorBodyString = response.errorBody().string();
        } catch (final IOException e) {
            throw new UncheckedIOException("Failed to convert response body to string.", e);
        }
        final String errormessage = String
                .format("Failed to mix ballot boxes [electionEvent=%s, ballotBoxes=%s]: %s", electionEventId, ballotBoxIds, errorBodyString);
        throw new BallotBoxServiceException(errormessage);
    }

    try {
        return response.body().string();
    } catch (final IOException e) {
        throw new UncheckedIOException("Failed to convert response body to string.", e);
    }
}
```

Figure 5.15: mixBallotBoxes function of BallotBoxMixingService class



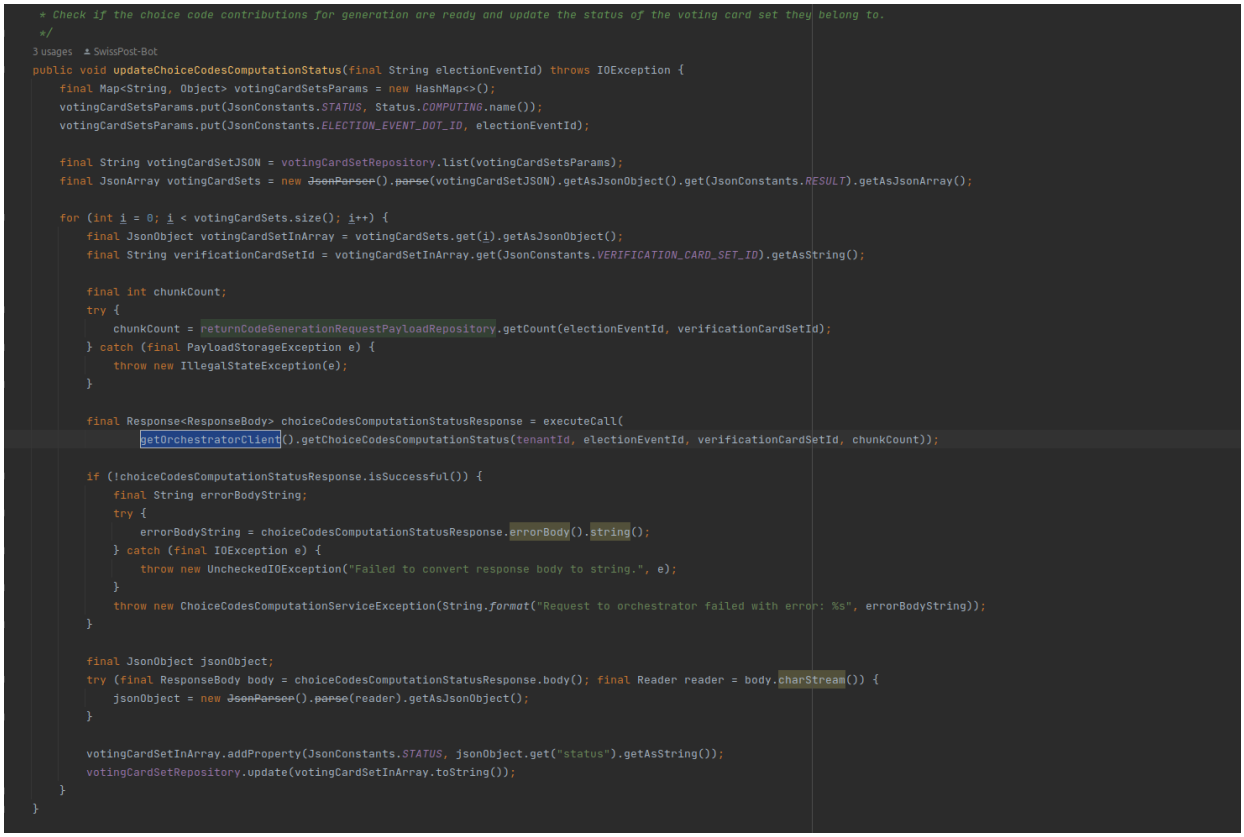
```
public void updateBallotBoxesMixingStatus(final String electionEventId) throws IOException {
    validateUUID(electionEventId);

    final Map<String, Object> ballotBoxesParams = new HashMap<>();
    ballotBoxesParams.put(JsonConstants.STATUS, Status.SIGNED.name());
    ballotBoxesParams.put(JsonConstants.ELECTION_EVENT_DOT_ID, electionEventId);

    final String ballotBoxJSON = ballotBoxRepository.list(ballotBoxesParams);
    final JsonArray ballotBoxes = new JsonParser().parse(ballotBoxJSON).getAsJsonObject().get(JsonConstants.RESULT).getAsJsonArray();

    final OrchestratorClient client = getOrchestratorClient();

    for (int i = 0; i < ballotBoxes.size(); i++) {
        final JsonObject ballotBoxInArray = ballotBoxes.get(i).getAsJsonObject();
        final String ballotBoxId = ballotBoxInArray.get(JsonConstants.ID).getAsString();

        final Response<ResponseBody> ballotBoxMixingStatusResponse = client.getBallotBoxMixingStatus(tenantId, electionEventId, ballotBoxId)
                .execute();
        if (!ballotBoxMixingStatusResponse.isSuccessful()) {
            final String errorBodyString;
            try {
                errorBodyString = ballotBoxMixingStatusResponse.errorBody().string();
            } catch (final IOException e) {
                throw new UncheckedIOException("Failed to convert response body to string.", e);
            }
            throw new IOException(String.format("Request to orchestrator failed with error: %s", errorBodyString));
        }

        final JsonObject jsonObject;
        try (final ResponseBody body = ballotBoxMixingStatusResponse.body(); final Reader reader = body.charStream()) {
            jsonObject = new JsonParser().parse(reader).getAsJsonObject();
        }

