Enhancing the Security of SCA Tool through the Implementation of Technical and Organisational Safeguards

MASTER'S THESIS

Richard Heinz

Submitted on 15 October 2025

Friedrich-Alexander-Universität Erlangen-Nürnberg
Faculty of Engineering, Department Computer Science
Professorship for Open Source Software

Supervisor:
Martin Wagner, M.Sc.
Prof. Dr. Dirk Riehle, M.B.A.

Friedrich-Alexander-Universität
Faculty of Engineering

Declaration of Originality

I confirm that the submitted thesis is original work and was written by me with the assistance of DeepL as a translation tool for linguistic assistance. Appropriate credit has been given where reference has been made to the work of others. The thesis was not examined before, nor has it been published. The submitted electronic version of the thesis matches the printed version.

Erlangen, 15 October 2025

License

Erlangen, 15 October 2025

Abstract

This work focuses on holistically increasing security in the Software Composition Analysis (SCA) tool through the practical implementation of technical and organizational measures. Building on a previous security audit, weaknesses in organizational processes and technical components are addressed, further risks are investigated, and countermeasures are designed.

The security mechanisms were developed on the basis of uniform and standardized requirements. Taking into account the applicable requirements, a wide range of security-related measures was implemented, covering various sub-areas of the SCA tool. The front end was expanded to include fine-grained control over permitted resources and communication channels. Identity management was supplemented with Multi-Factor Authentication (MFA) for the security of user profiles, and a security-conscious configuration for the lifecycle of a session was introduced. In addition to the technical measures, reporting for security-related events was integrated, on the basis of which countermeasures can be initiated immediately. To prevent the possibility of bypasses, care was taken to ensure that each measure is consistently enforced at all architectural levels of the SCA tool.

Overall, the results of this work lead to a robust and maintainable security architecture that specifically supports future extensions of the SCA tool.

Contents

List of Figures

List of Tables

Acronyms

2FA Two-Factor Authentication

AAL Authenticator Assurance Level

AAL1 Authenticator Assurance Level 1

AAL2 Authenticator Assurance Level 2

AAL3 Authenticator Assurance Level 3

AES Advanced Encryption Standard

AiTM Adversary-in-the-Middle

AOP Aspect-Oriented Programming

API Application Programming Interface

APP Applications

ASVS Application Security Verification Standard

ATO Account Takeover

ATT&CK Adversarial Tactics, Techniques & Common Knowledge

BCM Business Continuity Management

BCP Best Current Practice

BIA Business Impact Analysis

BOM Bill of Materials

BSI Bundesamt für Sicherheit in der Informationstechnik

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart

CD Continuous Deployment

CDN Content Delivery Network

CI Continuous Integration

CLI Command Line Interface

CON Concepts and procedures

CORS Cross-Origin Resource Sharing

CSP Content Security Policy

CSRF Cross-Site Request Forgery

CTO Chief Technology Officer

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

CVSSv4 Common Vulnerability Scoring System Version 4.0

CWE Common Weakness Enumeration

DDoS Distributed Denial of Service

DER Detection and response

DOM Document Object Model

DoS Denial of Service

FAU Friedrich Alexander University

HIBP Have I Been Pwned

HTML Hypertext Markup Language

HPP HTTP Parameter Pollution

HSTS HTTP Strict Transport Security

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IAM Identity and Access Management

IAL Identity Assurance Level

IETF Internet Engineering Task Force

IND Industrial IT

INF Infrastructure

IP Internet Protocol

ISMS Information Security Management System

IT Information Technology

IdP Identity Provider

JSON JavaScript Object Notation

KDF Key Derivation Function

LSI Landesamt für Sicherheit in der Informationstechnik

MDN Mozilla Developer Network

MFA Multi-Factor Authentication

MIME Multipurpose Internet Mail Extensions

NET Networks and communication

NIST National Institute of Standards and Technology

NONCE Number used once

OIDC OpenID Connect

OPS Operations

ORP Organization and personnel

OWASP Open Worldwide Application Security Project

PII Personally Identifiable Information

PIN Personal Identification Number

QR Quick Response

RFC Request for Comments

REST Representational State Transfer

ReDoS Regular Expression Denial of Service

RPO Recovery Point Objective

RTO Recovery Time Objective

SAML Security Assertion Markup Language

SBOM Software Bill of Materials

SCA Software Composition Analysis

SIEM Security Information and Event Management

SOP Same Origin Policy

SP Special Publication

SSR Server-Side Rendering

SSO Single Sign-On

SYS IT systems

TLS Transport Layer Security

TOTP Time-based One-Time Password

UI User Interface

URI Uniform Resource Identifier

URL Uniform Resource Locator

UX User Experience

VPN Virtual Private Network

WAF Web Application Firewall

XHR XMLHttpRequest

XSS Cross-Site Scripting

YAML YAML Ain't Markup Language

1 Introduction

Nowadays, software is rarely developed entirely in-house. Instead, applications are based on extensive open-source ecosystems in which frameworks, libraries, and build tools are linked together in dynamic dependencies. SCA addresses the resulting risk. It systematically records the components of an application and evaluates, among other things, compliance with license terms and the presence of known security vulnerabilities. However, for the productive operation of a web application for providing a SCA tool, it is not sufficient to collect findings on external components alone.

On the one hand, the security-related context of the application necessitates strengthening external credibility and acceptance. Even a single security incident due to inadequate security measures can cause users to question the validity of the SCA tool's results. On the other hand, compromised analysis tools jeopardize the confidentiality, integrity, and availability of the application. Analysis results from user organizations can be manipulated and cause real damage. In addition, there are regulatory and liability requirements relating to information security and data protection. Compromising an organization profile in the SCA tool web application exposes sensitive artifacts and integration secrets and, if personal data is processed, can trigger reporting[1] and notification[2] obligations, as well as fines[3] and liability risks. Overall, such an incident not only damages the affected organization, but also permanently undermines trust in the application itself.

This problem constellation gives rise to the obligation to increase the security maturity of the SCA tool as a whole in order to reduce the probability of compromise and the resulting consequences. To this end, requirements are first defined on the basis of guidelines and best practices. Based on these requirements, protective measures are then implemented consistently across the architecture components. The goal in implementing the measures is to implement a combination of complementary mechanisms and, as far as possible, to anchor them organizationally. The evaluation is based on measure-specific criteria and a comparison of the

[1] https://dsgvo-gesetz.de/art-33-dsgvo/
[2] https://dsgvo-gesetz.de/art-34-dsgvo/
[3] https://dsgvo-gesetz.de/art-83-dsgvo/

measures with the requirements. The final step is to summarize the results and classify open issues with a view to further work.

2 Literature Review

This thesis focuses on a practical approach to the technical and organizational security of the SCA tool. It is particularly important that the security measures to be implemented are developed in accordance with established standards and frameworks. This approach reduces the susceptibility to errors during implementation and thus ensures a comprehensive level of protection. Building on this, the following sections present key concepts and reference works for secure software development, including the IT baseline protection of the Bundesamt für Sicherheit in der Informationstechnik (BSI), the Open Worldwide Application Security Project (OWASP) Application Security Verification Standard (ASVS), and the guidelines of the National Institute of Standards and Technology (NIST) publication 800-63.

2.1 Overview of Secure Software Development

When considering the SCA tool in terms of the confidentiality, integrity, and availability of the web application, this results in an increased need for protection. The tool primarily examines source code, dependencies, and libraries for security vulnerabilities, licensing issues, and compliance violations. The high level of protection required with regard to confidentiality arises firstly from the fact that source code often constitutes intellectual property and is therefore strictly confidential. Secondly, the reports highlight particularly sensitive information in the form of software vulnerabilities that could be exploited by an attacker. Manipulation of the SCA tool reports can compromise integrity, jeopardizing an organization's entire security strategy. Restricting availability delays the implementation of an organization's security measures, thereby creating a larger window of opportunity for attack.

This necessitates the addition of security mechanisms based on proven principles to the web application. The BSI recommends that systems be built according to the `Zero Trust` principle. `Zero Trust` is described as an architectural design paradigm based on the principle of least privilege for all entities in the overall infrastructure. Instead of implicit trust, communication between entities should

use a proven and verified basis of trust. Proven trust is intended to minimize the security risk to confidentiality and integrity (Bundesamt für Sicherheit in der Informationstechnik, 2023c).

For the measures to be implemented, this means, on the one hand, that explicit verification and identity checks must be carried out for all affected endpoints that occur in the context of an authenticated request. On the other hand, it must be ensured that, if the framework conditions of the security measure allow, authorizations are strictly limited to the minimum necessary.

The *Secure by Default* principle offers additional protection by ensuring that a system and its components offer the highest possible level of protection right from the initial configuration. Unsecure options should be used consciously and with careful consideration of the resulting risks. From the user's perspective, user profiles and all associated configurations must offer a high level of security immediately after profile creation. This principle minimizes the risk of human misconfiguration and ensures consistent security across all deployment environments.

Building on this, *Defense in Depth* extends the secure initial configuration with several independent controls along each layer of the architecture. This means that the failure of individual security measures does not immediately compromise other system components.

The layers comprise technical components, organizational processes, and physical conditions. Securing the technical components focuses on the introduction of security measures at the network, system, application, and data levels. Security precautions for organizational processes include structured monitoring for incident detection and security policies to embed security in operational procedures. Ensuring secure physical conditions refers to measures to prevent unauthorized physical access to systems requiring protection.

Overall, these security principles should be applied, taking into account the context of the measures to be implemented, in order to ensure comprehensive protection of the SCA tool.

2.2 BSI IT Baseline Protection

The IT baseline protection provided by the BSI offers a comprehensive, practical approach to the systematic protection of information networks. The core components of IT baseline protection include the BSI standards, the IT baseline protection compendium, and the IT baseline protection profiles.

2.2.1 BSI-Standards

The BSI standards are a fundamental component of IT baseline protection and comprise standards `200-1` to `200-4`. The standard `200-1`[1] describes the basics for setting up, operating, monitoring, and improving an Information Security Management System (ISMS). It therefore describes methods for initiating and controlling information security in an institution.

