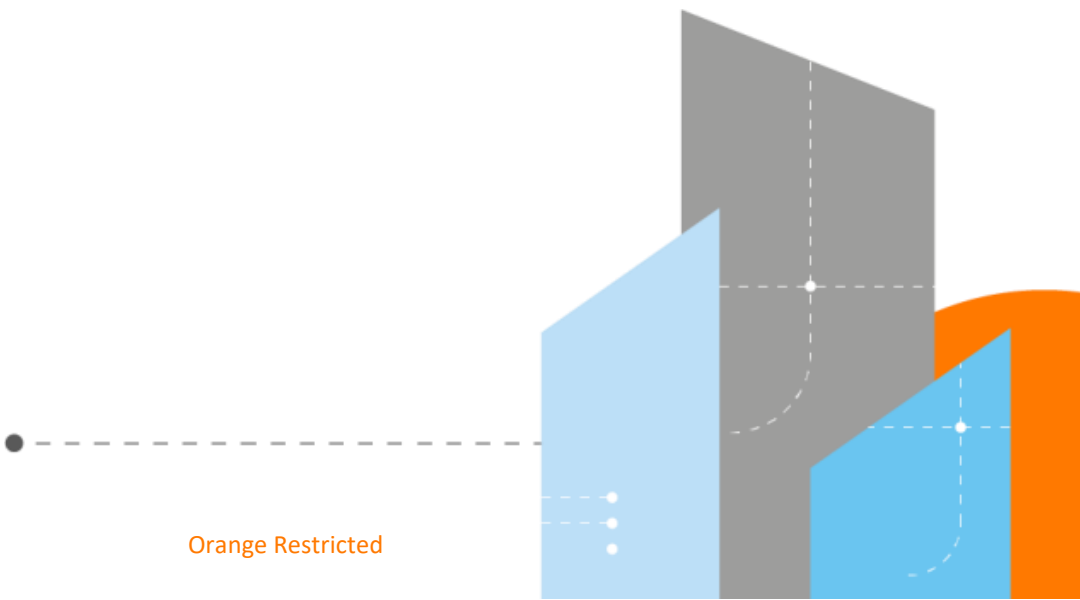
Cyberdefense

Federal Chancellery

Examination of the Swiss Internet voting system

Version: 1.0rev / Audit scope: Development process (2a)

24 June 2025



Contact information

Address	Contact
Orange Cyberdefense Switzerland SA Rue du Sablon 4 1110 Morges	Antonio Fontes Head of Advisory and Audit +41 21 802 64 01 antonio.fontes@orangecyberdefense.com

Contributors

Name	Function	Role
Antonio Fontes	Head of Advisory and Audit, Orange Cyberdefense Switzerland	Lead examiner

Document history

Version	Date	Author	Change details
0.1	05 June 2025	Antonio Fontes	Working version
1.0	20 June 2025	Antonio Fontes	First release
1.0rev	24 June 2025	Antonio Fontes	Fixed typos Improved some observations texts Added bibliography Extended note #44 (SAMM)

Contents

Management summary

Context, scope and objective of the examination

The objective of this examination was to assess to which extent Swiss Post's e-voting software development process complies with a subset of requirements (audit scope 2a – Development process) set forth by the Federal Chancellery's ordinance on e-voting.

In total, the examination covered 21 requirements.

Methodology

The examiners looked for evidence of effort to comply with said criteria by conducting interviews with Swiss Post personnel responsible for the development of the e-voting software, by analysing the related documentation (i.e., policies, procedures, specifications, reports, processes, etc.) and through interviews with selected personnel. Interviews were conducted both remotely (virtual sessions) and in-presence at Swiss Post's e-voting software development centre.

Results

During this examination, Swiss Post was able to demonstrate a high level of compliance with the requirements of the ordinance on e-voting. One minor finding (missing documentation) and two opportunities for improvement (extend the test concept with fuzz testing, establish control feedback and optimization loops) were identified. Three corresponding recommendations were formulated.

No major finding has been identified.

Final note

The examiners conclude this summary by thanking Swiss Post, and more particularly all the personnel that has been directly involved, for its cooperation and for the transparency demonstrated throughout the entire duration of the examination.

1 Context

1. Electronic voting (hereafter referred to as: “e-voting”) was introduced in Switzerland through multiple pilot schemes from 2004 onwards. A total of 15 cantons made e-voting possible in over 300 trials, until early 2019. Two implementations were available: the system provided by the canton of Geneva and the system operated by the Swiss Post (hereafter also referred to as “the Post”), initially developed by Scytl. In June 2019, the canton of Geneva announced the withdrawal of its e-voting system with immediate effect. It was followed in July of the same year by the announcement by the Swiss Post of the withdrawal of its e-voting system from operation to focus on improving the solution. At that point, e-voting was no longer possible in Switzerland.
2. In June 2019, the Swiss Federal Chancellery (hereafter also referred to as “Federal Chancellery”) was commissioned by the Federal Council to redesign a new trial phase, in collaboration with the cantons, using “e-voting systems, which are fully verifiable” [1]. This redesign of the trial phase focused on four objectives:
   a) Further development of the e-voting systems
   b) Effective controls and monitoring
   c) Increased transparency and trust
   d) Stronger connection with the scientific community
3. A taskforce was set up to make proposals for the future of internet voting. To that end, the Federal Chancellery invited experts from academia and industry to engage in a broad dialogue on internet voting in Switzerland. After this dialog, the Federal Chancellery and the cantons published a final report on the redesign and relaunch of internet voting trials, with a catalogue of measures [2].
4. The Federal Council took note of the final report and commissioned the Federal Chancellery to amend the legal bases of the Confederation regarding e-voting. In April 2021, the Federal Council opened a consultation procedure for the redesign of the e-voting trials. The redesign includes both a partial revision of the Ordinance on Political Rights (PoRo) [3] and a complete revision of the Federal Chancellery Ordinance on Electronic Voting (“VEleS”, or “OEV”) [4]. The OEV specifies, among others, the requirements for authorising electronic voting, including the technical and administrative controls for approving an e-voting system.
5. The Federal Chancellery issued an audit concept for the examination of Swiss internet voting systems defining the foundations for assessing the compliance of electronic voting systems with the draft OEV and its annex, as per chapter 26 of the annex of the draft OEV, and for obtaining recommendations for improvements [5].
6. In May 2022, the Federal Council enacted the partially revised Ordinance on Political Rights (PoRo) [6], which became applicable from July 1st 2022. The totally revised Federal Chancellery Ordinance on Electronic Voting (OEV) [7] came into force on the same date.