        // Only update the status if it is "MIXED", the other possible mixing
        // statuses are not handled by the Secure Data Manager
        final String mixedStatusValue = Status.MIXED.toString();
        if (jsonObject.get("status") != null && mixedStatusValue.equals(jsonObject.get("status").getAsString())) {
            ballotBoxInArray.addProperty(JsonConstants.STATUS, mixedStatusValue);
            ballotBoxRepository.update(ballotBoxInArray.toString());
        }
    }
}
```

Figure 5.16: updateBallotBoxesMixingStatus function of BallotBoxMixingService class

ChoiceCodesComputationService

In this service, the `Interceptor` class is not used, but the request is sent to the Voting Server, and a response is not validated but is used to set the status of the Voting Card. The value can be arbitrary, but the most promising is setting status to `"1"`, which means `LOCKED`.

In the figure 5.17, a listing of `updateChoiceCodesComputationStatus` function of class `ChoiceCodesCOmputationService` is presented:



```
 * Check if the choice code contributions for generation are ready and update the status of the voting card set they belong to.
 */
3 usages   SwissPost-Bot
public void updateChoiceCodesComputationStatus(final String electionEventId) throws IOException {
    final Map<String, Object> votingCardSetsParams = new HashMap<>();
    votingCardSetsParams.put(JsonConstants.STATUS, Status.COMPUTING.name());
    votingCardSetsParams.put(JsonConstants.ELECTION_EVENT_DOT_ID, electionEventId);

    final String votingCardSetJSON = votingCardSetRepository.list(votingCardSetsParams);
    final JsonArray votingCardSets = new JsonParser().parse(votingCardSetJSON).getAsJsonObject().get(JsonConstants.RESULT).getAsJsonArray();

    for (int i = 0; i < votingCardSets.size(); i++) {
        final JsonObject votingCardSetInArray = votingCardSets.get(i).getAsJsonObject();
        final String verificationCardSetId = votingCardSetInArray.get(JsonConstants.VERIFICATION_CARD_SET_ID).getAsString();

        final int chunkCount;
        try {
            chunkCount = returnCodeGenerationRequestPayloadRepository.getCount(electionEventId, verificationCardSetId);
        } catch (final PayloadStorageException e) {
            throw new IllegalStateException(e);
        }

        final Response<ResponseBody> choiceCodesComputationStatusResponse = executeCall(
                getOrchestratorClient().getChoiceCodesComputationStatus(tenantId, electionEventId, verificationCardSetId, chunkCount));

        if (!choiceCodesComputationStatusResponse.isSuccessful()) {
            final String errorBodyString;
            try {
                errorBodyString = choiceCodesComputationStatusResponse.errorBody().string();
            } catch (final IOException e) {
                throw new UncheckedIOException("Failed to convert response body to string.", e);
            }
            throw new ChoiceCodesComputationServiceException(String.format("Request to orchestrator failed with error: %s", errorBodyString));
        }

        final JsonObject jsonObject;
        try (final ResponseBody body = choiceCodesComputationStatusResponse.body(); final Reader reader = body.charStream()) {
            jsonObject = new JsonParser().parse(reader).getAsJsonObject();
        }

        votingCardSetInArray.addProperty(JsonConstants.STATUS, jsonObject.get("status").getAsString());
        votingCardSetRepository.update(votingCardSetInArray.toString());
    }
}
```

Figure 5.17: updateChoiceCodesComputationStatus function of ChoiceCodesCOmputationService class

VotingCardSetChoiceCodesService

Same as in `ChoiceCodesComputationService` Secure Data Manager does not validate the received output from Voting Server, and an adversary can overwrite `NodeContribution` files with arbitrary data. It could be the copy of the first valid `NodeContribution`, which will be copied into all `NodeContribution` files.

In the figures 5.18, 5.19 and 5.20, listings of `download` function of `VotingCardSetChoiceCodesService` class, `download` and `writeNodeContributions` functions of `VotingCardSetDownloadService` class are presented:

```java
public void download(final String votingCardSetId, final String electionEventId)
        throws ResourceNotFoundException, InvalidStatusTransitionException, IOException {

    LOGGER.info("Downloading the computed values. [electionEventId: {}, votingCardSetId: {}]", electionEventId, votingCardSetId);

    if (!idleStatusService.getIdLock(votingCardSetId)) {
        return;
    }

    try {
        final Status fromStatus = Status.COMPUTED;
        final Status toStatus = Status.VCS_DOWNLOADED;

        checkVotingCardSetStatusTransition(electionEventId, votingCardSetId, fromStatus, toStatus);

        final JsonObject votingCardSetJson = votingCardSetRepository.getVotingCardSetJson(electionEventId, votingCardSetId);
        final String verificationCardSetId = votingCardSetJson.getString(JsonConstants.VERIFICATION_CARD_SET_ID);

        deleteNodeContributions(electionEventId, verificationCardSetId);

        final int chunkCount;
        try {
            chunkCount = returnCodeGenerationRequestPayloadRepository.getCount(electionEventId, verificationCardSetId);
        } catch (final PayloadStorageException e) {
            throw new IllegalStateException("Failed to get the chunk count.", e);
        }

        for (int i = 0; i < chunkCount; i++) {
            try (final InputStream contributions = votingCardSetChoiceCodesService.download(electionEventId, verificationCardSetId, i)) {
                writeNodeContributions(electionEventId, verificationCardSetId, i, contributions);
            }
        }

        configurationEntityStatusService.update(toStatus.name(), votingCardSetId, votingCardSetRepository);

    } finally {
        idleStatusService.freeIdLock(votingCardSetId);
    }

}
```

Figure 5.18: download function of VotingCardSetChoiceCodesService class

```java
1 usage  SwissPost-Bot
private void writeNodeContributions(final String electionEventId, final String verificationCardSetId, final int chunkId,
        final InputStream contributions)
        throws IOException {
    final String fileName = Constants.CONFIG_FILE_NAME_NODE_CONTRIBUTIONS + "." + chunkId + Constants.JSON;
    final Path file = pathResolver.resolve(Constants.CONFIG_FILES_BASE_DIR).resolve(electionEventId).resolve(Constants.CONFIG_DIR_NAME_ONLINE)
            .resolve(Constants.CONFIG_DIR_NAME_VOTEVERIFICATION).resolve(verificationCardSetId).resolve(fileName);