Building on this, the `200-2`[2] standard addresses the methodology for effective information security management and offers concrete assistance. In addition to other aspects of ensuring information security, it describes three key approaches. These approaches include *basic*, *core*, and *standard* security. *Basic* security is an approach that aims to first achieve fundamental initial security across all relevant business processes of an institution. This approach is particularly preferable if the implementation of information security has a low level of maturity and there are no assets whose compromise would pose an existential threat to the institution. *Core* security is based on the principles of *basic* security, but focuses on protecting a manageable number of business processes with increased protection requirements. Furthermore, in the context of *core* security, the BSI describes that clearly identifiable assets must be present whose theft, destruction, or compromise would cause damage that threatens the existence of the institution. Minor security incidents with monetary or other damage are accepted in *core* security.

The long-term approach is described by the *Standard* security level. This should be selected if the institution already works with IT baseline protection and security concepts have already been implemented in accordance with this. In addition, the *Standard* security level is particularly suitable if the implementation of information security has already reached a sufficient level of maturity, so that no fundamental initial security measures are necessary in key areas (Bundesamt für Sicherheit in der Informationstechnik, 2017a).

The standard `200-3`[3] supplements the security measures with a risk analysis based on IT baseline protection. The standard is based on `200-2` and serves to identify residual risks that are not covered or only partially covered by basic protection.

Finally, the BSI standard `200-4`[4] includes Business Continuity Management (BCM), which is intended to ensure that an organization remains capable of act-

[1] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/BSI_Standards/standard_200_1.html?nn=128578

[2] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/BSI_Standards/standard_200_2.html?nn=128640

[3] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/BSI_Standards/standard_200_3.html?nn=128620

[4] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/BSI_Standards/standard_200_4.html

ing even in crisis situations and that critical business processes are maintained. The primary content describes the structure of an emergency management process, the execution of a Business Impact Analysis (BIA), and a risk analysis for emergencies. Guidance is provided on how to set up an emergency management process, how it can be integrated into the organizational structure, and how roles and responsibilities should be defined.

The BIA enables the identification of critical business processes and the length of time an organization can operate without these processes. In addition, the Recovery Time Objective (RTO) defines the maximum tolerable downtime of the system in order to determine the period within which business operations must be restored. Moreover, the Recovery Point Objective (RPO) is used to determine the maximum tolerable amount of data that may be lost in a crisis without causing significant damage to the organization. It serves as the basis for determining the required frequency of data backups in order to minimize potential data loss.

Due to the practical nature of this work, the BSI standard `200-2` and the associated IT Baseline Protection Compendium are of particular importance. The following section therefore takes a closer look at the IT Baseline Protection Compendium and, in particular, the components relevant to the architecture of the SCA tool.

2.2.2 IT Baseline Protection Compendium

The IT Baseline Protection Compendium contains a collection of thematically structured modules, each of which describes the risks and security requirements. The modules are process- and system-oriented and can be sorted into ten layers. Figure 2.1 shows the structure of the modules in the IT Baseline Protection Compendium.

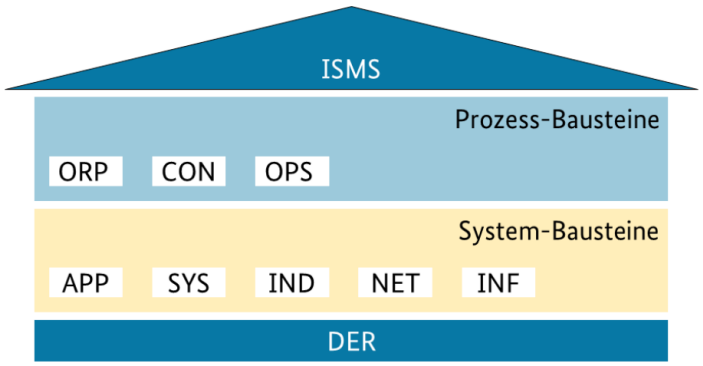

Figure 2.1: Layered model of the IT baseline protection components.

Source: (Bundesamt für Sicherheit in der Informationstechnik, 2023a).

The layer model can be subdivided into the ISMS layer, process and system components with thematically structured sublayers, and the Detection and response (DER).

The ISMS maps the management and control processes of information security. The building blocks of the ISMS layer address central management tasks, such as establishing security guidelines, defining roles and responsibilities, and providing the necessary resources.

The process modules comprise the layers Organization and personnel (ORP), Concepts and procedures (CON), and Operations (OPS). ORP addresses organizational and personnel security aspects. These include governance, role management, training and awareness-raising, and the definition of guidelines for specific procedures.

Building blocks for the CON layer cover concepts and procedures related to this topic. Among other things, data protection procedures, data backup concepts, and procedures for deleting and destroying data are presented. In addition, `CON.10` defines the risks associated with web applications and specific requirements for the development of web applications. The requirements are based on the selected procedure from the BSI standard `200-2`. For *basic* security, standardized requirements for authentication, access control, and secure session management for web applications are specified. The requirements for long-term *standard* security include secure software architecture for web applications, secure Hypertext Transfer Protocol (HTTP) configuration, prevention of Cross-Site Request Forgery (CSRF), and many more.

The process layer is completed by the OPS components, which contain the building blocks for security aspects of operational activities. The scope of the building blocks covers the operation of the organization's own unit, the use of third-party applications, and the outsourcing of business processes or activities to external service providers. The focus of the building blocks is on ensuring the confidentiality, integrity, and availability of business processes during ongoing operations through the use of standardized and repeatable processes.

System components include the layers Applications (APP), IT systems (SYS), Industrial IT (IND), Networks and communication (NET), and Infrastructure (INF). The scope of the APP layer covers the use of specialized and standard software as well as application services. The APP layer provides specific requirements for client applications, directory and network-based services, and business applications, among other things. Of particular relevance to the SCA tool architecture in this context is the building block "`APP.3.1`", which defines requirements for protection against unauthorized automated use, protection of confidential data, and many others within the framework of *basic* security. The *Standard* security for this building block additionally provides for more detailed

requirements in the form of secure configuration and secure connection of background systems.

The SYS layer of system components comprises components for securing servers, desktop systems, and mobile devices. This layer focuses on hardening systems through secure configurations, protection against the exploitation of vulnerabilities, and encrypted communication. IND is used to specify building blocks for Information Technology (IT) in an industrial context. The building blocks primarily include threats and requirements for process control and automation technology and production networks.

The NET layer is designed to ensure the confidentiality, integrity, and availability of network communications at all levels. Its building blocks relate to the protection of both internal networks and external connections. These system building blocks are completed with the INF layer, which forms the physical basis for the entire IT landscape. The building blocks of this layer identify infrastructural threats and define security requirements to ensure the protection of buildings, IT cabling, power supplies, and many other physical infrastructures (Bundesamt für Sicherheit in der Informationstechnik, 2023a).

The final layer is DER, which deals with the procedures for detecting, classifying, and effectively handling security-related incidents. It comprises components for security incident management, which in turn specify requirements for handling incidents, provisions for IT forensics, and the resolution of far-reaching security events. In addition, the threat situation and the requirements for vulnerability assessments and emergency management are defined.

2.2.3 IT Baseline Protection Profiles

IT baseline protection profiles are implementation guidelines that are preconfigured and adapted to specific scenarios or industries. The profiles bundle building blocks and requirements from the compendium for typical organizational contexts and aim to facilitate the introduction of an ISMS. This thesis focuses on product-related validation of the SCA tool, with the primary focus on component- and verification-oriented catalogs.

Therefore, the following section takes a closer look at the OWASP Application Security Verification Standard, which provides specific requirements and testing objectives for web applications.

2.3 OWASP ASVS

The OWASP ASVS provides a catalogued set of requirements for verifiable web application security. Version 5 of the ASVS was released in May 2025, with

various changes made in relation to version 4. On the one hand, new sections were added to ensure more detailed application security. On the other hand, the trust levels were adjusted to make it easier to get started with application security. The standard is divided into 17 security categories and lists three trust levels.

Before the security categories can be discussed, it is necessary to understand the security levels of the ASVS. The security levels are structured in level, with the first level serving to secure processes classified as critical. This level contains the minimum requirements that must be taken into account in order to establish a basic layer of protection in the context of the ASVS, on the basis of which the subsequent levels can be achieved.

The second level describes the implementation of all requirements defined in the ASVS for the second level. This level already provides a solid layer of protection, with various preconditions that must be met for an attack to be successful. However, the strongest layer of protection is represented by the third level, which describes requirements that provide enhanced security for the application. This can be achieved, for example, in the form of a multi-layered security architecture or additional controls.

It is important to note that the requirements specified in the ASVS are intended for a specific security level based on their level of detail and their security impact. In order for the security measure to be traceable, the reference to a specific security requirement for a level requires an explanation of the specific measure and the context in the SCA tool. Therefore, the essential security categories with their general effect are presented below and a contextual reference is established within the framework of development (see chapters 4 and 5).

Of particular relevance to this work are the categories `Web Frontend Security`, `Authentication`, `Session Management`, `Cryptography`, and `Configuration`.

The security category `Web Frontend Security` defines requirements for the documentation and secure delivery of data to the client. In addition to many other requirements, this includes the need for the secure use of cookies, the enforcement of security-related headers, and the validation of cross-origin requests. The focus of this category is on ensuring the integrity and trustworthiness of the delivered code. Based on the selected security level, requirements are specified in varying degrees of detail to prevent manipulation of the Document Object Model (DOM) in order to prevent typical attack scenarios such as Cross-Site Scripting (XSS) and code injection.

`Authentication` addresses the identity verification of an application and the entire login lifecycle. The requirements discussed therein include specifications for password policies, the implementation of MFA, and the creation of robustness against attacks on the authentication process. In addition, specifications are

made depending on the desired level of logging of authentication attempts without revealing sensitive information.

After a user has authenticated themselves, a session is created for them. This serves as explicit proof of identity for authenticated requests within the web application. To ensure that this session exists in a secure context, protective measures are defined in the `Session Management` category of the ASVS. The first step is to define fundamental requirements, such as the correct verification of all session tokens using a secure backend service. In addition, requirements for the lifecycle of a session are defined. This includes the implementation of session timeout handling and the implementation of correct session termination, including all associated implicit consequences.