7. Orange Cyberdefense Switzerland ("OCD CH", formerly SCRT) was mandated by the Federal Chancellery to assess the compliance of Swiss Post's e-voting software development process against the applicable requirements of the OEV (*Scope 2a: Development process*) [8]. The examination was subsequently followed by two follow-up publications reporting on the progress of the recommendations [8], [9].
8. In November 2022, an updated version of the audit concept was issued by the Federal Chancellery [10].
9. In 2025, OCD CH was once again mandated by the Federal Chancellery to perform a new, full-scope audit of the e-voting system provided by the Post. On this occasion, the Federal Chancellery updated its audit concept to include additional requirements [11].

2 Methodology

2.1 Process

10. The examination was based on OCD CH's information systems audit methodology. The process specifies four-phases, as depicted in the figure below:

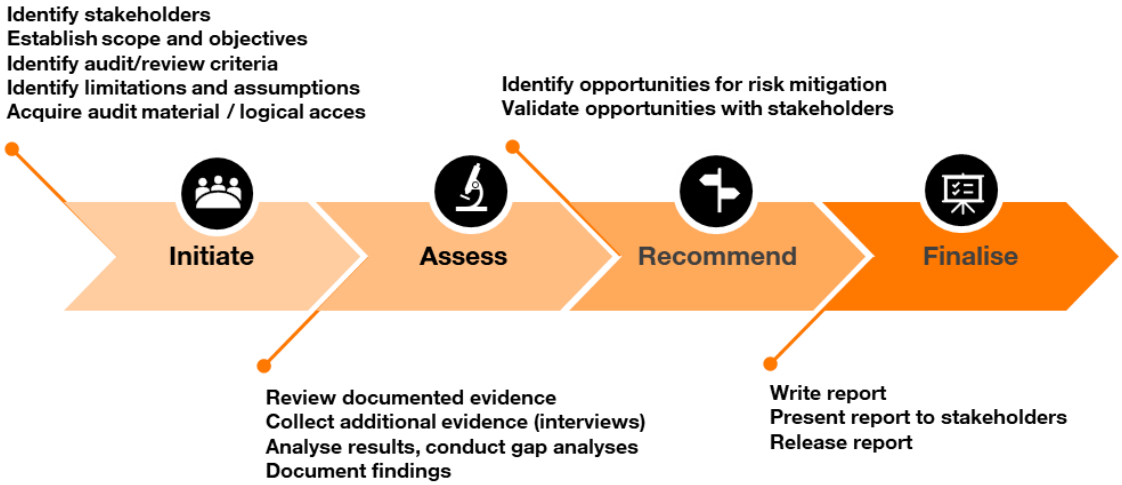


Figure 1: Examination process

2.2 Collection of evidence

11. As a general principle, the examiners aimed at acquiring two types of evidence for each requirement.

12. Types of evidence included: documents (e.g., policies, procedures, reports, etc.) and statements obtained from examinees during interviews.

2.3 Findings

13. The examiners raised a finding when a nonconformity or a significant opportunity for improvement was observed.

2.4 Classification of findings

14. Findings were classified based on their impact on requirements. The impact of each finding was assessed by the Federal Chancellery, as follows:

- **Minor nonconformity** - The examiners' observations reveal a partial or isolated failure to meet a requirement set by the Federal Chancellery.
- **Major nonconformity** - The examiners' observations reveal a failure to meet a requirement set by the Federal Chancellery.
- **Opportunity for improvement (OFI)** - While no failure—full or partial—to meet a requirement set by the Federal Chancellery, the examiners have identified an opportunity to meaningfully strengthen the development process.

2.5 Relevance of the assessment criteria

15. The examiners raised an issue when the wording of a given requirement set in the OEV was perceived as unclear, or subject to interpretation, preventing the examiners from performing an objective assessment of the criterion.

2.6 Assumptions

2.6.1 Trustworthiness of statements

16. The examiners assume that the examinees were honest and transparent when providing answers to the examiners' assessment questions. The observation of the actual implementation of the OEV's requirements within the e-voting system was limited to the demo made by the e-voting representatives of the Thurgau canton carried out to verify the accuracy of the examinees' statements.

2.6.2 Enforcement of security measures

17. The examiners assume that the security measures described in the documents provided as evidence in the context of the present examination are implemented and are effective. No observation of the actual implementation of the OEV's requirements within the e-voting system was carried out to verify the accuracy of the statements made in the security documents.

2.6.3 Tabula rasa

18. Although the examiners were aware of the results from a previous assessment, earlier observations were disregarded, and each requirement within the audit scope was evaluated anew. As a result, readers familiar with prior findings may encounter observations that differ from those in earlier reports. Such discrepancies may suggest a regression; however, they could also stem from (a) the examiners adopting a new or revised interpretation of the requirements or the evidence provided, or (b) the submission of new, potentially less satisfactory evidence.

3 Examination criteria

19. The examination focussed on assessing twenty Ordinance requirements selected by the Federal Chancellery as part of the scope 2a (development process. These include the following:

3.1 Section 8 - Information and instructions

Key	Requirement
8.13	Known flaws and the need for action associated with them are communicated transparently.

Table 1 - E-voting requirements: Information and instructions

3.2 Section 17 - System tests

Key	Requirement
17.1	The functions relevant to the security of the system (security functions) are tested. The tests are documented with test plans and expected and actual test results. The test plan: ■ specifies the tests to be performed, ■ describes the scenarios for each test, including any dependencies on the results of other tests. The expected results must show the results that are expected if the test is successfully executed. The actual results must be consistent with expected results.
17.2	An analysis must be made of the test coverage. This includes evidence that: ■ the tests defined in the test documentation match the functional specifications of the interfaces, ■ all interfaces have been fully tested.
17.3	An analysis must be made of the depth of testing. This includes evidence that: ■ the tests defined in the test documentation match the subsystems related to security functions and modules that play a role in ensuring security, ■ all subsystems related to the security functions mentioned in the specifications have been tested, ■ all modules that play a role in ensuring security have been tested.

Table 4 - E-voting requirements: System tests

3.3 Subsection 24.1 - Development and maintenance of information systems

Key	Requirement
24.1.1	A life cycle model is defined. The life cycle model: ■ is used for the development and maintenance of the software. ■ provides for the necessary controls during the development and maintenance of the software. ■ is documented.