    try (final OutputStream stream = newOutputStream(file)) {
        IOUtils.copy(contributions, stream);
    }
}
```

Figure 5.19: download function of VotingCardSetDownloadService class

Figure 5.20: writeNodeContributions function of VotingCardSetDownloadService class

BallotBoxDownloadService

The SDM uses a direct channel to the Voting Server in this service. `Interceptor` class is used, so the data sent to Voting Server is signed, but as shown before, the response signature is absent and not verified. However, as it is a communication with an untrusted component, the response should be verified and validated, but SDM does not perform that. This enables an adversary to overwrite `BallotBox` file with a copy of an interested `BallotBox`, arbitrary data, or corrupt the `BallotBox`.

In the figures 5.21, 5.22 and 5.23, listings of `download`, `downloadBallotBox` and `writeBallotBoxStreamToFile` functions of `BallotBoxDownloadService` class are presented:

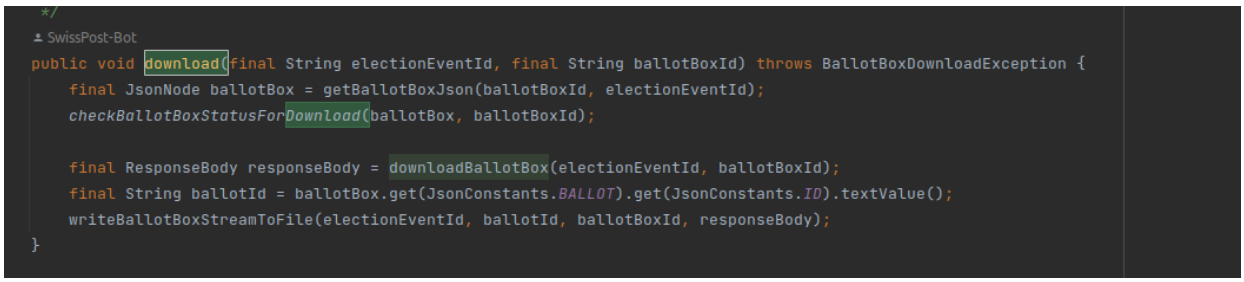


Figure 5.21: download function of BallotBoxDownloadService class

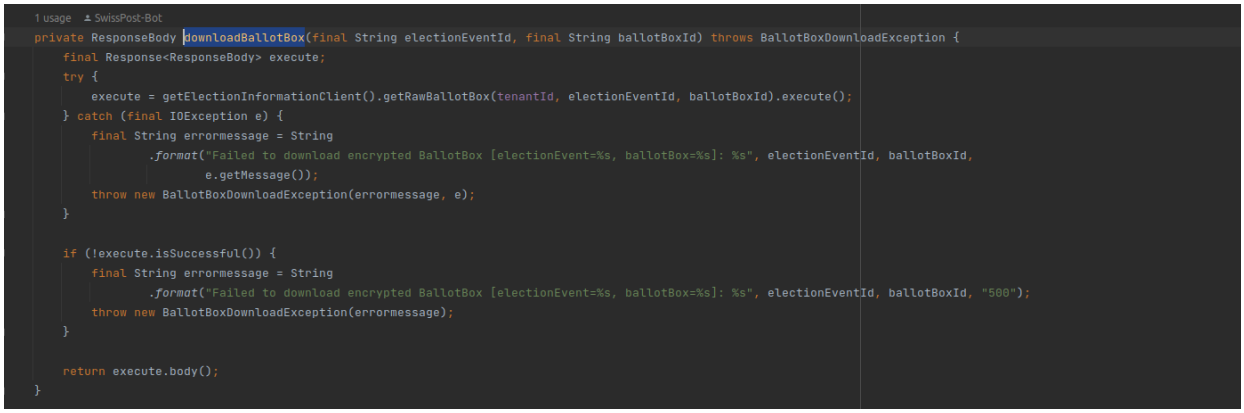


Figure 5.22: downloadBallotBox function of BallotBoxDownloadService class

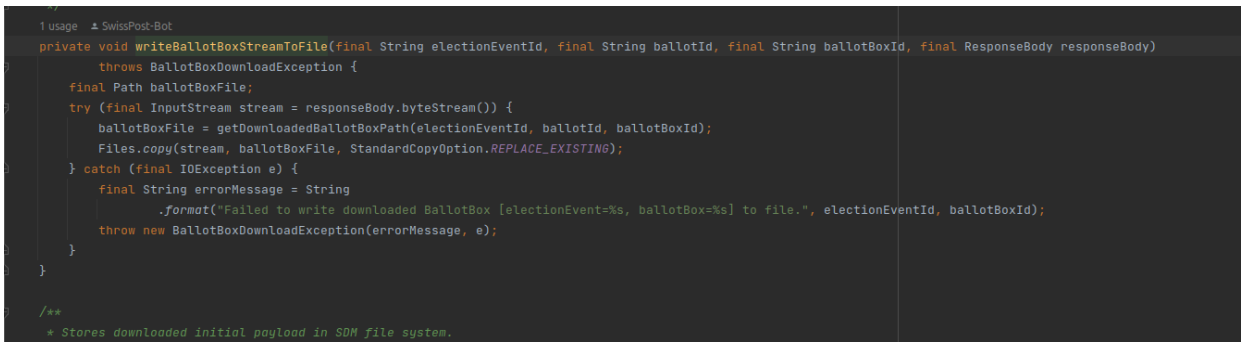


Figure 5.23: writeBallotBoxStreamToFile function of BallotBoxDownloadService class

5.4.6 Remote Code Execution in SDM

The endpoints `/generate-pre-voting-outputs/{electionEventId}` and `/generate-post-voting-outputs/{electionEventId}/{ballotBoxStatus}` in export operation service do not verify a body parameter `privateKeyInBase64` received from front-end, which then is being executed in `Runtime.getRuntime().exec()` as part of the command. To achieve the full chain from sink to source `electionEventId` should exist, and in case of a second request, `ballotBoxStatus` should be one of the predefined statuses.

The second way to exploit the same issue is through the `electionEventAlias`, but it has a higher cost because an administrator would need to create an `electionEventId` with a malicious alias. Only an administrator could exploit this bug because online SDM is isolated from incoming traffic.

The main discovered problem is that the input is assumed to be base64 encoded while it

is not. Furthermore, the `ESAPI` and other forms of validation are not used because Swiss Post believes that a malicious administrator can temper files and execute code on the machine even without such vulnerabilities. However, the diagram from the Architecture document[3] suggests that an administrator can interact only with the front-end of the system but not with other components.

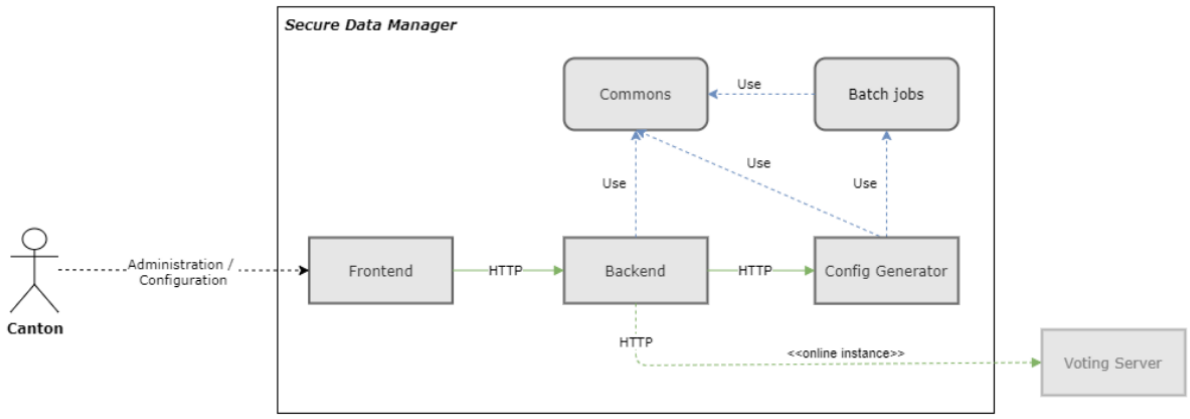


Figure 5.24: Secure Data Manager Level 2 architecture view[3]

The author provided an additional Proof of Concept to Swiss Post and asked the team to clarify questions, such as whether or not all SDM parts run on a single machine. A malicious administrator can temper with the whole part of the system without exploiting any vulnerabilities without doing any privilege escalation and can pivot to the user running the back-end of SDM. Another question was about the trustworthiness of an administrator who is untrustworthy with handling a USB drive but is trustworthy with operating SDM. Also, the author asked about the input validation because the author found none in the whole component.

The previous questions were answered in detail. From the question, the author can summarize that the online SDM components run on one machine with communication using plain HTTP between the components. That means that a malicious Administrator can temper all online SDM components even though only the front-end system should be accessible, according to the diagram. Additionally, an administrator is trustworthy according to the following answer from Swiss Post “We consider the setup component trustworthy throughout the whole election (end even after election period ended). However, the setup component only plays an active role during the configuration phase and is no longer needed in the voting and the tally phase”.

In addition to the general questions, the author raised a question about a possible design problem, where an election administrator might have more privileges than he should. According to the Swiss Post, a malicious administrator can temper all files of a component, which might be avoided by following the “Least privilege principle”. In theory, an administrator should only

access a GUI of the front-end component in the sand-boxed environment and nothing else except what is needed for election setup. The election administrator's user should have the least privileges in the system. At the same time, the back-end components should run in another environment and validate the data sent from the front-end component.

As of the 8th of June 2022, the report is accepted, and Swiss Post will investigate whether they will add more checks before using input data or not. The report CVSS score is 3.9 due to a downgrade from 9.0. Swiss Post is reevaluating this report because the author sent an additional proof of concept and a much more detailed explanation of the issue.

Exploitation

In this subchapter, the author presents a trace from source to sink on `/generate-pre-voting-outputs/{electionEventId}` path. The second endpoint exploitation would be the same, just that it would require having a working ballotBoxStatus.