The security category `Cryptography` deals with aspects of cryptography to ensure the confidentiality, integrity, and authenticity of data in applications. The primary aim is to eliminate known vulnerabilities and the use of outdated methods. With regard to the SCA tool, the requirements for the use of secure encryption algorithms and cryptographic hash functions are of central importance. A specified requirement describes the use of an approved encryption algorithm such as Advanced Encryption Standard (AES) with secure modes. Requirements relating to the hash functions used include the use of approved hash functions and their correct configuration (OWASP Foundation, 2025b).

The security category `Configuration` and the requirements for configuring backend communication in combination with the requirements for handling secrets are particularly important for the SCA tool architecture. Specifications are defined to ensure that all communication between all backend components must take place in an authenticated form. Furthermore, requirements are specified that enable the secure creation, storage, management, and deletion of backend secrets.

In summary, the ASVS provides a practical and verifiable catalog of requirements that offers specifications for increasing the overall security of a web application. In addition, it enables the measurement of the security level and the introduction of appropriate tests that facilitate the validation of the security measures implemented. The revision published in May 2025 lowered the entry barrier and made the requirements more precise. This makes the implementation of the security requirements more accessible and increases the effectiveness of the controls.

While the ASVS addresses technical controls, it is necessary to consider NIST 800-63 in order to identify the context-specific configuration parameters that enable the desired level of security to be achieved.

2.4 NIST SP 800-63

NIST Special Publication (SP) 800-63 deals with digital identity guidelines and is a set of rules describing how digital identities should be managed, authenticated, and validated. The guideline consists of four documents that shed light on different phases of identity management.

The first document provides an overview of the phases of identity management and references individual aspects of the subsequent documents. The core document describes the overall context and does not specify any technical requirements. For this reason, the focus is on the following documents, which define specific security requirements for identity verification and authentication.

The second document is NIST SP 800-63A, which describes enrollment and identity proofing. It focuses on specific requirements for identity verification, in which applicants must prove their identity to the credential service provider and how the proof is evaluated. In the context of a web application, this means that the specifications of this document primarily concern the processes for registration and login. For the purpose of classifying the security of identity verification, three Identity Assurance Level (IAL) are defined. IAL1 specifies that it is not necessary to link the applicant to a real identity and that all attributes resulting from the subject's activities must be treated as self-asserted. IAL2 represents an identity verification in which the evidence proves the real existence of the stated identity and confirms that the applicant is associated with the real identity. The highest level is IAL3, where identity verification requires physical presence (Grassi et al., 2017). The purpose of IAL is therefore not to strengthen authentication endpoints, but to prove the authenticity of an identity by validating that the identity is actually assigned to a specific natural person.

Authentication and lifecycle management is addressed in the third document, NIST SP 800-63B. This document focuses on strengthening processes in the context of verifying an existing identity. It describes different forms of authentication and classifies them on the basis of three Authenticator Assurance Level (AAL) levels. AAL describes how secure the authentication method itself is and is not based on the correspondence between the identity and the natural person. In detail, Authenticator Assurance Level 1 (AAL1) describes authentication methods that use a single factor. This includes authentication via a password or Personal Identification Number (PIN) as well as via an email-based link for login. Authenticator Assurance Level 2 (AAL2) provides a high level of security that the applicant controls one or more authentication factors linked to the profile used for authentication. In addition to the first factor in the form of AAL1, AAL2 requires a second physical factor for successful authentication. One possible combination of AAL2 results from the use of the profile password, which is based on knowledge, in combination with a one-time password from an external

authenticator device. Thus, the knowledge for the first factor is supplemented by the possession of the authenticator device for the second factor. These independent factors require different attack paths and therefore make it possible to significantly reduce the attack surface on the user profile (Temoshok et al., 2025).

The third AAL offers a very high level of security that the applicant has control over the authentication means associated with the user account. Authenticator Assurance Level 3 (AAL3) is achieved when the underlying requirements are met. This assumes that at least two factors are active and that the second factor comes from a hardware-based authenticator. In addition, the private key must be device-bound and the authentication must be phishing-resistant. Overall, document SP 800-63B defines detailed requirements based on these levels for the entire authentication process.

The final document SP 800-63C addresses federated authentication. It defines requirements for authentication via Identity Provider (IdP) that are intended to ensure that the integrity and confidentiality of the transmitted data is maintained. In addition, the defined requirements ensure that the user only discloses the necessary sensitive data within the scope of this form of authentication.

In summary, NIST SP 800-63 is particularly relevant for securing the architecture components of the SCA tool, as it represents an internationally recognized standard for evaluating and implementing secure identity and authentication mechanisms. The definition of levels also makes it possible to classify security levels consistently and to align technical and organizational measures with objective criteria.

3 Requirements

This chapter specifies the requirements for the security-enhancing measures for the SCA tool, which are to be implemented at an organizational and technical level. The specification of the requirements is independent of the underlying technology and is derived from (i) the results of the previous safety analysis, (ii) the relevant standards and best practices, such as BSI IT baseline protection, OWASP ASVS and the NIST SP series.

For each requirement, an additional obligation is defined in accordance with Best Current Practice (BCP)-14[1]. BCPs are documents published by the Internet Engineering Task Force (IETF) on recognized procedures, guidelines and standards for the Internet. BCP-14 defines the keywords described in Table 3.1 in order to ensure a clear and uniform interpretation of requirements.

Term	**Explanation**
MUST	Describes a mandatory requirement that must be complied with without exception.
MUST NOT	Describes a mandatory prohibition, whereby the behavior described must not occur under any circumstances.
SHOULD	Describes a recommendation for implementation and emphasizes that there may be legitimate reasons for non-implementation. Careful consideration of the full implications is required.
SHOULD NOT	Describes a non-recommendation for implementation and emphasizes that there may be legitimate reasons for implementation. Careful consideration of the full impact is required.
MAY	Describes an optional requirement where the behavior is permitted but not mandatory.

Table 3.1: Normative keywords according to BCP-14 (RFC 2119 / RFC 8174)
Note: According to RFC 8174, the normative meaning applies only when the keywords are in *UPPERCASE*.

[1] https://www.rfc-editor.org/info/bcp14

The terminology in Table 3.1 forms the basis for the specific definition of the following requirements. First, a reference to the vulnerabilities identified in the previous work should be established and criteria for the evaluation formalized. The analysis is then extended to include additional risk factors. Finally, it is determined how results should be documented, prioritized, and implemented.

3.1 Analysis of identified vulnerabilities

The previous thesis identified security-related deficiencies in organizational procedures, the use of external tools, and the SCA tool implementation itself. These findings were supplemented by possible solutions for the specific application context (Rosenberger, 2025). Within the scope of this thesis, this results in the following security requirements:

R-1 Evaluation of the identified vulnerabilities
All vulnerabilities identified by Rosenberger **MUST** be checked for their current status and exploitability. The risk posed by them **SHOULD** be re-verified at the time of reevaluation.

R-2 Practical evaluation of the approaches
The proposed solutions **MUST** be reviewed for actual added security value. In addition, the solutions **MUST** be examined for feasibility, taking into account the avoidance of usability or functionality restrictions. In the event of a restriction, alternative implementation options **SHOULD** be considered.

3.2 Identification of additional vulnerabilities

Beyond the findings documented in Rosenberger's work, the SCA tool must be continuously examined for additional technical and organizational vulnerabilities. These can be logically derived from existing findings, result from insecure processes in the development process, or lie in security-critical usage flows themselves. This results in the following security requirements:

R-3 Authentication and account lifecycle flows
Security-critical usage flows such as *login*, *registration*, *identity verification*, *password reset/recovery*, and *session management* **MUST** be checked for common attack vectors and hardened against them.

R-4 Abuse at critical endpoints
Critical endpoints, especially those that can be reached without authentication, **MUST** be examined for their susceptibility to abuse. Vulnerabilities **MUST** be documented and **SHOULD** be prioritized based on risk assessment.

3.3 Documentation and Traceability

To ensure traceability and transparency with regard to the identification and remediation of security vulnerabilities, detailed documentation of the identified vulnerabilities is required. This documentation should also serve to assess the risk and prioritize the vulnerabilities so that appropriate measures can be taken in a timely manner. Furthermore, documenting security vulnerabilities enables the steps taken and measures implemented to be verified. This results in the following requirements in terms of documentation and traceability:

R-5 Traceability and transparency
Every security vulnerability and every security-related change **SHOULD** be recorded as a GitHub issue. If prioritization is carried out, the closure of the security vulnerability **MUST** be traceable, for example via a pull request and commits. The security vulnerability **MAY** be released for discussion within the scope of the issue or via secure, alternative means of communication.

R-6 Risk assessment and prioritization
The identified vulnerabilities **MUST** be assessed based on the probability of occurrence and the potential damage (Rosenberger, 2025). For high and very high risks, a risk treatment concept **MUST** be developed immediately that excludes risk acceptance. For medium and low risks, risk treatment **SHOULD** be carried out in order of priority, which may also include risk acceptance under certain circumstances.

3.4 Development of security solutions

Based on the identified security risks, it is necessary to design appropriate solutions and implement existing approaches. The following requirements should be taken into account:

R-7 Implementation of prioritized findings
The vulnerabilities prioritized during the preliminary work and reassessment **MUST** be remedied using appropriate technical and organizational measures. When developing solutions, care **SHOULD** be taken to address the root cause. In addition, the principles of *Secure by default*, *Least privilege*, and *Defense in depth* **SHOULD** be followed in accordance with the BSI IT-Grundschutz and the OWASP ASVS.

R-8 Maintainability, configurability, and efficiency
The security solutions to be developed **MUST** be modular, reusable, and configurable for specific environments. In addition, they **SHOULD** be de-

signed generically so that security is maintained as development progresses. The application **SHOULD NOT** be impaired by the implemented security solutions in terms of User Experience (UX).

R-9 Monitoring and measurement
If applicable to the specific context, appropriate monitoring and measurement of both effectiveness and violations **SHOULD** be performed. Furthermore, the performance of the endpoints affected by the security measure **SHOULD** be assessed and, if necessary, optimized.

R-10 Explicit definition
Default values for security-related configurations **SHOULD** be explicitly stated in order to raise awareness about the configuration.

Based on these requirements, the security solutions will be designed and implemented in the following chapter.