Key	Requirement
24.1.2	A list must be made of the development tools used and configuration options chosen for the use of each development tool.
24.1.3	The documentation for the development tools includes: ■ a definition of the development tool, ■ a description of all conventions and directives used in the implementation of the development tool, ■ a clear description of the significance of all configuration options for using the development tool.
24.1.4	The implementation standards to be applied must be specified.
24.1.14	The software is provided with a unique identification.
24.1.15	The configuration management documentation includes: ■ a description of how configuration items are identified, ■ a configuration management plan describing how the configuration management system will be used in the development of the software and the procedures that will be followed for the adoption of changes or new elements, ■ evidence that the procedures for adoption provide for adequate review of changes for all configuration items.
24.1.16	The configuration management system: – uniquely identifies all configuration items, – provides automated measures to ensure that only authorised changes are made to configuration items, – supports the development of the software through automated procedures, – ensures that the person responsible for accepting the configuration item is not the same person who developed it, – identifies the configuration items that make up the security functions, – supports verification of all changes to the software using automated procedures, including logging of the author and the date and time of the change, – provides an automated method for identifying any configuration items that are affected by a change to a particular configuration item, – can identify the version of the source code on the basis of which the software is generated.
24.1.17	All configuration items are inventoried in the configuration management system.
24.1.18	The configuration management system is used in accordance with the configuration management plan.
24.1.19	A configuration list is created that contains the following items: ■ the software, ■ evidence of the checks required to ensure security compliance, ■ the parts that make up the software, ■ the source code, ■ the commit history, ■ reports on security flaws and on the status of their correction. For each element relevant to security functions, the developer is named. Each element is uniquely identified.

Key	Requirement
24.1.20	Software development security documentation includes: ■ a description of the physical, procedural, personnel, and other security measures necessary to protect and ensure the integrity of the design and implementation of the software in its development environment, ■ evidence that the security measures provide the necessary level of protection to preserve the integrity of the software.

Table 2 - E-voting requirements: Development and maintenance of information systems

3.4 Subsection 24.4 - Systematic correction of flaws

Key	Requirement
24.4.1	Processes are defined for the correction of flaws. The processes include: ■ documentation of specific aspects, in particular with regard to the traceability of flaws for all versions of the software, and of the methods used to ensure that system users have information on flaws, corrections and possible corrective actions, ■ the obligation to describe the nature and impact of all security flaws, information on the status of work to find a solution and the corrective measures adopted, ■ a description of how system users can make reports and enquiries about suspected flaws in the software known to the software developers, ■ a procedure requiring a timely response and automatic dispatch of security flaw reports and appropriate corrective actions to registered system users who may be affected by the flaw.
24.4.2	A process is defined for handling reported flaws: ■ This process ensures that all reported and confirmed flaws are corrected and that the procedures for correction are communicated to system users. ■ It provides for arrangements to ensure that the correction of security flaws does not give rise to new security flaws.

Table 3 - E-voting requirements: Systematic correction of flaws

3.5 Subsection 24.5 - Quality assurance

Key	Requirement
24.5	Regular and objective checks are carried out to ensure that the processes carried out and the associated work products comply with the description of the processes, standards and procedures to be implemented. Deviations are followed up until they are corrected.

Table 4 - E-voting requirements: Quality assurance

3.6 Subsection 25.13 - Quality of the source code and documentation

Key	Requirement
25.13.3	The integration tests cover all modules.
25.13.4	The software tests cover all modules.

Table 5 - E-voting requirements: Quality of the source code and documentation

4 Findings

4.1 F01 Insufficient documentation of security measures

Key	**F01**
Title	Insufficient documentation of security measures
Class	Major – **Minor** – Improvement
Requirement(s)	24.1.20 OEV - Software development security documentation includes: ■ a description of the physical, procedural, personnel, and other security measures necessary to protect and ensure the integrity of the design and implementation of the software in its development environment, evidence that the security measures provide the necessary level of protection to preserve the integrity of the software.
Rationale	Although the examiners observed that a broad and comprehensive set of security measures had been implemented to safeguard the e-voting development environment against identified threats, they received limited documented evidence concerning the physical, organizational, and personnel controls in place. Such evidence would typically inform observers of the measures put in place to secure the development environment from unauthorized access, in particular: ■ Measures deployed to protect Swiss Post e-voting development facilities from unauthorized physical access; ■ Measures deployed to protect the e-voting development infrastructure from unauthorized logical or physical access; ■ Measures to protect developer workstations and other compute / store assets from unauthorized logical or physical access, including immobilized and mobile equipment used to develop the e-voting software; ■ Personnel and organizational measures to protect e-voting software from fraudulent access or tampering, errors, or distractions (e.g., screening, onboarding, regular/continuous review, and termination procedures).
Recommendation(s)	Consider expanding the security whitepaper—or an equivalent supporting document—to include detailed information on the additional classes of measures (organizational, physical, personnel) implemented to protect the e-voting software's development environment.
Additional remarks	-

Table 6 – Finding F01 Insufficient documentation of security measures

4.2 F02 Opportunity for improved testing strategy (fuzz testing)

Key	**F02**
Title	Opportunity for improved testing strategy (fuzz testing)
Class	Major – Minor – **Improvement**
Requirement(s)	17.2 OEV - An analysis must be made of the test coverage. This includes evidence that: ■ the tests defined in the test documentation match the functional specifications of the interfaces, all interfaces have been fully tested.
Rationale	The examiners observed that fuzz testing[1] is not performed on interfaces, including those not directly exposed to voters. Fuzzing is a test method explicitly designed to observe a system's behaviour under random and/or unexpected situations. These may lead to the discovery of defects or undesired states that typically would not be identified through the execution of the documented test cases. Fuzzing may also be particularly useful when assessing interfaces whose operation was not (or could not be) verified through formal methods bur rather tested using methods that provide partial visibility (e.g., runtime security testing, penetration testing, bug bounties, etc.).
Recommendation(s)	Consider expanding the test strategy to incorporate fuzz testing, ensuring that at a minimum, the external interfaces of all components are subjected to fuzzing.
Additional remarks	This finding and associated recommendation were reported in a previous examination [8].