```
97    :OC   -> getting ElectionEventId
99    :OC   -> getting request value
101   :OC  -> validateUUID() check if UUID exists and follows the format
103   :OC  -> check if `request.getPrivateKeyInBase64()` is not empty and that's all (This is a problem!)
107   :OC  -> call to `buildParameters(electionEventId, request.getPrivateKeyInBase64(), "")`
356   :OC  -> get Event Alias
166   :EES -> get Event Alias
52    :EER -> get Event Alias
59,63 :EER -> No verification of the value from database! (Also might be a problem)
363   :EES -> Adds static path to parameters
365   :EES -> Check if key is not empty (no check for bad characters, thinks it's validated and base64 encoded)
366   :EES -> Adds key to parameters!
368   :EES -> Adds path to parameters
108   :OC  -> executeOperationForPhase(parameters, phasename, true)
326   :OC  -> get commands for phase (It's not empty)
328   :OC  -> skips check as the list is not empty
333   :OC  -> sequentialExecutor.execute(commandsForPhase, parameters, listener)
32    :SE  -> replaceParameters(command, parameters)
116   :SE  -> replacedCommand.replaceAll("#"+key+"#", value). We are interested in this one only as it is executed
126   :SE  -> returns replacedCommand, partialCommand
33    :SE  -> assigns replacedCommand to fullCommand
40    :SE  -> gets executed in `Runtime.getRuntime().exec()`
```

Figure 5.25: RCE execution trace

The list of affected files is following:

1. `OperationsController.java\verb` -> OC

2. `ElectionEventService.java` -> EES
3. `ElectionEventRepository.java` -> EER
4. `SequentialExecutor.java` -> SE

The trace of the calls is presented in the figure 5.25, then the result of escaping from the quotes by sending the malicious input is shown in the figure 5.26. After that, screenshots for the trace are presented in figures 5.27, 5.28, 5.29, 5.30, 5.31, 5.31, 5.32 and 5.33. Finally, the trace is shown in line, Filename, explanation, or action format.