4 Technical Safeguards

This chapter presents the technical security measures designed to reduce the attack surface of the SCA tool architecture. Based on the requirements derived in Chapter 3, it addresses vulnerable dependencies, authentication and the lifecycle of a session, header-based protection measures, and browser-side defense mechanisms. In addition, the measures are evaluated for functional side effects. Overall, the measures should lead to comprehensive security for the SCA tool and not result in any restrictions on use.

4.1 Outdated and Vulnerable Libraries

Rosenberger identified various outdated and vulnerable software components in the context of the `fecf785`[1] commit (as of 2024-11-09) (Rosenberger, 2025). Since the SCA Tool has evolved functionally and in terms of its dependencies since then, the findings at that time can only be interpreted as a snapshot. The aim of the following investigation is therefore to create a new baseline and to evaluate the libraries at the current evaluation status.

For this purpose, a CycloneDX Software Bill of Materials (SBOM) is generated for the current analysis time (2025-08-12) based on the commit `b65f530`[2] , evaluated in a self-hosted instance of Dependency-Track and the resulting findings verified. The verification includes the differentiation of direct versus transitive dependencies, the comparison of the Common Vulnerabilities and Exposures (CVE) entries with the primary sources[3] and the prioritization according to the latest available Common Vulnerability Scoring System (CVSS) version. This creates a comparable and reproducible basis for decision-making, which is used to systematically document resolved, existing and new risks and address them through upgrades or mitigations.

[1] Full commit: `fecf785a2888e7f8f062975859c6d017185cfae3` (view).
[2] Full commit: `b65f530581737badb947e34a805ea61a22e31580` (view).
[3] https://www.cve.org/

4.1.1 Rosenbergers Security Audit: Outdated and vulnerable libraries

To assess the relevance of the vulnerabilities identified in the previous audit, the current project status was checked. Table 4.1 lists the identified vulnerabilities in the form of a list of packages, the package version, the identified CVE, and the vulnerability rating based on CVSS. The fifth column classifies each CVE in relation to the current dependency status.

Package	Version	CVE	CVSS	Status (2025-08-12)
`lilconfig`	2.1.0/3.0.0	CVE-2024-21537	8.8 (HIGH, v3.1)	Upgraded (3.1.3)
`libxmljs2`	0.33.0	CVE-2024-34393	8.1 (HIGH, v3.1)	Not affected
`libxmljs2`	0.33.0	CVE-2024-34394	8.1 (HIGH, v3.1)	Not affected
`braces`	3.0.2	CVE-2024-4068	7.5 (HIGH, v3.1)	Upgraded (3.0.3)
`cross-spawn`	7.0.3	CVE-2024-21538	7.5 (HIGH, v3.1)	Upgraded (7.0.6)
`next`	14.1.4	CVE-2024-46982	7.5 (HIGH, v3.1)	Upgraded (15.4.5)
`next`	14.1.4	CVE-2024-47831	7.5 (HIGH, v3.1)	Upgraded (15.4.5)
`cookie`	0.6.0	CVE-2024-47764	6.9 (MED, v4.0)	Upgraded (1.0.2)
`railroad-diagrams`	1.0.0	CVE-2024-26467	6.1 (MED, v3.1)	Not present
`rollup`	4.21.0	CVE-2024-47068	6.1 (MED, v3.1)	Upgraded (4.35.0)
`micromatch`	4.0.5	CVE-2024-4067	5.3 (MED, v3.1)	Upgraded (4.0.8)
`nanoid`	Not specified	CVE-2024-55565	4.3 (MED, v3.1)	Upgraded (3.3.11)

Table 4.1: Weaknesses identified in the pre-audit (Rosenberger) with associated CVE and CVSS

The vulnerability `CVE-2024-21537` of the package `lilconfig` allows attackers to execute arbitrary code due to the insecure use of the JavaScript function `eval` within the `dynamicImport` function. This is a particularly serious vulnerability which, in combination with the fact that two different versions of the same package are installed, increases the risk profile. At the current state of implementation, the outdated package (2.1.0) has been uninstalled and the version of the remaining package has been upgraded to version `3.1.3`. This version is above the patch threshold (3.1.1) and is shown in the dependency track with a risk score of zero and also no vulnerabilities.

The findings on the two vulnerabilities for the `libxmljs2` package describe two severe security vulnerabilities with which an attacker can carry out a Denial of Service (DoS) attack or remote code execution. However, these vulnerabilities do not result from the application logic, but from the toolchain for SBOM generation using `cylonedx`. As this package is not included in the productive artifacts, the entry is classified as *Not affected*. For subsequent analyses, the SBOM is generated via npx so that toolchain packages no longer appear in the Bill of

Materials (BOM).

The vulnerability CVE-2024-4068 of the package braces noted in the previous audit describes an uncontrolled resource consumption due to unbalanced brackets as input. In the current implementation status, only braces 3.0.3 is installed and Dependency-Track provides a risk score of 0 and 0 vulnerabilities. According to CVE-2024-4068 this version is not affected by the vulnerability and can be classified as *Upgraded (3.0.3)*.

CVE-2024-21538 describes a vulnerability in the package cross-spawn which makes the system vulnerable to Regular Expression Denial of Service (ReDoS). This vulnerability affects the version interval >=7.0.0 and < 7.0.5. The currently installed version is 7.0.6 and is therefore no longer affected by the vulnerability. Dependency-Track also has a risk score of 0 and 0 vulnerabilities.

The vulnerability CVE-2024-47831 of the package Next describes a vulnerability in the image optimization, which can be exploited for a DoS attack and was already solved in the version 14.2.7. In addition, the package had the vulnerability CVE-2024-46982, which allows cache poisoning in non-dynamic Server-Side Rendering (SSR) routes in pages router. This vulnerability has been fixed with version 14.2.10. Currently, Next in the SCA Tool application has the version 15.4.5 and has a risk score of 0 and 0 vulnerabilities according to the dependency track.

In the context of CVSS scores for MEDIUM severity vulnerabilities, CVE-2024-47764 for the package cookie has been fixed by upgrading to version 1.0.2. The package railroad-diagrams is no longer part of the dependencies and therefore does not pose a threat to the SCA Tool. The package rollup in version 4.35.0 no longer contains the vulnerability CVE-2024-47068. The package micromatch was mitigated by upgrading to version 4.0.8. Finally, Dependency-Track for the package nanoid in version 3.3.11 had a risk score of 0 and 0 vulnerabilities.

4.1.2 2025-08-12: Analysis of outdated and vulnerable libraries

After evaluating the vulnerabilities identified in the security audit, it is necessary to identify the components currently marked as obsolete or vulnerable. The following table 4.2 summarizes all components currently classified as vulnerable. For each item, the technical cause of the problem, the affected version span and a risk classification are then explained.

form-data - CVE-2025-7783 (CRITICAL, CVSSv4.09.4). The library generates multipart boundary values with Math.random(), which allows predictability and thus favors HTTP Parameter Pollution (HPP). Affected package versions are <2.5.4, 3.0.0 - 3.0.3 and 4.0.0 - 4.0.3. Robust versions in which the

Package	Version	CVE	CVSS	Status (2025-08-12)
form-data	4.0.2	CVE-2025-7783	9.4 (CRIT., v4.0)	Upgraded (4.0.4)
webpack-dev-server	4.15.2	CVE-2025-30359	5.3 (MED., v3.1)	Upgraded (5.2.1)
webpack-dev-server	4.15.2	CVE-2025-30360	6.5 (MED., v3.1)	Upgraded (5.2.1)
express	4.21.2	CVE-2024-10491	5.3 (MED., v3.1)	False Positive
http-proxy-middleware	2.0.7	CVE-2025-32996	4.0 (MED., v3.1)	Upgraded (3.0.5)
http-proxy-middleware	2.0.7	CVE-2025-32997	4.0 (MED., v3.1)	Upgraded (3.0.5)
on-headers	1.0.2	CVE-2025-7339	3.4 (LOW, v3.1)	Upgraded (1.1.0)
brace-expansion	2.0.1	CVE-2025-5889	2.3 (LOW, v4.0)	Not affected

Table 4.2: Currently identified vulnerabilities (Dependency Track, August 12, 2025)

vulnerability has been fixed are `2.5.4` and `4.0.4`. `form-data` is transitively included in the codebase via Axios. The affected functionality concerns node-side multipart/form-data requests. Within SCA Tool, only `fetch` requests with a JavaScript Object Notation (JSON) body are transferred, thus the likelihood of exploitation in this context is negligible. Regardless of this, a preventive update to a patched version is essential in order to avoid opening a security gap in future implementations.

webpack-dev-server - CVE-2025-30359/30360 (MEDIUM). Under certain conditions, both findings enable source code to be exfiltrated from development environments. `CVE-2025-30359` is based, among other things, on the handling of the `toString` method in connection with classic script requests. `CVE-2025-30360` results from insufficient validation of the `Origin` header in WebSocket connections. Both vulnerabilities have been fixed as of version `5.2.1`. For the SCA tool, the practical risk is considered low, as `webpack-dev-server` is only used in the development environment and is not used in production. A residual risk exists if a malicious website is visited in parallel while a development server is running, thereby exposing source code. An escalation would be possible if the exposed code contains further vulnerabilities that can be combined to form a chain of attacks. An update to `5.2.1` further reduces the remaining risk.

express - CVE-2024-10491 (MEDIUM). Insufficient sanitization in "`response.links()`" allows resources to be injected into the link header. In particular, a combination of characters such as commas, semicolons, or tags is allowed, which can be used to construct malicious resources. However, this package is a false positive on the part of Dependency Track, as versions `3.0.0-alpha1 - 3.21.2` are identified as affected in `CVE-2024-10491`. In contrast, the component currently used in the SCA tool is version `4.21.2`.

http-proxy-middleware - CVE-2025-32996/32997 (MEDIUM). Due to a control

flow error, the `writeBody` function can be called twice, which can lead to protocol inconsistencies (32996). In addition, despite a failed `bodyParser` call, the `fixRequestBody` function continues to run, which could potentially compromise integrity (32997). The vulnerability `CVE-2025-32996` was fixed in version `2.0.8` and `3.0.4`, and `CVE-2025-32997` was fixed in `2.0.9` and `3.0.5`. To maintain functionality, the current version `2.0.7` should be upgraded to `2.0.9` in the short term and to the latest version `3.0.5` in the long term.

on-headers - CVE-2025-7339 (LOW). Headers can be modified when passing an array parameter to the `response.writeHead()` function. Due to the requirement that high privileges must be granted and that this is a local attack vector, the vulnerability poses a low risk. Nevertheless, upgrading to version `1.1.0` increases the overall security of the system.

brace-expansion - CVE-2025-5889 (LOW). In the function `expand`, inefficient handling of regular expressions allows ReDoS attacks to be carried out. In the SCA tool, the vulnerability occurs transitively in `openapi-typescript`, `@redocly/openapi-core`, and `eslint`. Productive paths already use `brace-expansion` in version `2.0.2`, which is not affected by the vulnerability.