Table 7 – Finding F02 Opportunity for improved testing strategy (fuzz testing)

[1] « Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program » (Wikipedia, 2025)

4.3 F03 Opportunity for establishing control feedback and optimization loops

Key	**F03**
Title	Opportunity for establishing control feedback and optimization loops
Class	Major – Minor – **Improvement**
Requirement(s)	24.5 OEV - Regular and objective checks are carried out to ensure that the processes carried out and the associated work products comply with the description of the processes, standards and procedures to be implemented. Deviations are followed up until they are corrected.
Rationale	Although the examiners observed the use of processes and tools intended to detect defects, errors, and vulnerabilities in work items, these mechanisms do not yet appear to be sufficiently monitored to ensure proper execution or automation. For example, key indicators—such as execution accuracy, coverage ratios, frequency ratios, failure rates, and regression metrics—are not systematically tracked. This lack of oversight may impede the timely identification of weaknesses in the processes and tools, potentially diminishing operational efficiency. However, the associated risk is partially mitigated by the test concept, which employs a variety of testing tools and methods across the same set of artefacts at different stages of development (e.g., static analysis, runtime testing, penetration testing). This layered approach provides an implicit level of quality control over the tools and methods in use.
Recommendation(s)	Consider establishing a monitoring framework that provides clear visibility into the performance and operation of the controls and measures implemented to meet the Chancellery's e-voting requirements. This framework could be developed in two stages: ■ An initial phase focused on monitoring the accurate execution of controls and measures, allowing for quantitative assessment and the identification of deviations from standard performance. ■ When relevant, a subsequent phase dedicated to analyzing both performance and its fluctuations, facilitating root cause analysis and the development of targeted corrective actions.
Additional remarks	-

Table 8 – Finding F03 Opportunity for establishing control feedback and optimization loops

5 Summary of results

20. The tables below present a summary of the findings observed for each Federal Ordinance requirement covered by the assessment. For each requirement, the number of major nonconformities, minor nonconformities, and opportunities for improvement (OFIs) is indicated.

Section 8 (information and instructions)

Key	Requirement	Major	Minor	OFI
8.13	Known flaws and the need for action associated with them are communicated transparently.	-	-	-

Table 9 - Summary of results (section 8)

Section 17 (system testing)

Key	Requirement	Major	Minor	OFI
17.1	The functions relevant to the security of the system (security functions) are tested. The tests are documented with test plans and expected and actual test results. The test plan: ■ specifies the tests to be performed, ■ describes the scenarios for each test, including any dependencies on the results of other tests. The expected results must show the results that are expected if the test is successfully executed. The actual results must be consistent with expected results.	-	-	-
17.2	An analysis must be made of the test coverage. This includes evidence that: ■ the tests defined in the test documentation match the functional specifications of the interfaces, all interfaces have been fully tested.	-	-	1
17.3	An analysis must be made of the depth of testing. This includes evidence that: ■ the tests defined in the test documentation match the subsystems related to security functions and modules that play a role in ensuring security, ■ all subsystems related to the security functions mentioned in the specifications have been tested, all modules that play a role in ensuring security have been tested.	-	-	-

Table 10 - Summary of results (section 17)

Section 24.1 (development and maintenance)

Key	Requirement	Major	Minor	OFI
24.1.1	A life cycle model is defined. The life cycle model: ■ is used for the development and maintenance of the software. ■ provides for the necessary controls during the development and maintenance of the software. is documented.	-	-	-
24.1.2	A list must be made of the development tools used and configuration options chosen for the use of each development tool.	-	-	-
24.1.3	The documentation for the development tools includes: ■ a definition of the development tool, ■ a description of all conventions and directives used in the implementation of the development tool, a clear description of the significance of all configuration options for using the development tool.	-	-	-
24.1.4	The implementation standards to be applied must be specified.	-	-	-
24.1.14	The software is provided with a unique identification.	-	-	-
24.1.15	The configuration management documentation includes: ■ a description of how configuration items are identified, ■ a configuration management plan describing how the configuration management system will be used in the development of the software and the procedures that will be followed for the adoption of changes or new elements, evidence that the procedures for adoption provide for adequate review of changes for all configuration items.	-	-	-
24.1.16	The configuration management system: – uniquely identifies all configuration items, – provides automated measures to ensure that only authorised changes are made to configuration items, – supports the development of the software through automated procedures, – ensures that the person responsible for accepting the configuration item is not the same person who developed it, – identifies the configuration items that make up the security functions, – supports verification of all changes to the software using automated procedures, including logging of the author and the date and time of the change, – provides an automated method for identifying any configuration items that are affected by a change to a particular configuration item,	-	-	-

Key	Requirement	Major	Minor	OFI
	– can identify the version of the source code on the basis of which the software is generated.			
24.1.17	All configuration items are inventoried in the configuration management system.	-	-	-
24.1.18	The configuration management system is used in accordance with the configuration management plan.	-	-	-
24.1.19	A configuration list is created that contains the following items: ■ the software, ■ evidence of the checks required to ensure security compliance, ■ the parts that make up the software, ■ the source code, ■ the commit history, ■ reports on security flaws and on the status of their correction. For each element relevant to security functions, the developer is named. Each element is uniquely identified.	-	-	-
24.1.20	Software development security documentation includes: ■ a description of the physical, procedural, personnel, and other security measures necessary to protect and ensure the integrity of the design and implementation of the software in its development environment, evidence that the security measures provide the necessary level of protection to preserve the integrity of the software.	-	1	-

Table 11 - Summary of results (subsection 24.1)

Section 24.4 (systematic correction of flaws)

Key	Requirement	Major	Minor	OFI
24.4.1	Processes are defined for the correction of flaws. The processes include: ■ documentation of specific aspects, in particular with regard to the traceability of flaws for all versions of the software, and of the methods used to ensure that system users have information on flaws, corrections and possible corrective actions, ■ the obligation to describe the nature and impact of all security flaws, information on the status of work to find a solution and the corrective measures adopted, ■ a description of how system users can make reports and enquiries about suspected flaws in the software known to the software developers,	-	-	-

Key	Requirement	Major	Minor	OFI
	a procedure requiring a timely response and automatic dispatch of security flaw reports and appropriate corrective actions to registered system users who may be affected by the flaw.			
24.4.2	A process is defined for handling reported flaws: ■ This process ensures that all reported and confirmed flaws are corrected and that the procedures for correction are communicated to system users. It provides for arrangements to ensure that the correction of security flaws does not give rise to new security flaws.	-	-	-

Table 12 - Summary of results (subsection 24.4)

Section 24.5 (quality assurance)

Key	Requirement	Major	Minor	OFI
24.5	Regular and objective checks are carried out to ensure that the processes carried out and the associated work products comply with the description of the processes, standards and procedures to be implemented. Deviations are followed up until they are corrected.	-	-	1

Table 13 - Summary of results (subsection 24.5)

Section 25.13 (quality of the source code and documentation)

Key	Requirement	Major	Minor	OFI
25.13.3	The integration tests cover all modules.	-	-	-
25.13.4	The software tests cover all modules.	-	-	-

Table 14 - Summary of results (subsection 25.13)

6 Summary of recommendations

21. The following table summarizes the recommendations based on the findings. For each recommendation, the table lists its identifier, the associated finding, the applicable OEV requirement, and the classification of the finding (minor, major, or OFI).