```
1
[Ljava.lang.String;@182decdb
java hhh token aaa
------
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.sun.xml.bind.v2.runtime.reflect.opt.Injector (file:/home/weil/.m2/repository/com/sun/xml/bind/jaxb-impl/2.2.11
WARNING: Please consider reporting this to the maintainers of com.sun.xml.bind.v2.runtime.reflect.opt.Injector
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
2022-05-21 02:20:47 [main] ERROR c.p.i.e.s.i.p.SequentialExecutor:59 - Error '100' when executing command:
           java -jar some/sdmpath/bla/bla/integration/file_converter_tool/file-converter.jar
                   -action election_event_alias_to_id
                   -election_export_in some/sdmpath/bla/bla/sdmConfig/elections_config.json
                   -output some/sdmpath/bla/bla/config/1242142142/CUSTOMER/output/election_event_alias_to_id_EVENTALIAS.csv
                   -log some/sdmpath/bla/bla/config/1242142142/CUSTOMER/logs/file_converter_EVENTALIAS.log
                   -privateKey "PRIVATE_KEY"
        : java.io.IOException: Cannot run program "
           java -jar some/sdmpath/bla/bla/integration/file_converter_tool/file-converter.jar
                   -action election_event_alias_to_id
                   -election_export_in some/sdmpath/bla/bla/sdmConfig/elections_config.json
                   -output some/sdmpath/bla/bla/config/1242142142/CUSTOMER/output/election_event_alias_to_id_EVENTALIAS.csv
                   -log some/sdmpath/bla/bla/config/1242142142/CUSTOMER/logs/file_converter_EVENTALIAS.log
                   -privateKey "YmFzZTY0ZW5jb2RlZGRhdGEK"||do_something_else ""
       ": error=2, No such file or directory
2022-05-21 02:20:47 [main] ERROR c.p.i.e.s.i.p.SequentialExecutor:59 - Error '100' when executing command:
           java -jar some/sdmpath/bla/bla/integration/file_converter_tool/file-converter.jar
                   -action data_config_pn
                   -election_export_in some/sdmpath/bla/bla/sdmConfig/elections_config.json
                   -dataConfig some/sdmpath/bla/bla/config/1242142142/CUSTOMER/output/dataConfig_EVENTALIAS.json
                   -output some/sdmpath/bla/bla/config/1242142142/CUSTOMER/output/dataConfig_updated_EVENTALIAS.json
                   -log some/sdmpath/bla/bla/config/1242142142/CUSTOMER/logs/file_converter_EVENTALIAS.log
```

Figure 5.26: Proof of Concept of RCE in SDM

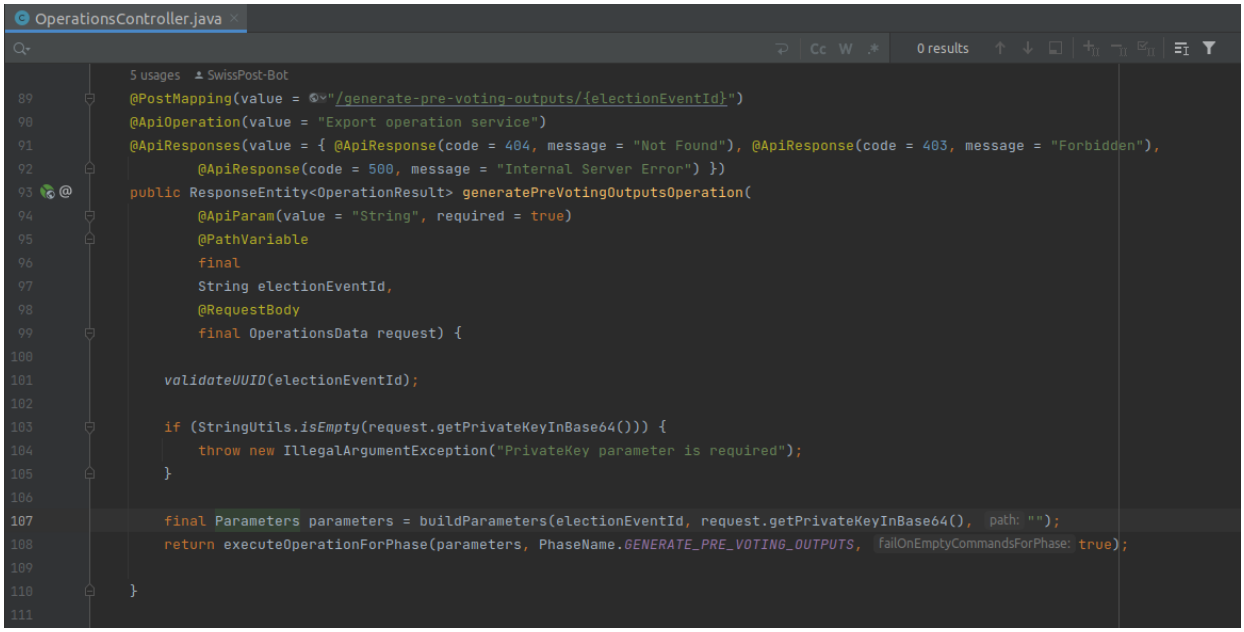


Figure 5.27: Lack of input sanitization in API mapping code

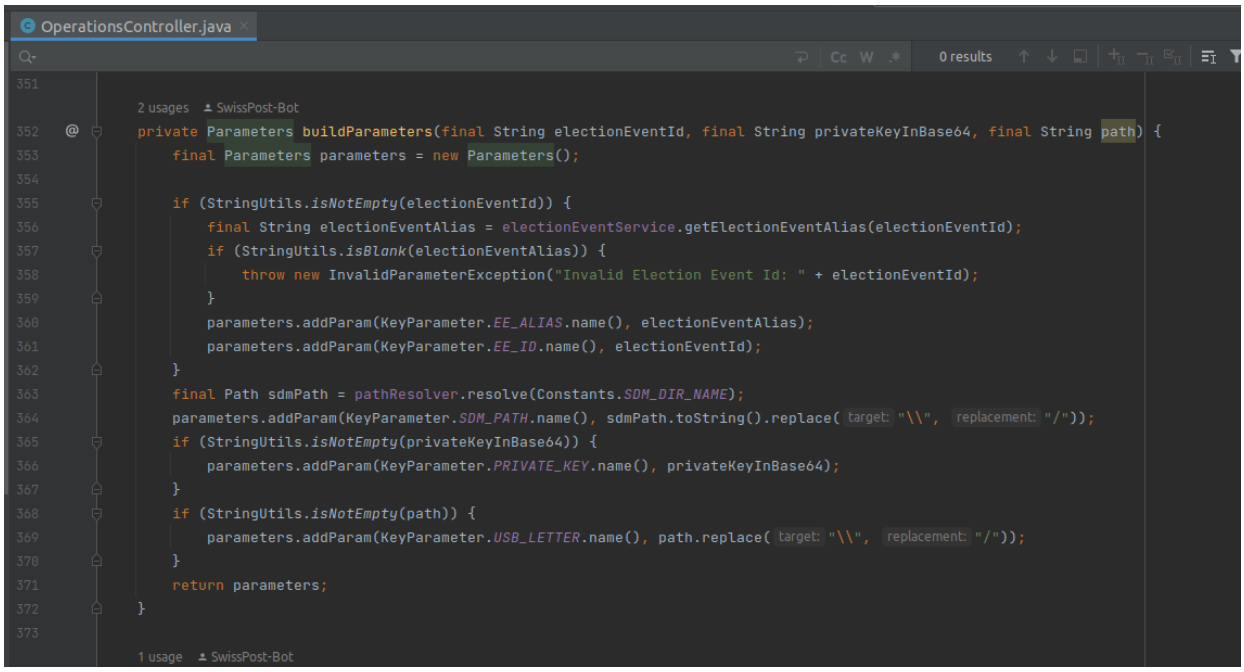


Figure 5.28: buildParameters function listing

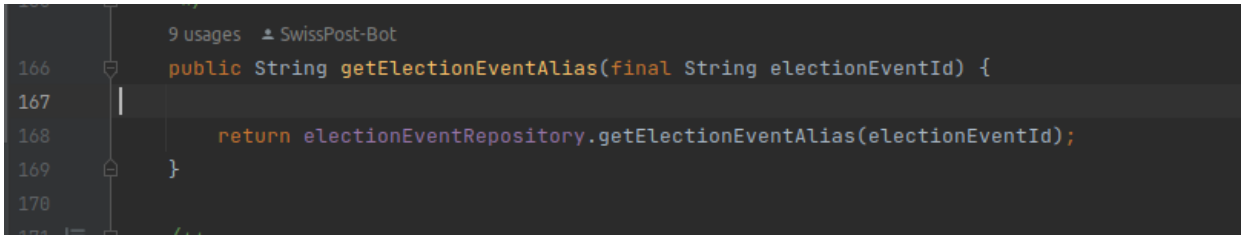


Figure 5.29: getElectionEventAlias function listing

Figure 5.30: getElectionEventAlias function of ElectionEventRepository class listing

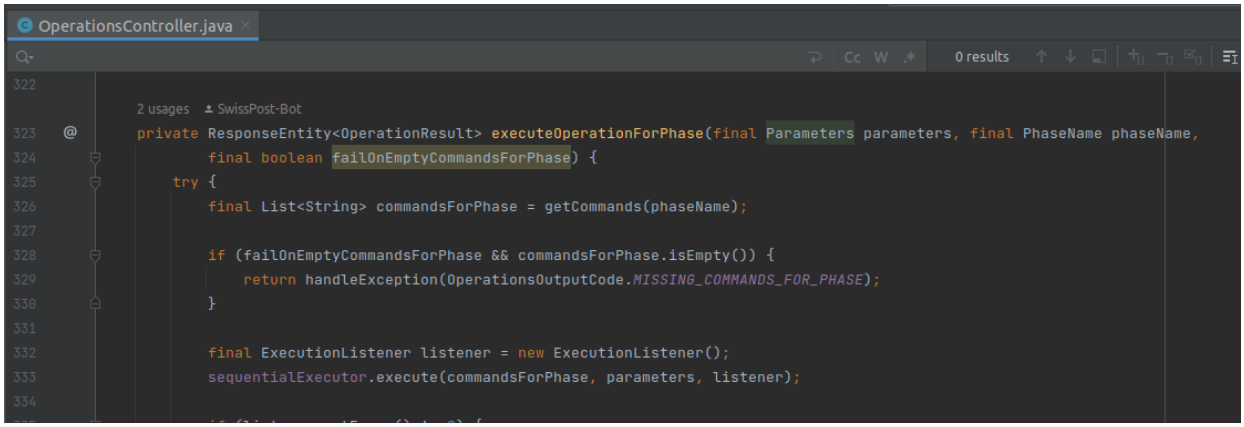


Figure 5.31: executeOperationsForPhase function listing

SequentialExecutor.java