4.2 Secure Password Policy

According to `NIST SP 800-63B Rev. 4`, a minimum length of 15 characters *MUST* be required for password-based single-factor authentication, and when used as part of multi-factor authentication, the length *MAY* be reduced to a minimum of 8 characters, although longer passphrases and a maximum length of at least 64 characters are recommended (Temoshok et al., 2025). In accordance with OWASP guidelines, passwords should also be checked against known compromises (OWASP Foundation, 2025a). Ory Kratos provides native configuration parameters for this purpose, which are shown in the table 4.3.

Environment	**min password length**	**HIBP**	**ignore network errors**	**identifier similarity**
Local Dev	8	False	`true`	False
Dev	8	False	`true`	False
Staging	15	True	`false`	True
Testing	15	True	`false`	True
Production	15	True	`false`	True

Table 4.3: Password policy per environment

The table describes the recommended password policy, taking into account the

deployment environment. The field `min_password_length` defines the minimum permissible length of a password for the application, while `Have I Been Pwned (HIBP)` defines whether the password should be checked for potential compromises. The parameter `ignore_network_errors` is crucial for the availability of various functions of the application and determines how the flow should continue if a network error occurs during the compromise check. If this parameter is set to *False*, a network error during HIBP validation immediately results in a failed flow. In the context of registration, this error would prevent a user from creating a profile. Furthermore, it is possible to use `max_breaches` to specify how often this password must be found in data leaks before Kratos rejects it as insecure. In addition, Ory Kratos offers the parameter `identifier_similarity_check_enabled`, which prevents the setting of a password that has an impermissible similarity to the user ID.

A restrictive password policy is not necessary for the local development environment or the development cluster. It would create additional external dependencies without achieving any security gains in these contexts. Since only test profiles are used here and no productive data, HIBP validation and similarity checks are disabled. This avoids external service dependencies and security-related validation interruptions. In addition, the configuration has been made more robust against temporary network disruptions by ignoring network errors. Similarity checks in regard to the identifier remain disabled so as not to unnecessarily interfere with the development flow.

In production, restrictive configuration should be enforced in accordance with the recommendations in order to minimize the risk posed by compromised or trivial passwords. This means that the password length should be increased to 15 characters, HIBP validation should be performed, the flow should be interrupted in the event of network errors, and similarity checks should be carried out. Although these security settings represent a strong password policy, they also reduce the usability of the application and increase user frustration (Komanduri et al., 2011). Strict composition rules lead users to use sticky notes, schema replacements, and other coping strategies, which in turn reduce security. If, instead of a hard rejection, the user is provided with security-relevant information and has the power to make decisions, they tend to choose a stronger password. The visual representation of the password strength indicator has no substantial influence on password strength (Ur et al., 2012).

Although `NIST SP 800-63B` and the OWASP guideline recommend a restrictive password policy, enforcement is not activated within the scope of this thesis. This is due to a conscious product decision in favor of UX. In detail, it must be decided whether enforcement should take place and, if so, the availability of the HIBP service must be ensured so that there are no flow-blocking failures. If a decision is made against enforcement, a concept for non-blocking user notifications

and accompanying processes must be developed. A corresponding enforcement configuration remains documented in accordance with the guidelines (see Issue #1453, Draft Pull Request #1454). Implementation is listed as a roadmap item and requires a UX evaluation and the definition of an integration variant.

4.3 Secure Logging Configuration

In the context of identity management, minimising sensitive log content is a key security principle. The Ory Kratos documentation explicitly emphasises that the `log.leak_sensitive_values` option should always be set to *false* (Ory Corp., 2025c). Otherwise, there is a risk that Personally Identifiable Information (PII), secrets, tokens and other sensitive information will end up in logs. Temporary activation is only acceptable for strictly limited, local debugging scenarios and only if the output is not stored or forwarded. Taking into account the OWASP ASVS, which requires that applications do not log credentials or other sensitive information (OWASP Foundation, 2025b), it is necessary to keep the logging of such sensitive information to a minimum. The logging configuration for each deployment environment, as illustrated in table 4.4, was identified in the context of Ory Kratos' identity and user management

Environment	`log.level`	`log.format`	`leak_sensitive_values`
Dev local	`trace`	`json`	`true`
Dev cluster	`debug`	`text`	`true`
Testing	`info`	`text`	`false`
Staging	`info`	`text`	`false`
Production	`info`	`text`	`false`

Table 4.4: Ory-Kratos log configuration per deployment environment

The table shows that `leak_sensitive_values` is set to `true` for the local development and the development cluster, while *Testing*, *Staging* and *Production* are correctly configured to `false`. In addition, `log.level` in *Dev local* (trace) and *Dev cluster* (debug) is significantly more verbose than in the other environments, which increases the probability of occurrence and the impact of information leaks. `Trace` and `debug` increases the level of detail, and even if only test users are used in the development environments, there is a risk of revealing real secrets, such as OpenID Connect (OIDC) client secrets, Application Programming Interface (API) keys or internal infrastructure details, which could be misused if disclosed. Pseudonymous test data does not reliably rule out the possibility of re-identification.

Against this background, a binding baseline is defined in which the `log.leak_sensitive_values` option is set to `false` in every versioned configuration. Temporary activation is only permitted for limited, local debug sessions without log forwarding or persistence of the information. Once this session is complete, the configuration must be immediately reverted. To ensure this reversal, a Git Pre-commit hook is used that strictly rejects commits with `leak_sensitive_values` set to `true` in versioned files.

The Git pre-commit hook can be bypassed by a developer through deliberate or inadvertent misconfiguration and therefore merely serves as an early warning that active logging of sensitive information within the repository must be disabled. To enforce this rule, an additional job called *Security Policy* has been integrated into the Continuous Integration (CI) workflow, which is mandatory for merging. This job examines repository files that contain a Kratos-specific configuration setting that could lead to the disclosure of sensitive information. If such a configuration is identified, the job fails and a merge into the main branch is prevented.

The *Security Policy* is defined as a standalone job within the workflow so that code-based security policies can be validated automatically and flexibly expanded in line with future requirements.

4.4 Multi-Factor Authentication

As part of the previous security audit (Rosenberger, 2025), a serious gap was identified due to the lack of MFA in the login processes of the SCA tool. In the current implementation, the level of protection of the authentication endpoint is limited to the complexity of the password for a user ID. The user ID is the user's email and is therefore public. If the password is not sufficiently complex, an automated brute force attack is sufficient to gain access to the user profile and its authorizations.

In order to minimize the risk of such attacks and reduce the attack surface, it is necessary to secure the authentication of the user with a second factor. This means that even a successful attack on a user's password would fail due to the absence of the second factor.

4.4.1 Two-Factor Authentication Setup

When designing the security measure, it is necessary to take the user's workflow into account, as the configuration of the second factor must be carried out in cooperation with the user. For the highest possible level of security, the registration process appears to be suitable for incorporating Two-Factor Authentication (2FA). Each new user profile would therefore be secured by a second factor

before the first login. However, this design creates a trade-off between security and user-friendliness, whereby user-friendliness suffers considerably in favor of security. The resulting advantage is that a newly created user profile, which potentially does not yet contain any sensitive data, is secured with effect for the future. However, the disadvantage is that this measure makes the registration process considerably more extensive and requires additional dependencies, such as an authenticator app and a password manager. If registration takes place via a smartphone, the user would need another smartphone to scan the Quick Response (QR) code. However, as the profile does not contain any security-critical data immediately after registration, immediate configuration of the second factor is practically unnecessary.

The user's profile page functions as the configuration point for 2FA because of the stated reasons. At this location users can enable MFA to improve their profile security through a second-factor setup option. The system generates a Time-based One-Time Password (TOTP) secret for the user which appears both as a QR code and plain text. The QR code stores information in a format that authentication applications can automatically scan to reduce setup complexity. The application shows the secret code as plain text to enable users who cannot use QR code scanning to finish the configuration. Presenting the secret as plain text allows users to store it within their password manager applications.

This security measure makes it possible to protect against the loss of the mobile device, as the user is still in possession of the secret and thus the basis for generating the one-time passwords. However, the secure storage of the secret is the sole responsibility of the user. For technical and data protection reasons, it is not possible for the application to make this secret available again after it has been generated or to grant access in any other way.

The user enters the secret in the application of their choice and receives the six-digit single-entry password generated in a 30-second interval. They subsequently enter the single-entry password into the form for configuring the MFA, whereupon the server verifies its validity based on the stored secret and the current time. In the event of a match, the 2FA is already set up from this point in time and the further steps only serve to independently restore the user profile.

4.4.2 Recovery Codes

In scenarios where a user has lost both access to their authenticator device and any active, privileged session in the SCA tool, a conventional re-provisioning of the 2FA cannot be automated via the device. To cover this situation in accordance with the requirements of NIST SP 800-63B, a procedure for issuing and managing recovery codes was implemented, which enables the user to perform a one-time authentication without a valid session.

During the initial setup of the 2FA, the user is presented with twelve alphanumeric recovery codes, each of which is generated by a cryptographically strong random number generator. The codes consist of eight characters and are secured before being stored in the database at using a one-way hash method. A function for invalidating all previously issued codes and simultaneously generating a new code-set has been linked to the user interface in the self-service portal.