Key	Finding	OEV req.	Class	Recommendation
R01	F01	24.1.20	Minor	Consider expanding the security whitepaper—or an equivalent supporting document—to include detailed information on the additional classes of measures (organizational, physical, personnel) implemented to protect the e-voting software's development environment.
R02	F02	17.2	OFI	Consider expanding the test strategy to incorporate fuzz testing, ensuring that at a minimum, the external interfaces of all components are subjected to fuzzing.
R03	F03	24.5	OFI	Consider establishing a monitoring framework that provides clear visibility into the performance and operation of the controls and measures implemented to meet the Chancellery's e-voting requirements. This framework could be developed in two stages: ■ An initial phase focused on monitoring the accurate execution of controls and measures, allowing for quantitative assessment and the identification of deviations from standard performance. ■ When relevant, a subsequent phase dedicated to analyzing both performance and its fluctuations, facilitating root cause analysis and the development of targeted corrective actions.

Table 15 - Summary of recommendations

7 References

[1] Administration numérique suisse, “Swiss citizens should be able to vote electronically.” Accessed: Jun. 24, 2025. [Online]. Available: https://www.digital-public-services-switzerland.ch/en/egovernment-implementation-plan/redesigning-evoting

[2] Swiss Federal Chancellery, Political Rights Section, “Redesign and relaunch of trials - Final report of the Steering Committee Vote électronique (SC VE).” Nov. 30, 2020. Accessed: Dec. 06, 2021. [Online]. Available: https://www.bk.admin.ch/dam/bk/en/dokumente/pore/Final%20report%20SC%20VE_November%202020.pdf.download.pdf/Final%20report%20SC%20VE_November%202020.pdf

[3] Swiss Federal Chancellery, Political Rights Section, “Partial revision of the Ordinance on Political Rights and total revision of the Federal Chancellery Ordinance on Electronic Voting (Redesign of Trials).” Apr. 28, 2021. Accessed: Dec. 06, 2021. [Online]. Available: https://www.bk.admin.ch/dam/bk/en/dokumente/pore/Explanatory%20report%20for%20consultation%202021.pdf.download.pdf/Explanatory%20report%20for%20consultation%202021.pdf

[4] Swiss Federal Chancellery, Political Rights Section, “Federal Chancellery Ordinance on electronic voting (OEV).” Apr. 28, 2021. Accessed: Dec. 06, 2021. [Online]. Available: https://www.bk.admin.ch/dam/bk/en/dokumente/pore/Explanatory%20report%20for%20consultation%202021.pdf.download.pdf/Explanatory%20report%20for%20consultation%202021.pdf

[5] Swiss Federal Chancellery (FCh) - Political Rights section, “Audit concept for examining Swiss Internet voting systems - v1.3.” May 18, 2021.

[6] Swiss Federal Chancellery, Political Rights Section, “Partial revision of the Ordinance on Political Rights and total revision of the Federal Chancellery Ordinance on Electronic Voting (Redesign of Trials).” May 25, 2022. Accessed: Dec. 06, 2021. [Online]. Available: https://www.newsd.admin.ch/newsd/message/attachments/71705.pdf

[7] Swiss Federal Chancellery, “Federal Chancellery ordinance on electronic voting (OEV).” May 25, 2022. [Online]. Available: https://www.fedlex.admin.ch/eli/cc/2022/336/en

[8] A. Fontes, “Examination of the Swiss Internet voting system - Audit scope 2a - Development processes - Follow-up Audit (round 2).” Nov. 02, 2022. [Online]. Available: https://www.bk.admin.ch/bk/en/home/politische-rechte/e-voting/ueberpruefung_systeme.html

[9] A. Fontes, “Examination of the Swiss Internet voting system - Audit scope 2a - Development processes (round 3, follow-up).” Aug. 04, 2023. [Online]. Available: https://www.bk.admin.ch/dam/bk/en/dokumente/pore/E_Voting/Examination_reports_August2023/Scope%202a%20Final%20Report%20SCRT%2004.08.2023.pdf.download.pdf/Scope%202a%20Final%20Report%20SCRT%2004.08.2023.pdf

[10] Swiss Federal Chancellery (FCh) - Political Rights section, “Audit concept for examining Swiss Internet voting systems - v1.5.” Sep. 15, 2022. [Online]. Available: File reference: 431.0-2/5/12/16
[11] Swiss Federal Chancellery (FCh) - Political Rights section, “Audit concept for examining Swiss Internet voting systems - v1.6.” Feb. 07, 2025. [Online]. Available: File reference: 431.0-2/15/4/7
[12] OWASP, “OWASP Web security testing guide ver.4.2.” 2020. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/v42/
[13] the ZAP dev team, *ZAP - Zed Attack Proxy*. (Mar. 25, 2025). Checkmarx. [Online]. Available: https://www.zaproxy.org/
[14] OWASP, “Software assurance maturity model (SAMM),” OWASP SAMM. Accessed: Dec. 08, 2021. [Online]. Available: https://owaspsamm.org/
[15] “SAMM Assessment.” Accessed: Jun. 24, 2025. [Online]. Available: https://owaspsamm.org/assessment/
[16] Microsoft security, “Secure the developer environment for Zero Trust.” Feb. 26, 2025. [Online]. Available: https://learn.microsoft.com/en-us/security/zero-trust/develop/secure-dev-environment-zero-trust

8 Appendix

8.1 Index of tables

8.2 Evidence catalogue

22. The table below summarises the evidence documented and made available by Swiss Post to the examiners:

OEV Key	Evidence proposed
8.13	Bug bounty platform procedures E-voting security whitepaper Gitlab issues page Public examination rules
17.1, 17.2, 17.3	Test concept Test concept - implementation Test plan and scenarios Test results archive Xray reports Xray scan reports
24.1.1,24.1.2, 24.1.3, 24.1.4, 24.1.14, 24.1.15, 24.1.16, 24.1.17, 24.1.18, 24.1.19	Secure Software Development Process Secure Software Development Process - internal details
24.1.20	Security Whitepaper
24.4.1, 24.4.2	About Contributing Security Whitepaper SPOC community guidelines
24.5	E-voting security whitepaper OWASP SAMM assessment report
25.13.3, 25.13.4	Secure Software - Development Process Test concept Unit test reports

Table 16 - Evidence catalogue

8.3 Interview log

23. The table below summarises the interviews conducted as part of the assessment:

Date	Location	Topic(s) / objective(s)
2025-04-30	Virtual	Development process Roles and responsibilities
2025-04-30	Virtual	Risk management Threat modeling
2025-04-30	Virtual	Test concept I (test strategy)
2025-04-30	Virtual	Software development governance (OWASP SAMM review)
2025-05-07	Virtual	Trusted build and development
2025-05-07	Virtual	Test concept II (test plan and methods)
2025-05-14	On-site	Test concept III (business logic testing, security testing)
2025-05-14	On-site	Development process tooling and documentation
2025-05-14	On-site	Test concept IV (evidence of testing)
2025-05-14	On-site	Third-party dependencies
2025-05-14	On-site	Final debriefing