```
SwissPost-Bot
public void execute(final List<String> commands, final Parameters parameters, final ExecutionListener listener) {

    for (final String command : commands) {
        String mockCommand = command;
        try {
            // Replace the parameters.
            final String[] partialCommands = replaceParameters(command, parameters);
            final String fullCommand = partialCommands[0];
            mockCommand = partialCommands[1];

            // Remove unwanted environment variable that will be inherited by the child process.
            final String[] envp = buildCommandEnv();

            // Execute the command.
            final Process proc = Runtime.getRuntime().exec(new StringTokenizer(fullCommand, delim: " \t\n\r\f").getTokenArray(), envp);
```

Figure 5.32: execute function listing

SequentialExecutor.java

```
                listener.onProgress(line);
            }
        }
        LOGGER.debug("<<----- Plugin Output");
    } catch (final IOException e) {
        LOGGER.warn("Failed to read plugin execution output", e);
    }
}

1 usage  SwissPost-Bot
private String[] replaceParameters(final String command, final Parameters parameters) {
    String partialCommand = command;
    String replacedCommand = command;

    for (final KeyParameter key : KeyParameter.values()) {
        if (replacedCommand.contains(key.toString())) {
            final String value = parameters.getParam(key.name());
            if (value == null || value.isEmpty()) {
                throw new IllegalArgumentException("Parameter #" + key.name() + "# is null or empty");
            } else {
                replacedCommand = replacedCommand.replaceAll( regex: "#" + key + "#", value);
                if (key == KeyParameter.PRIVATE_KEY) {
                    partialCommand = partialCommand.replaceAll( regex: "#" + key + "#", replacement: "PRIVATE_KEY");
                } else {
                    partialCommand = partialCommand.replaceAll( regex: "#" + key + "#", value);
                }
            }
        }
    }