Each recovery code is designed for one-time use and is compared with the stored hash value when redeemed at. A successful use authenticates the user, but does not start a re-provisioning flow, in the course of which the user registers a new authenticator device. This is due to the fact that in Ory-Kratos the so-called `lookup secrets` serve exclusively as a fail-safe mechanism for the second authentication factor. The user is provided with a privileged session, which serves as the basis for manually re-activating the 2FA (Perl, 2025).

This method enforces the principle `separation of concerns` by handling identity verification in a dedicated authentication flow and enrolling new authenticator devices through a separate provisioning process. Due to the fact, that users must explicitly re-provision their 2FA, the attack surface is minimized and the 2FA setup integrity is protected from unintentional compromise during a one-time recovery procedure.

NIST SP 800-63B establishes minimum requirements for recovery codes within Section 3.1.2. Verifiers **SHALL** ensure that each look-up secret can only be used successfully once and becomes unusable after authentication. Secrets must be stored in such a way that they are protected against offline attacks. This **SHALL** be done using an entropy greater or equal to 112 bits and an approved hash function or alternatively with a salt and a suitable Key Derivation Function (KDF) in which the salt has at least 32 bits. For secrets with less than 64 bits of entropy, rate limiting **SHALL** also be implemented in accordance with rate limiting section 3.2.2. An approved encryption mechanism in an authenticated protected channel must always be used to transmit the look-up secrets in order to prevent eavesdropping and man-in-the-middle attacks (Temoshok et al., 2025).

To fully comply with the NIST-SP 800-63B requirements for Look-Up Secrets, the Ory Kratos configuration must be customized in accordance to listing 4.1.

```
hashers:
   algorithm: argon2
   argon2:
      parallelism: 1
      memory: 128MiB
      iterations: 3
      salt_length: 16
      key_length: 32
```

Listing 4.1: YAML configuration for the self-service settings flow

On the one hand, hashers argon2id is defined as the KDF under the top-level key `hashers`, as this memory-hardening method offers both low parallelizability and high resistance to trade-off attacks and thus covers the NIST requirements for a salt length greater or equal to 32 bits and the use of an approved key derivation method (National Institute of Standards and Technology, 2017). On the other hand, the parameters *parallelism*, *memory*, *iterations*, *salt_lengt* and *key_length* ensure that all look-up secrets are protected with at least 112 bits of effective entropy.

This implementation secures the user profile with a second factor and provides a robust recovery mechanism. In this layer, such attack vectors as brute-force attacks, rate limiting bypasses and replay attacks are properly mitigated. However, the configuration introduces two notable security issues, which are discussed in the following section to further enhance the security level.

4.4.3 Authenticator Assurance Level Elevation

The AAL is a classification level defined by NIST that describes the security strength of an authentication procedure and the authenticators used. The higher the level, the more robust the mechanism. In the current state of implementation, no separate login flow is triggered for the AAL elevation. Consequently, the session still remains at AAL1. Neither the session cookie nor the internal session object have an attribute that signals a successful second authentication level.

This creates an attack surface in which an attacker exfiltrates the valid session cookie through XSS, man-in-the-browser or physical access, and subsequently gains access to protected endpoints without knowing the password or the TOTP code. The attacker can retrieve the CSRF-Token embedded in the form by issuing a GET request to a valid flow. Using the session cookie, the obtained CSRF token and the parameter `totp_unlink=true`, a POST request can then be crafted to permanently disable 2FA and revert the account to single-factor authentication.

However, Ory Kratos employs the first successful TOTP code as both device

confirmation and authentication during the initial setup process. After successful verification, Ory Kratos raises the session to AAL2 on the server side. All security-critical operations require a valid TOTP code entry during this privileged session. This implementation effectively minimizes the open period during which attackers can perform session hijacking attacks.

4.4.4 Disabling two factor authentication

Once a user has set up their 2FA and persisted their recovery codes, they are given the option of deactivating them again at a time of their choosing. This is due to various security organizational reasons. In the event of a device change or loss, the user should be able to invalidate the previous configuration and carry out a new setup.

However, this measure can also result in a considerable security risk, in which an attacker takes over a privileged session (AAL2) by means of a session hijacking attack. This would enable them to remove the 2FA in the settings flow without having to re-enter the current one-time code.

This problem is discussed in Github issue #2450 of the Ory Kratos repository by KajsaEklof (2022) and represents a compromise between availability and security of the user profile. The Chief Technology Officer (CTO) of Ory comments that this omission of an additional code check was deliberately chosen to enable users to decouple lost or no longer available authenticator end devices. Otherwise, access to the user account would be impossible in the event of device loss.

In order to shift the trade-off more towards security, the configuration of a suitably short duration for privileged sessions within the settings area is suitable. The configuration in listing 4.2 was made in the Kratos settings flow.

```
selfservice:
   flows:
      settings:
         ui_url: http://localhost:5173/settings
         privileged_session_max_age: 15m
         required_aal: highest_available
         after:
            totp:
               hooks:
                  - hook: revoke_active_sessions
```

Listing 4.2: settings-flow: privileged session configuration

The configuration parameter `required_aal` set to `highest_available` enforces

that all functions of the settings flow can only be accessed within a privileged AAL2-session. The validity period of this session is limited to 15 minutes using `privileged_session_max_age`. After this period has expired, the second factor is queried again before sensitive actions can be carried out within the settings area. The `revoke_active_sessions` hook invalidates all previous sessions of the same user immediately after successful verification of the second factor. This ensures that potentially compromised sessions are invalidated at this time and only the last authorized session is valid.

This measure ensures that a stolen or otherwise compromised session cookie can only be used for a maximum of 15 minutes. In addition, any change to the configuration of 2FA is only possible within a privileged session and even if the second factor is successfully deactivated, all parallel sessions are invalidated. As a result, the attack window for session hijacking and replay attacks is significantly reduced and the overall resilience of the authentication system is increased.

4.5 Mandatory Re-authentication for Sensitive Operations

The confidentiality of user accounts is a key security aspect. Sensitive actions such as changing passwords, updating email addresses, or managing API tokens are particularly critical. When securing these sensitive actions, it is necessary to consider the underlying architecture of the SCA Tool. Ory Kratos is used for user management, and Monolith Representational State Transfer (REST) API endpoints are provided for business logic, among other things. This means that it is necessary to secure the Kratos endpoints appropriately and to provide robust protection for the endpoints of the Monolith API. Ory Kratos addresses this problem through the concept of privileged sessions.

4.5.1 Ory Kratos Endpoints

In the original draft (Rosenberger, 2025), it was recommended to set the parameter *privileged_session_max_age* to one second or ideally to 0. The aim is to force a new request for the password or the second factor almost immediately after a successful login and thus reduce the time window for session hijacking attacks to practically zero.

From a security perspective, Rosenberger's approach offers several decisive advantages. Firstly, automated attacks, such as replay attacks and session fixation, are effectively ruled out, as they cannot be carried out in less than a second. Secondly, a maximum assurance level is always required for all sensitive operations, which means that even users who have already been authenticated have

to go through a re-authentication process before every critically classified action. Thirdly, the effectiveness of this security measure can be clearly demonstrated. A test setup that forces mandatory re-authentication immediately after login is easy to reproduce and enables reliable validation.

Despite the strengths mentioned, however, there are various critical weaknesses. On the one hand, recurring re-authentications lead to considerable usability losses. Typical activities such as simple profile editing exceed the one-second threshold, which forces users to repeatedly log in again and in practice often leads to insecure workaround behaviors such as bypassing password managers. However, session timeout intervals, according to OWASP, should be chosen in such a way that they balance security and usability and avoid too frequent user logouts (OWASP Cheat Sheet Series, 2025). Fine-grained context differentiation does not exist, on the other hand. A generic Always-On re-authentication scheme treats all endpoints equally, so that even non-critical actions such as displaying profile information require a new login. In recovery flows, this can lead to users being unintentionally locked out when setting up or restoring their second factor.

In accordance with the security and usability requirements of the SCA tool, a compromise-oriented approach was chosen, resulting in the configuration presented in listing 4.3. This applies to the *Staging*, *Testing* and *Production* environments, while the time window in the *development* environments has been left at 15 minutes.

```
1 selfservice:
2         flows:
3                 ...
4                 settings:
5                         ui_url: https://app.scatool.com/
                             ↪ settings
6                         privileged_session_max_age: 5m
7                         required_aal: highest_available
```

Listing 4.3: *kratos.yaml*: Enforcing Re-Authentication for Profile Changes

A privileged session timeout of 5 minutes is combined with the consistent enforcement of a maximum assurance level for all security-critical endpoints. A 5-minute timeout limits the window for session hijacking to a practically negligible interval, but is well below the NIST SP 800-63B recommended upper limit of 30 minutes of inactivity to ensure smooth user interaction (National Institute of Standards and Technology, 2017). The commitment to AAL2 ensures that all privileged actions are protected with recently verified 2FA, further minimizing risk exposure. This approach follows the recommendations of the OWASP Session Management

Cheat Sheet, according to which timeouts must be set contextually in such a way that they balance security and usability (OWASP Cheat Sheet Series, 2025).

4.5.2 Monolith REST API Endpoints

Since User Interface (UI) routes and API actions in the context of business logic lie outside the self-service flows provided by Ory Kratos, a Sudo-mechanism has been implemented in the monolith that enforces re-authentication for security-related operations. Exposure in scenarios such as session theft or shoulder surfing should be avoided by ensuring that the authorization to make sensitive changes only remains valid within a verified time window.

An important use case where this validation is of great importance is the creation or removal of API tokens. These endpoints are outside the Ory Kratos self-service flow and should only be managed under secure conditions. In order to protect future sensitive actions from unauthorized access in the same way, it is necessary to decouple the primary routine logic from the security logic.

The technical implementation is based on a Spring Aspect-Oriented Programming (AOP) aspect, which intercepts all controller methods annotated with `@RequiresSudo` and enforces authentication confirmation before execution. For this purpose, the Ory Kratos `whoami`-endpoint is queried and the timestamp of the last successful authentication is validated. If this value is not available, the session issuance time is used as a fallback. If the calculated difference exceeds the configured upper limit `sudoMaxAgeMinutes`, an exception is triggered and the method call is aborted. The value for `sudoMaxAgeMinutes` is set to five minutes in line with the Kratos configuration in order to balance usability and security. The policy follows the OWASP ASVS requirement for defense against session abuse by enforcing full reauthentication before modifications can be made to the user profile that affect the authentication itself (OWASP Foundation, 2025b). Moreover the AOP approach ensures uniform enforcement across all controller methods and reduces deviations within the implementation of security mechanisms.