Table 17 – Interview log

8.4 Examiners observations

8.4.1 Section 8 (information and instructions)

24. Flaws identified internally or reported through designated public channels are disclosed on the e-voting Gitlab platform.
25. The platform offers a ticketing system (issues) that enables the public to track Swiss Post's responses to findings or vulnerabilities affecting the e-voting system. Publicly available information typically includes the impact assessment, the decision made (e.g., mitigation or acceptance), and the actions taken by Swiss Post's e-voting development team.
26. Swiss Post's internal security team also conducts independent security research on its own products. Vulnerabilities discovered through these efforts are expected to be published on the Gitlab platform, ensuring transparency regardless of whether the issue was found internally or reported externally.
27. In addition to Gitlab's issue tracking, vulnerabilities can be reported through Swiss Post's proprietary messaging platform (IncaMail) or via an online form. It is important to note that Swiss Post acts as the primary recipient for several of these reporting channels. This arrangement may raise concerns regarding the potential circumvention of the third-party oversight initially envisioned by the Swiss Chancellery (i.e., the observers).
28. There exists a scenario in which Swiss Post could withhold or delay the disclosure of newly reported vulnerabilities, particularly during periods of heightened operational pressure—such as when ballots are underway, when resources are temporarily constrained, or when significant political decisions are pending.

8.4.2 Section 17 (system testing)

29. The examiners are required to assess the notion of a function's relevance to security in this context. In the context of the audit, function relevance to the security of the system was qualified in two cases:

- Functions directly used as part of the e-voting protocol (e.g., cryptographic functions carrying the burden of secrecy of the vote),
- Functions, whose incorrect design or implementation could be abused to compromise the security of the e-voting platform (e.g., a computation method in which a malformed or forged input could trigger a resource exhaustion).

30. The examiners note that the distinction between the two concepts (security functions vs. secure functions) remains uninformed in the concept. Security functions, which compose the e-voting protocol, are identified as such and thus benefit from accrued scrutiny and are subject to a multilayer verification process. This verification process includes:

- Review of the design of changes proposals,

- Explicit involvement of security and cryptography specialists both from the Swiss Post's e-voting development team but also third-party experts,
- Review of the implementation (through the e-voting examination program, among others),
- Public review (the design and implementation are subject to voluntary reviews, by experts worldwide).

31. The second category of functions—those whose incorrect or inadequate design or implementation could be exploited by threat agents to compromise the e-voting platform—has not yet been formally identified. As a result, these functions are not explicitly subjected to targeted testing. The absence of a formal identification process for such functions may undermine the effectiveness of the test plan, particularly in situations where comprehensive coverage is unattainable or incomplete. Nevertheless, this vulnerability is largely mitigated by the overall e-voting test strategy, which incorporates a "business logic test" (BLT) as part of the verification process for new platform releases. The BLT phase involves executing the test protocol specified in the OWASP Web Testing Guide [12], which, among other measures, mandates testing all input interfaces against malformed or malicious inputs.
32. This process (BLT) is, however, fully externalized to such an extent that the e-voting ultimately trusts the tester (a third-party information security services provider) and does not track whether a) the test plan was fully executed and/or whether potential regressions are observed, b) the content of the test plan. It should be noted that the tester includes information on both successful tests and failed tests, thus facilitating the tracking of potential regressions.
33. Regarding the test plan itself, the examination team notes and confirms the documentation of the test plan, of the test scenarios and of the testing procedures, including documented tests specifications, dependencies towards other tests and expected results of the tests. Reports of successful testing could be examined.
34. Tests are organized and documented per voting component, which facilitates the verification of the OEV requirements 17.2 and 17.3.
35. Tests can be listed per module/component and are inventoried. All existing modules appear in that list.
36. Test dependencies are not shown in the test plan but may be unnecessary due to the adopted strategy:

- No tests should fail
- Test cases aim at performing a complete procedure.

37. Tests can be listed per module/component and are inventoried. All existing modules appear in that list.
38. The test strategy involves, among others:

- Unit testing: publicly available test plan, highly reproducible and comparable across runs, current coverage above 85% for the entire code base, regressions can be identified thanks to complete traceability of test runs and their author.
- Functional testing: current coverage is reportedly complete (all functional requirements are subject to functional testing) but the effectiveness of the implementation remains untracked and thus unmeasurable. By extension, tests delegated to the BLT process (business-logic testing) remain unidentified.

39. The e-voting development team uses various tools and methods at all major phases of its systems development lifecycle to routinely review its work products (i.e., source code, pipeline instructions, infrastructure code) and test them for defects, errors, and security vulnerabilities. The following testing methods and tools are used by the e-voting development team prior to each release of the e-voting software:

- Checking for known publicly reported vulnerabilities (CVEs) in third parties and libraries embedded in the product,
- Unit testing for all interfaces and methods,
- Scanning the source code for known defects, errors and vulnerabilities (static testing)
- Conducting end-to-end scenario testing using various methods such as smoke testing, regression testing and business logic testing,
- Scanning the runtime and release artifacts for known errors and vulnerabilities (runtime / dynamic testing),
- Conducting a security assessment based on the OWASP's web application security testing guide,
- Conducting a penetration test upon each release.

40. All subsystems are subject to testing (verbal + visual confirmation in test plan acquired).

41. Depth of testing appears to cover smoke testing (execution of rapid pass/fail scenarios that tend to maximize the number of components and interfaces tested within the smallest unit of testing), unit testing (unitary instruction tests) and functional tests. Function testing approach implements a positive testing model ("verify that this works or that this failure mechanism works correctly").

42. Abuse cases are not formalized per-se as part of the regular test plan but rather some plans can be considered abuse cases in rare occasions ("ensure it fails"), but they are covered through the BLT. Fuzzing and monkey tests are not performed.

43. It was verified that subsystems are tested (request to provide test artifacts on randomly selected components) but there was no central view that provided the information (maturity issue?). Information is available but requires manual drill or pull.