    return new String[] { replacedCommand, partialCommand };
}
```

Figure 5.33: replaceParameters function listing

5.5 Adversarial Scenarios

In this section, the author assumes that a certain component or subcomponent is compromised, and then the author determines how an adversary can perform lateral movements. The assumptions of compromised components are made based on the threat model of Swiss Post E-Voting. In contrast, the assumptions of further actions are backed by findings in this report, findings of other researchers on the E-Voting Bug Bounty Program, findings on other products from personal experience, and findings of other people published on the Internet.

The section is mainly theoretical, but not purely, because of the facts that the author can provide. Also, some of these assumptions lead to findings.

5.5.1 Malicious Voting Server Service

In this section, the author assumes that one of the services of the Voting Server is malicious.

This assumption could be made since Voting Server consists of 10 different services, and according to the threat, model[4] Voting Server is untrustworthy. Additionally, all subcomponents of an untrustworthy component are also untrustworthy. That is why the author can make such an assumption.

Attack prerequisites

An adversary can get into the service by executing the following attacks:

1. Exploit a vulnerability in the input of the service sent from attacker-controlled Voter Portal
2. Exploit a vulnerability in the input sent from the voter that was processed by Voter Portal and sent further to one of the services of Voting Server

Environment

After successfully exploiting the service, an adversary would find itself in a docker container managed by Kubernetes. Communication channels will vary depending on the service an adversary has access to. Still, in general, an adversary has access only to the traffic sent through the service and a host of the container, which is a Worker Node.

Possible further attacks

A malicious service can try to change, record, and delete data sent through this component. It is a minimum of what a malicious service can do, and it will introduce availability-related issues, but only partially.

A better attack scenario would be escaping the container and then escalating its privileges to the root of the Voting Server. It could be achieved by following two attack scenarios:

1. Misconfiguration in Dockerfile as shown in the findings in section 5.4.3, which will give an adversary root privileges at once.
2. 0-day exploit acquired or developed by the adversary. The possible costs would be up to 200,000$ according to Zerodium Payouts[6].

Impact

1. Gain privileges over the whole Voting Server component
2. Being able to change, record and delete data sent through the Voting Server
3. Cause availability related problems

The finding in this report created a precedent where the Swiss Post E-voting team could make another mistake and allow an adversary to break out of the container without any effort. Overall, this finding just lowered the bar for an adversary, and a highly sophisticated adversary with unlimited resources such as nation-state hacker groups would be able to either purchase a 0-day exploit or develop it.

5.5.2 Malicious Voting Server

In this section, the author assumes that the whole Voting Server is malicious. The author can make such an assumption because Voting Server is untrustworthy.

Attack prerequisites

An adversary can get into the Voting Server by executing the following attacks:

[6] https://zerodium.com/program.html

1. Perform attacks as described in Possible further attacks in Malicious Voting Server Service section 5.5.1
2. Perform attacks as described in Attack prerequisites in Malicious Voting Server Service section 5.5.1
3. Exploit a vulnerability in the component that accepts input from a Message Broker, which the malicious Control Component could use.
4. Exploit a vulnerability in the component that accepts input from Secure Data Manager operated by a malicious administrator or use ssh keys or other means and plant malware with the help of a malicious administrator

Environment

After successful exploitation of the Voting Server, an adversary will find itself with full access to a Worker Node. An adversary has direct access to Message Broker and Voter Portal and indirect access to SDM, Control Components, and voter.

Possible further attacks

A malicious Voting Server can try to change, record, replay and delete data sent through it, thus causing minimum availability-related issues. In addition, an adversary can try to inject malicious data sent to or sent back to SDM, Control Components, or Voter Portal. As was shown in the Data Flow chapter 5.4, an adversary can successfully forge messages sent back to SDM and that way change the state of the SDM without changing the state of other parts of the system or corrupt votes. Also, it was possible to forge signed messages to Control Components, as was shown by `@reversemode` [8][9]. It is possible to send malicious javascript responses to the Voting Portal that will serve them according to the paper written by Network Security Group of ETH Zurich[10]. In addition to that, Voting Server has direct access to the Message Broker as the credentials are stored in the system variables. That way, anyone controlling the server can try to control a Message Broker. The outcome depends on the credentials that are stored on the server.

It is currently unknown how Voter Portal JS Response Check is implemented. That is why the author assumes that an adversary can bypass it and an adversary can break vote secrecy on the voter end, as noted by the Network Security Group of ETH Zurich.

1. Forge malicious responses to SDM, try to change the state of the system, or lock the votes.
2. Forge malicious responses to Voter Portal and try to pivot to the component
3. Forge malicious javascript responses to Voter Portal that will serve these files to voters and thus break vote secrecy

4. Forge malicious requests to Control Components by forging the signature.
5. Take the control over Message broker with the help of stored credentials
6. Sabotage requests from SDM to Control Components, by tricking the SDM into thinking that request has been passed over to Control Component, while in reality, it has not been
7. Drop voter votes from being confirmed and then used in the tally phase.

Impact

1. Vote Secrecy is broken when a Voting Server serves malicious JS files back to Voter Portal.
2. Availability is affected by poisoning the link to the Control Components and making a Message Broker unavailable even on restart.
3. Availability is affected when Voting Server can trick the SDM into changing the state of voting card sets or ballots.
4. Availability is affected when Voting Server is dropping either selectively or not the votes so that they are not taken into account during the tally phase.
5. Confidentiality, Integrity, and Availability are affected when Voting Server can directly pivot into Message Broker. The outcome depends on the credentials stored.

Overall having access to the Voting Server gives many routes to follow, and all of them are flawed, as shown by this research and research from the Network Security Group of ETH Zurich and @reversemode. Some of these flaws could have been avoided by introducing another gateway specifically for SDM and Control Component communication. Other components lack validation and sanitization, and Control Components checked the signatures incorrectly. The impact of breaking the Voting Server is significant now only because an untrustworthy component is a single point of failure for the communication between parts of the system.