After the method call is canceled and an exception is triggered, an HTTP response with the status code `401 Unauthorized` is generated to signal to the client that authorization is missing. The status code `403 Forbidden` was deliberately not used here, as no negative authorization decision was made in this context, rather, an update of the user session is required. To ensure that the user returns to the original page after successfully logging in, the current Uniform Resource Identifier (URI) is transferred in the `return_to` parameter.

The annotation-based and centralized security logic offers various advantages in terms of maintainability, consistency, and quality assurance. On the one hand, the declarative pattern allows existing and future endpoints to be secured with

a `@RequiresSudo` annotation without additional boilerplate code. Second, the centrally configured `sudoMaxAgeMinutes` barrier configures a policy for all endpoints outside the self-service flow, whereby new critical actions are included in the same protection area by means of an annotation. It is important to note that this value must always be kept consistent with the Ory-Kratos configuration. In addition, the approach improves testability and monitoring by allowing unit and integration tests to check the aspect behavior for different error cases. Finally, metrics such as the number of expired sudo windows, the frequency of forced re-authentications, or the time to renewal can be used to quantify the observed risk situation and thus make data-supported parameter adjustments.

4.6 Ory Kratos Session Configuration

The investigation of session management in the SCA tool revealed various security-critical configurations and deficiencies in terms of security. The following sections examine session management with a focus on security-related default settings, explicit timeouts, and security-conscious configuration of Cross-Origin Resource Sharing (CORS) headers and sensitive cookies.

4.6.1 Absence of Global Session Timeouts

In the present configuration (see Appendix A), no higher-level *session*-block has been defined, which means that Kratos only controls the lifetime of individual self-service flows, but not the actual validity of user sessions. Kratos then uses a default session validity of 24 hours, which cannot be terminated by an idle timeout due to the lack of specification. Without an explicit idle timeout, sessions remain active until the full 24 hours have elapsed, which significantly extends the window for session hijacking. To mitigate this risk, a global *session*-block according to the Listing 4.4 was added.

```
1 session:
2         lifespan: 8h
3         ...
```

Listing 4.4: *kratos.yaml*: Session-Timeout

This defines an absolute maximum duration of eight hours. In this way, sessions are automatically terminated after a maximum total lifetime of eight hours, drastically reducing the window of opportunity for stolen cookies to be exploited.

4.6.2 Inactivity-Timeouts

In addition to the absolute lifetime of sessions, it is recommended to implement an inactivity timeout that prematurely terminates a session after a specified period without user activity. Without this inactivity timeout, the hijack window for a stolen session token remains open for the full remaining lifetime, allowing an attacker to execute any API calls on behalf of the user within this period. OWASP ASVS requires that the sessions time out after a defined period of inactivity and that periodic re-authentication is performed (OWASP Foundation, 2025b). For practical implementation, the OWASP Session Management Cheat Sheet specifies idle timeouts of between 15 and 30 minutes, taking into account the protection requirements of an application (OWASP Cheat Sheet Series, 2025).

Since Ory Kratos does not provide this functionality natively, a separate, server-side addition must be ensured. Since client-side idle timeouts can be easily circumvented and do not provide reliable verification at the server level, the last activity time must be stored persistently in the session database and checked against the current timestamp by a middleware policy for each incoming request. A periodically running background job then automatically deactivates all sessions whose idle time exceeds the defined threshold, while the frontend is exclusively responsible for user notifications. This approach complies with the requirements of NIST SP 800-63B and OWASP ASVS, thus ensuring a high level of security in line with industry standards.

4.6.3 CORS

As part of a defense-in-depth strategy, it is required to consistently restrict both CORS access and redirect URIs at every relevant security levels. Default or wildcard settings pose a significant security risk to the application. For example, in the current SCA Tool implementation, CORS support is enabled for Ory Kratos, but there are no explicit specifications for permitted origins, HTTP methods, headers, and credentials. Although the Ory Kratos documentation points out the importance of a concrete definition of the values, it leaves open which default concepts apply if no user-specific configurations are made.

To determine the default settings actually used, an HTTP request was sent to the Kratos server and the returned response headers were then analyzed. For this purpose, the `Change password` function within the settings page was used, which returned the response headers shown in listing 4.5.

```
HTTP/1.1 200 OK
access-control-allow-credentials: true
access-control-allow-origin: *
access-control-expose-headers: Content-Type
cache-control: private, no-cache, no-store, must-revalidate
content-type: application/json; charset=utf-8
set-cookie: ory_kratos_session=<ory_kratos_session>; Path
   ↪ =/; Expires=Tue, 22 Jul 2025 09:12:32 GMT; Max-Age
   ↪ =86361; HttpOnly; SameSite=Lax
vary: Origin, Cookie
date: Mon, 21 Jul 2025 09:13:11 GMT
connection: close
transfer-encoding: chunked
```

Listing 4.5: Response-Header of the Kratos API

An examination of the response headers reveals an incorrect CORS configuration, which leads to functional problems and potential security risks. Since there is no explicit definition of the configuration parameters for the allowed origins and the use of credentials, the server sends a wildcard for all origins in the response headers and allows the use of credentials. This configuration violates the Same Origin Policy (SOP) and allows any domain to initiate authenticated cross-origin requests to the Kratos API, sending session cookies and other credentials along with them. The associated security risk arises in particular from CSRF attacks, session hijacking, and data access, as attackers can issue requests from domains in their possession to access protected endpoints such as `whoami` and extract vulnerable user data.

It is important to note that modern browsers block all authenticated requests with this combination of CORS headers. This means that, in the context of modern browsers, legitimate frontend applications cannot perform authenticated requests due to the lack of configuration. Furthermore, the headers set are contradictory and ineffective, resulting in no clear benefit. In the context of legacy browsers, this combination also poses a security risk, allowing an attacker to exploit the above-mentioned security vulnerabilities.

To counteract this security risk for legacy browsers and ensure correct configuration, the parameters must be defined in a suitably restrictive manner. The parameters shown in table 4.5 are available within the Ory Kratos configuration.

Parameter	Description
`allowed_origins`	Exact origins (scheme, host, port) permitted to access the public API
`allowed_methods`	HTTP methods accepted for cross-origin requests (e.g., GET, POST, PUT, PATCH, DELETE)
`allow_credentials`	Whether credentialed requests are allowed (cookies, `Authorization` header, TLS client certificates)
`allowed_headers`	Non-simple request headers permitted in cross-origin requests (e.g., `Authorization`, `Content-Type`)
`exposed_headers`	Response headers readable by browser scripts (e.g., `Content-Type`)
`max_age`	Cache lifetime of preflight responses in seconds (`Access-Control-Max-Age`)

Table 4.5: Overview of CORS configuration parameters in Ory Kratos

In particular, the parameters require the following values for a secure configuration.

1. **Allowed origins and methods**

   The parameter `allowed_origins` allows the definition of a whitelist to enable the exchange of resources exclusively for specified origins. When deploying SCA Tool in various clusters, special attention must be given to specifying the correct frontend domain for each cluster. In addition the HTTP methods `GET`, `POST`, `PUT`, `PATCH`, `DELETE` are permitted. The cluster domains relevant for configuration are shown in figure 4.6.

Environment	Origin
Local development	http://localhost:5173
Dev-Cluster	http://app.scatool.localhost:8080
Staging-Cluster	https://staging.scatool.com
Testing-Cluster	http://10.131.64.77
Production-Cluster	https://app.scatool.com

Table 4.6: Approved Origins per deployment environment

2. **Credentials**
   Using the boolean flag `allow_credentials` signals to the browser whether authentication data may be transmitted in cross-origin requests. This is necessary for the functionality of Ory-Kratos and, in combination with trusted sources, does not pose a security risk.

3. **Header restriction**
   The restriction of headers is divided into allowed headers, which may be

sent by the client, and exposed headers, which are disclosed by the server to the client. The `Authorization` header is set as an allowed header to allow the client to access protected resources by passing an access token or API key using the *Basic* or *Bearer* authentication scheme. As part of session management, it is necessary to set the `Set-Cookie` (exposed header) and `Cookie` headers (allowed headers). The `Set-Cookie` header is transmitted from the server to the client to request that the client set the corresponding cookies, whereas the `Cookie` header is transmitted from the client to the server to prove ownership of the cookie. Specifically, Ory Kratos sets the session ID and the CSRF middleware token as cookies. The client stores these cookies in the browser and includes them with every request to the server, thereby validating its session and providing CSRF protection. Additionally, `Content-Type` is configured as both an allowed and an exposed header, enabling proper content negotiation between the client and server.

4. **Preflight Caching**
   Based on `max_age` set to *3600*ms, preflight requests are cached for one hour. Preflight requests are requests using the `OPTIONS` method, which are made before sending a complex cross-origin request. This checks whether the request origin, headers and methods are accepted by the server. The server responds with the permitted origins, methods, headers and whether the sending of credentials is permitted. Based on this response, the client's browser either executes the request or blocks it with a CORS error message.

It should be noted that preflight requests are only triggered for complex cross-origin requests that go beyond a simple HTTP method (GET, POST, HEAD) with standard headers. Simple requests that do not meet these criteria bypass the preflight mechanism and are therefore not affected by the cache duration.

4.6.4 Cookies

As part of the definition of CORS headers, the transmission of authentication data from authorized origins was approved. Cookies, which are responsible for session management and CSRF protection, among other things, are one form of authentication data.

With respect to cookie scoping and security, the following attributes are applied (MDN Web Docs, 2025b).

1. **Expires and Max-Age**
   The `Expires` attribute contains a date and specifies how long a cookie is valid. The `Max-Age` attribute works in the same way, but expects a maximum age in seconds. These attributes serve the purpose of signaling to the browser when a cookie has expired and should be deleted.