8.4.3 24.1 (development and maintenance)

44. On the development process, it is observed:

- Dev/architect role-based training: not automatically triggered for newcomers. All current members of the dev team underwent training. Risk is mitigated by the (currently) high stability of the team (employee attrition close to 0).
- Security requirements analysis: performed, mainly through the Chancellery's Ordinance on e-voting [7]
- Threat modelling: performed regularly and on all security-relevant work items requests. However, the trigger that effectively processes the execution of a threat modelling session remains informal. Maturity is elevated and can be observed through the use of reference methods, the use of purpose-built extensions (e-voting specific top-level threats) to increase threat findability, models are auditable and traced).
- Coding convention: a guide (hitchhiker's' handbook) is available to the development team. The team knows about the location of the guide.
- Security API: well established concept within the e-voting protocol but remains informal outside (e.g., voter portal).
- Static analysis: all source code is subject to code scanning prior to release. Evidence of scanning provided, with auditable results for past tests.
- Peer-review: submission of new code, infrastructure code, pipeline code, and other artifacts requires approval from a third-party senior developer (configured in Bitbucket).
- Depdendency scanning / composition analysis: all libraries and third-party components are systematically scanned for known vulnerabilities (CVEs) both prior to release and on a continuous basis thereafter, as well as during the build process, using a commercial third-party tool. This tool automatically generates new work items and creates upgrade tasks within the configuration management platform employed by the e-voting software team. Scan reports are maintained as evidence.
- Dependency scanning: if a vulnerability is flagged during live operations—such as while a ballot is in progress—a predefined procedure is initiated. The vulnerability undergoes assessment to determine its reproducibility, impact and the likelihood of exploitation. Based on the evaluation, the response may involve a complete shutdown of the e-voting platform or the planning and deployment of a patch, depending on the severity and risk level identified.
- Dependency vetting: the integration of third-party components is governed by an architecture board approval process, which prohibits direct embedding of external libraries via developer workstations. To initiate approval, a formal dependency assessment form must be completed and submitted in accordance with a documented, standardized procedure. This process requires the collection of key information, including the number of previously identified CVEs, the package's historical activity, the developers' historical activity, and the applicable licensing schemes. Additionally, a threat model must be produced to ensure that all relevant risks associated with the component are identified and can be monitored over time.

Authorizations are time-limited; continued access to a library necessitates periodic updates to the associated security information.

- Dependency vetting: a notable push toward re-importing third-party components into the internal codebase, with the aim of reducing reliance on external dependencies, is observed. This internalization process includes refactoring the original code to minimize the attack surface and retaining only those elements deemed strictly necessary. However, the rationale behind this initiative remains unclear to the examiners, and the provided justification was not found to be fully convincing. The primary argument cited is attack surface reduction—specifically, mitigating risks such as the compromise of a developer account that could alter an open-source package.
- The development team was made aware of the debt transference countereffect often encountered when software engineering teams internalize third-party code. By severing ties with the external source code supplier, the team assumes additional responsibilities, including: a) proactively identifying potential threats and vulnerabilities that may be discovered in the original package; b) establishing and maintaining both patch detection and patch management processes to ensure that changes to the original package are identified, assessed for their impact, and incorporated into the internalized code as necessary; and c) managing an increased volume of code that must be maintained internally.
- Runtime security testing (automated): automated runtime security testing is carried out using a publicly available, open-source platform that offers automated assessment capabilities [13]. This tool utilizes an HTTP-based spider engine to map system entry points and their specifications—though the attack surface can also be defined manually if required. It then executes a suite of well-known attack vectors against each interface to identify potential vulnerabilities, such as invalid or error responses, server error messages, or the detection of canaries. The development team has been made aware of the typical limitations and risks associated with over-reliance on automated tools for security testing. These include the potential inability to accurately map the software's attack surface, insufficient coverage of test scenarios, inadequate test case selection, and the presence of bugs or coding errors within the scanning engine itself. Some of the risk is mitigated with a runtime security test performed by a specialized information security provider and conducted prior to each release. Additionally, regular penetration testing is incorporated into the ongoing audit procedures for the e-voting system, source code is publicly available, build is publicly reproducible, and a bug bounty program allows for broader and independent scrutiny by the security community. Evidence of the results and related artefacts from both the runtime security and penetration tests were presented to the examiners.
- Runtime security testing / business logic testing: a BLT test is performed by a third-party information security services provider upon each release. The test is fully delegated to the supplier and the e-voting software team does not track the test plan nor review it to guarantee the absence of regressions or missing necessary tests. The examiners were allowed to review one a recent report and could confirm

that the supplier executes a standardized testing procedure [12] and fully documents its results (both pass and fail outcomes). While not stored as structured data, records are kept available and can be audited. The team was informed that the approach may be suffer from limitations, such as an extensive focus on HTTP attacks and the voter portal (the e-voting system also exposes interfaces that communicate with other protocols and may be exposed to other categories of threat actors that will not be limited by the e-voting portal), and a business logic test plan that remains under control of the supplier,

- Penetration testing: performed, see runtime security testing.
- Bug bounties: performed, see runtime security testing.
- Process documentation: the process is documented.
- Counter-testing / tool attestation: one of the static analysers was tested using fault injection (i.e., intentionally injecting software vulnerabilities and errors in the code to assert good operation of the testing platform). The second analyser was not yet tested. More generally, security testing tools and methods are not yet subject to counter-testing nor attestation procedures.
- Systems development lifecycle: the e-voting software team uses OWASP SAMM [14] as a governance framework to align its development process with secure software engineering best practices. The team's processes and practices were audited by Swiss Post's information security team following SAMM's available self-assessment guidance material [15] which produced average scores above 2.0 for each of the five business functions (governance, design, implementation, verification, operations). Results of the audit, including reports, scores, recommendations and the roadmap to align the e-voting software team to its objective were shown to the examiners.

45. The configuration of the testing tools is inadequately documented, and, for some tools, documentation is entirely absent. However, due to the highly automated and containerized nature of the integration and deployment processes, the logging behaviour was also reviewed during the assessment. It was confirmed that the specific configurations of the tools used for building, packaging, and testing artefacts can, in most cases, be reconstructed from logs generated by the automation layer. For example, all command parameters invoked during build, deployment, or new releases are systematically logged and retained as part of the standard logging strategy. Consequently, while information regarding the configuration options of the testing toolset can be retrieved through investigative means (i.e., by accessing log archives), it remains inaccessible by more regular methods.

46. Several standards are defined internally and at multiple levels. These include coding standards, standard security requirements (baseline requirements) and testing standards. Procedures are set to confirm adherence to a subset of said standards. The peer review process includes verifying for compliance with internal standards as part of the tasks performed by developers. Security-relevant work items (tagged as such) are reviewed by personnel with adequate training and/or experience in covering the security standards, which typically include internal security directives such as

health checks and the *security reminder*. Another layer of review resides in the architecture board, where design decisions can be approved. Finally, the definitions of ready and done (i.e., the process through which a candidate work item is considered ready for implementation or completed, respectively) are also standardized and attestation of compliance can be obtained through checklists to which an architect or developer visa is associated.