5.5.3 Malicious Message Broker

In this section, the author assumes that a Message Broker is malicious. The author can make such an assumption even though it is unclear whether Message Broker is trustworthy or untrustworthy from the specification. The author can only suggest this because Message Broker is deployed in the Swiss Post network and belongs to Voting Server and Voter Portal components that are untrustworthy.

Attack prerequisites

An adversary can get into the Message Broker by executing the following attacks.

1. Exploit and gain control over Orchestrator service of Voting Server.
2. Exploit and gain control over Voting Server.
3. Exploit a known or 0-day vulnerability in RabbitMQ.

Environment

A malicious Message Broker has a channel to Voting Server and Control Components. Additionally, there is an implicit channel to SDM through Voting Server.

Possible further attacks

A malicious Message Broker is limited in attack surface and attack capabilities because it does not have its private key for signing the messages. Mainly it can only execute attacks that will impact the availability of the services or their state.

1. Drop Messages and return confirmation to Voting Server to change the SDM or Voting Server state.
2. drop votes sent through the system.

Impact

1. Availability is affected when Message Broker sabotages the delivery or exchange of messages with or without votes.

Overall only availability could be affected when Message Broker is malicious because the exchanged messages between the Voting Server and Control Components are signed. Message Broker does not have access to a certificate issued by the common root of trust.

5.5.4 Malicious Voter Portal

In this section, the author assumes that Voter Portal is malicious. The author can assume that the Voter Portal is untrustworthy overall but is only trustworthy with vote secrecy. That means

that unless the Voter Portal is breached, the protocol assumes that Voter Portal is trustworthy for maintaining the secrecy of the votes.

Attack prerequisites

An adversary can get into the Message Broker by executing the following attacks:

1. sends malicious data from the malicious Voting Server and gains access to the portal.
2. executes a 0-day or a known exploit in the Reverse Proxy that serves the Voter Portal to the voter.
3. sends malicious data as a voter to the Voter Portal to gain privileges.

Environment

A malicious Voter Portal or Client has a direct channel to the Voting Server and a Voter itself. Additionally, a Portal has an implicit channel to the Control Components.

Possible further attacks

A malicious Voter Portal can break vote secrecy as it can serve malicious javascript files to the voter and gain knowledge of what the voter chose. It can also try to exploit a Voting Server to be able to drop votes selectively or corrupt specific votes.

1. Break vote secrecy of future votes by serving malicious javascript.
2. Try to exploit the services of Voting Server with malicious data.
3. Drop votes selectively or make them invalid.

Impact

1. Vote Secrecy is affected when a Voter Portal serves malicious javascript
2. Availability is affected when a Portal drops votes selectively

In the end, Voter Portal makes it much easier to find out which votes to drop or corrupt to fabricate the election. But, of course, to make that even more manageable, an adversary would need to control the Voting Server and coordinatively lock and corrupt the votes.

Chapter 6

Conclusion

In this project, the author analyzed the E-Voting System Specification, Architecture document, and documents related to Operations and Testing. Then the author covered the implementation of all channels described in the adversarial scenarios and reviewed specific defense mechanisms.

Even though the system was not reproducible and the documentation did not describe some parts in detail - the author managed to find at least five vulnerabilities in 9 different spots. The first one was not reported because Swiss Post abandoned the Secure Logs as of version 0.14.0.0. The other four were reported. As of the 8th of June, two reports were accepted as low severity one, which is being reevaluated, and two reports are under review. The second vulnerability concerned a misconfiguration in Dockerfile. The third one concerned a lack of validation in SDM that led to RCE. The fourth one concerned the lack of a signature and data verification and validation in 5 different spots that led to voting locking and corruption. The fifth one concerned the lack of input sanitization that may lead to Path Traversal in the Voting Server.

From the implementation-specific findings, the author also found a possible design problem, where Voting Server acts as a communication medium for SDM and Control Components, which could be avoided by introducing a separate gateway between SDM and Control Components, which also might be untrustworthy, but which would not be reachable by the malicious Voting Server or Voting Client.

Overall, the system is still in development, and it will change in the future. Furthermore, the documentation is partially incomplete, and it should change. Additionally, Swiss Post could also complete the instructions for reproducing the system and parts of the system to let the security researchers focus on security. However, the author should note that the Swiss Post team does a great job in analyzing and improving the E-Voting system based on the researcher's input.

Finally, through the work done in the project, the author can be assured that it is yet early to deploy E-Voting System for real elections, and it still needs improvements based on suggestions and reports that this work and the work of other researchers provide.

Bibliography

[1] Ella Kummer. “Swiss Post E-Voting - Master Semester Project Report”. In: *Master Semester Project Report*. 2021.

[2] Swiss Post. “Infrastructure whitepaper of the Swiss Post e-voting system”. In: *E-voting documentation*. 2022.

[3] Swiss Post. “SwissPost Voting System architecture document - Version 0.9.1”. In: *E-voting documentation*. 2022.

[4] Swiss Post. “Swiss Post Voting System - System specification Version 0.9.9”. In: *E-voting documentation*. 2022.

[5] Mozilla Foundation. “Subresource Integrity”. In: 2022. URL: https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity.

[6] Bitnami. “Why non-root containers are important for security”. In: 2018. URL: https://docs.bitnami.com/tutorials/why-non-root-containers-are-important-for-security.

[7] Docker Inc. “Best Practices for writing Dockerfiles”. In: *Docker Documentation*. 2022. URL: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/.

[8] Ruben Santamarta. “Finding vulnerabilities in Swiss Post’s future e-voting system - Part 1”. In: 2022. URL: https://www.reversemode.com/2022/01/finding-vulnerabilities-in-swiss-posts.html.

[9] Ruben Santamarta. “Finding vulnerabilities in Swiss Post’s future e-voting system - Part 2”. In: 2022. URL: https://www.reversemode.com/2022/05/finding-vulnerabilities-in-swiss-posts.html.

[10] ETH Zurich Network Security Group. “Swiss Post E-Voting Scope 4: Network Security Analysis”. In: 2022. URL: https://www.newsd.admin.ch/newsd/message/attachments/71145.pdf.