47. All releases are tagged with a unique identification sequence. This applies not only to production releases but also to other intermediary releases (native behaviour of the configuration management system).

48. While most of the documented evidence can be found in the development process, the testing concept and the security whitepaper, the precise configuration and versioning of the programs or scripts used to execute satellite or pipeline tasks remain either poorly documented or spread across multiple documents. A typical symptom of this "information spread" appears in the initial requirement-evidence mapping matrix provided to the examiners at the beginning of the assessment: for each requirement, a series of documents is provided, but not the location of the information in the document. Examiners either needed to find each precise detail in each of the documents provided or ask the Swiss Post e-voting team for direct pointers.

49. The chosen configuration management solution natively offers all of the functions required in the requirement (OEV 24.1.16), including but not limited to:

- The automatic unique tagging / identification of all configuration items, including individual items (e.g., script, source code document, configuration file, binary, etc.) and collections of items (e.g., branches, releases, commits and pull requests, etc.);
- Granular access control capabilities (i.e., enforcement of granular roles definitions, individual or grouped assignments, at various levels (e.g., resource, project, group of projects, etc.);
- Full build and integration/deployment automation capabilities;
- Change approval workflows (i.e., require third-party approval, role-based approval, user-based approval, approval bypass procedures, etc.);
- Exposing security functions through identifiable security-relevant settings;
- Auditing of all read and write operations to the data plane (e.g., source code) and to the control plane (e.g., security configuration);
- Atomicity of changes, including all affected dependencies;
- Unique association between a production release identifier and its corresponding artifacts (e.g., source code, configuration assets, etc.);
- Production of inventories of assets.

50. Configuration assets can be inventoried through the configuration management system, including but not limited to:

- Software releases;
- Audit reports of reviews performed/approvals on the assets;

- Inventory of a software release;
- Source code linked with a software releases;
- Commits associated with a software release;
- Issues flagged as security-relevant associated with a software release;
- States on security-relevant issues associated with a software release;
- Authorship of all artifacts associated with a software release;

51. Each asset processed by the configuration management system is associated to a unique identification key, including an identification for each version of a given asset.

52. The security documentation of the software development process enumerates a comprehensive set of technical controls and measures to either increase the robustness and resilience of the e-voting software.

53. However, the documentation poorly addresses measures beyond the technical, technological, or organizational domains—specifically, physical security controls and personnel-related safeguards. Such measures would typically include protocols to prevent unauthorized physical or logical access to developer equipment (which may extend beyond company-issued devices, as threat actors are not necessarily limited to targeting professional assets) and to development systems, such as centralized computing and storage resources. Additionally, personnel-related controls—such as candidate screening, employee onboarding, periodic reviews, and termination procedures—are not detailed. These objectives could generally be achieved using external resources dedicated to the protection of development environments [16].

54. The evidence provided lacks information on how evidence of efficiency of measures and controls is obtained (see also: requirement 24.5 – quality assurance).

8.4.4 Subsection 24.4 (systematic correction of flaws)

55. The process through which flaws and defects are corrected is documented and is subject to accrued transparency thanks to the duplication of information in the external Gitlab e-voting project issue page.

56. The impact of a finding deemed relevant for the security of the e-voting system is assessed as part of its initial qualification using the CVSS scoring system.

57. Steps and stages through which the findings is processed can be audited through the external issues page. The internal issue tracker also provides visibility on these operations.

58. The flaw reporting process inherently relies on a certain degree of trust in Swiss Post. This is partly due to the existence of multiple reporting channels, ranging from highly transparent options—such as the publicly accessible GitLab project issues page—to more opaque ones, like the online form available on Swiss Post's website. At time of examination, there was no formalized process to ensure that the resolution of a vulnerability does not introduce regressions. Nevertheless, the examiners consider it likely that the overall test strategy—which applies a sequence of diverse testing

methods to the same artifacts, and incorporates increasingly formal approaches when mitigation directly impacts the e-voting protocol—serves to significantly reduce the risk of regression.

8.4.5 Subsection 24.5 (quality assurance)

59. The processes undertaken to design and develop the e-voting software are thoroughly documented and subjected to regular reviews, conducted annually, to ensure alignment with the requirements stipulated by either Swiss Post or the Swiss Federal Chancellery. Additionally, the development team's work products—such as source code, pipeline instructions, and infrastructure code—are routinely reviewed to identify not only quality issues but also common software defects, errors, and security vulnerabilities.

60. Certain verification processes for these work products are automated, employing tools such as CVE checkers for third-party libraries, source code analyzers, and runtime application testing solutions. Other verification activities are carried out using specialized methods, including business logic testing, penetration testing, and participation in public bug bounty programs. Any deviations from established requirements are systematically translated into actionable work items, ensuring complete traceability until each issue is resolved or formally closed—whether through system testing, systematic flaw remediation, or ongoing development and maintenance efforts.

61. Should a defect be identified as potentially impacting the system's security, it is subjected to an impact assessment and subsequently reported in the publicly accessible issue tracker. The Swiss Federal Chancellery emphasizes that both processes and work products must undergo regular and objective oversight, asserting that "regular and objective checks are carried out to ensure that the processes undertaken and the work products produced meet established expectations." This suggests that the Chancellery's definition of "objective oversight" extends beyond an annual review of processes.

62. The examiners identified limited evidence of robust, consistent monitoring mechanisms for processes designed to ensure that work products are free from defects, errors, or security vulnerabilities. Although fault injection—deliberately introducing vulnerabilities into the source code—was confirmed during a single instance of vulnerability scanning, this practice appeared to be an isolated measure rather than part of a structured, systematic approach.

63. Similarly, a shared checklist was observed to verify that each release complies with a set of requirements.

64. Additionally, it was noted that findings from the bug bounty program are occasionally leveraged to refine internal defect detection processes, though this approach appears situational rather than systematically integrated. While performance metrics, such as unit testing coverage and business logic testing coverage, can be manually generated upon request, the examiners found no indication of a centralized framework to

consolidate and monitor these metrics as key performance indicators or equivalent benchmarks.

8.4.6 Subsection 25.13 (quality of the source code and documentation)

65. The examiners obtained evidence that all modules are subject to both integration (functional) and system (unit) testing.

66. The unit testing plan demonstrated a commendable coverage, attaining over 85% of the total codebase at the time of assessment. Whether this level of coverage suffices lies beyond the scope of this evaluation. The examiners found no compelling rationale to mandate near-total coverage (exceeding 99%), as the associated cost and effort would be disproportionate to the benefits. Nevertheless, they observed the absence of a predefined target threshold and noted that the actual coverage metrics were not being actively monitored during the assessment (refer to Section 24.5 – Quality Assurance).