2. **Domain**
   The `Domain` attribute is responsible for enabling a specific cookie for a domain. Without defining the attribute, the cookie is a `Host-Only Cookie`, which means that the cookie is only set for the host and cannot be set in the subdomains. If, on the other hand, a cookie is enabled for a domain, the cookie is valid for the domain itself as well as for all subdomains. This means that the cookie can be sent to parent domains in the event of client requests. However, the server may only specify the host or parent domains in the `Set-Cookie` header to ensure protection against domain hijacking, in which an attacker can falsify session information by accessing a parent domain through the setting of cookies for a subdomain.

3. **Path**
   The `Path` allows to limit the validity of the cookie to specific Uniform Resource Locator (URL) paths. It is important to note that when specifying a path, all subpaths are included and the cookie is also valid for these.

4. **HttpOnly**
   `HttpOnly` is used to control access to a cookie via JavaScript. If the attribute is set, its value cannot be read using JavaScript. This provides an additional layer of security, whereby an attacker who has gained the ability to execute code via an XSS vulnerability cannot access the session cookie and thus the user's active session.

5. **Secure**
   A cookie with the `Secure` attribute is only sent to the server via encrypted Hypertext Transfer Protocol Secure (HTTPS) connections. This attribute primarily protects against eavesdropping and man-in-the-middle attacks and increases the integrity of the cookie and thus the session. Together with the `HttpOnly` attribute, both the confidentiality and integrity of sensitive cookies are ensured.

6. **SameSite**
   Finally, the `SameSite` attribute is of central importance. It enables the controlling whether and under what conditions cookies are included in cross-site requests. Cross-site requests are requests that occur between two pages with different schemes or domains. This behaviour is intended to create a security layer against CSRF attacks. The `SameSite` attribute can be defined in three modes.

   1. **Strict**
      This value offers the highest protection against CSRF attacks by not transmitting the cookie during cross-site requests or top-level navigation. Top-level navigation occurs when the URL in the browser's address bar changes and the user is taken to a new page or path.

With this configuration, the cookie would not be set by external links due to top-level navigation or cross-site requests and would only exist under the user's own domain.

2. **Lax**
   The value `Lax` blocks the transfer of cookies in cross-site requests similarly to `Strict`, but permits top-level navigation from external sites to the cookie's origin site under two conditions. First, the navigation must be executed using a safe HTTP method that does not change the server state and only performs a read operation. Safe methods include `GET`, `HEAD`, and `OPTIONS`. In addition, `Lax` takes the context of top-level navigation into account. Second, the navigation must be performed deliberately by the user in order for the cookie to be set. In the context of an embedded resource, such as an `iFrame`, an `image`, or the automated execution of a script, the cookie is not transferred.

3. **None**
   `None` disables the restrictions of the attribute and allows unrestricted transmission of cookies for all cross-site requests, regardless of the HTTP method or the context of the request. This configuration requires the use of the `Secure` attribute, which means that cookies can only be transmitted via HTTPS. Otherwise, the browser rejects the cookie and it is not set.

In the context of SCA Tool, it is necessary to secure the cookies presented in the table 4.7.

Name	Path	Expires	HttpOnly	Secure	SameSite
active_organization	/	Thu, 21 Aug 2025<T>	false	false	None
csrf_token_<random>	/	Wed, 22 Jul 2026<T>	true	false	Lax
ory_kratos_session	/	Wed, 23 Jul 2025<T>	true	false	Lax

Table 4.7: Overview of all SCA Tool Cookies

The cookie `active_organization` is set by the monolith and is used exclusively on the client side to assign the organization to a user. Due to the fact that the cookie contains non-sensitive information that requires client-side access, the HttpOnly attribute can be left set to `false`.

However, for the `staging` and `production` deployment environments, it is necessary to set the Secure attribute to `true` so that the cookie is transmitted exclusively in encrypted form. In addition, the cookie is only required for internal purposes, which means that the `SameSite` attribute should be set to at least `Lax`. This ensures that the cookie does not become a security vulnerability in future developments, even if it does not currently contain any sensitive data.

The configuration of Ory Kratos is responsible for setting the cookies `csrf_token_<random>` and `ory_kratos_session`. Ory Kratos automatically manages the attributes `Secure` and `HttpOnly`. While the `Secure` attribute is set based on the `dev` flag, which ensures that the cookie is also functional in the development environment, `HttpOnly` is always enabled for Kratos cookies (Ory Corp., 2025a).

This leaves the instance-specific configuration parameters `Domain`, `path` and `same_site`. In addition, Kratos allows configuring the global default values that apply to all cookies, as well as specific values for the session cookie. Based on this architecture, cookies issued by Kratos in the future will be secured by the default values.

The global default values are set according to the domain of the deployment environment (see Table 4.6). Since SCA Tool does not include segmentation of sub-applications and no differentiation between session and CSRF token cookies is required, the `path` attribute is set for the root and all subpaths.

To ensure global protection of Kratos cookies in the context of cross-site requests, the `same_site` attribute must be set to `Lax`. `Strict` would provide a stronger security layer against cross-origin attacks, but would have the disadvantage that the CSRF token would not be transferred in the context of top-level navigation. As a result, a user who accesses the SCA Tool login page via an external link would not be able to log into the application due to the missing token.

The configuration of the session cookie essentially corresponds to the global cookie default settings, but the parameter `same_site` set to `Lax` is selected for a different security consideration. The session cookie can be generated either by the native authentication process of the or alternatively via an external IdP, with IdPs GitHub and Gitlab being supported. During a simplified IdP login, the client initiates the process by redirecting to the IdP, where the user authenticates. Upon successful authentication, the IdP then redirects the client back to the SCA Tool with an authorization token, which is then validated to establishes a local session. This final redirect represents a cross-site request, which would be prevented in the case of a strictly configured cookie policy.

4.7 Multiple Active User Sessions

In the security audit, Rosenberger points out that the original implementation of the SCA Tool allows multiple sessions per user to be used simultaneously, which significantly increases the risk of session hijacking. An attacker with access to a valid session cookie retains full system privileges (Rosenberger, 2025).

To counteract this security risk, it is necessary to optimize the Ory-Kratos configuration with regard to session handling. For this purpose, Ory-Kratos provides

the *revoke_active_sessions* hook, which invalidates active sessions within the Ory database as part of a specific user action. Security-relevant user actions include password changes, password recovery, and login using a password or OIDC provider. In the context of a user logout, invalidation is not necessary, as multiple sessions are already excluded during a login process. Including the logout flow for invalidating sessions would only lead to redundant database accesses and would not provide any added security value.

The implementation of 2FA has added another security-critical user action that requires all previous sessions to be invalidated. Once a user has configured 2FA, their session is elevated to AAL2. This protects logins following the configuration, but not existing ones. An attacker who has access to a previous session is restricted by the AAL1 of their session from performing security-critical actions, but is still able to navigate within the application and perform non-security-critical actions. To prevent such a compromise of a user profile, all previous sessions are invalidated after successful setup of 2FA. The complete configuration for the invalidation of multiple sessions can be found in the listing 4.6.

```
flows:
   ...
   settings:
      ...
      after:
         password:
            hooks:
               - hook: revoke_active_sessions
         totp:
            hooks:
               - hook: revoke_active_sessions
   ...
   recovery:
      ...
      after:
         hooks:
            - hook: revoke_active_sessions
   ...
   login:
      after:
         password:
            hooks:
               - hook: revoke_active_sessions
         oidc:
            hooks:
               - hook: revoke_active_sessions
```

Listing 4.6: Ory-Kratos session revocation configuration

This setting is identical in every deployment environment and invalidates all

previous sessions for all security-critical user actions. To ensure that the security measure is also enforced at the application level, revoked Kratos sessions must be reliably detected and prevented each time the application is called.

For each incoming request to the Spring monolith, the `KratosFilter` servlet filter is called first. In it, the cookie sent by the browser is queried against Ory-Kratos. In cases where Ory-Kratos reports the session as inactive or absent, for instance following a `revoke_active_session` hook, the request is redirected to the login page.

At the frontend level, the *Ory-Middleware* ensures that the self-service flows are secured and controlled. Transferred redirection parameters are checked to avoid open redirect vulnerabilities and mapped to internal app routes. If an inactive or missing session occurs during a flow, the user is returned to the entry page and, after successful login, redirected to the original page.

By combining the `revoke_active_sessions` hooks activated in Listing 4.6 with the server-side *KratosFilter* and the web-based *Ory-Middleware*, the security measure for preventing compromised sessions is consistently enforced. Once a session has been invalidated, the status is detected with the next user interaction and, in this case, the application forces an immediate redirect to the respective entry page. Outdated session cookies thus lose their usability, as access to internal pages of the web application is denied. In addition, the controlled redirection increases the usability of the application by initiating a re-authentication flow instead of confronting the user with an error state.

4.8 Security Header

Web applications are exposed to a variety of client-side attack vectors, which are often due to insufficiently restricted content controls. These include, in particular, cross-site scripting, in which untrusted scripts are executed by manipulating the DOM or by injecting event handler attributes. Clickjacking is another typical attack scenario in the context of uncontrolled resource management, where iFrames can be embedded without restriction. In addition, the injection of malicious resources poses a major threat to web applications. The OWASP Top 10 lists, among other things, XSS and injection as particularly serious threats to the confidentiality, integrity, and availability of web applications (OWASP Foundation, 2021). In addition, to protect against XSS, clickjacking, and other attacks, the BSI recommends that the HTTP headers Content Security Policy (CSP), Strict-Transport-Security, Content-Type, X-Content-Type-Options, and Cache-Control **SHOULD** be used and that they **SHOULD** be tailored to the web application as restrictively as possible (Bundesamt für Sicherheit in der Informationstechnik, 2023b).

Central to resource control is the definition of a content security policy, which uses a declarative set of rules to specify the origin of scripts, styles, images, and other resources. Among other things, whitelists, Number used once (NONCE) , and hashes are specified, preventing unauthorized code from being executed or external content from being reloaded in an uncontrolled manner.

4.8.1 Theoretical foundations

HTTP security headers represent a first layer of client-side protection by communicating to the browser via a response header which behavior is permitted and which is not. This includes both the origin of reloadable resources and protocol or IFrame restrictions. In accordance with the defense-in-depth principle, reliance is not placed on a single protective measure, but rather a combination of complementary headers is used.

1. **Content Security Policy**
   Defines whitelists, NONCEs, and hashes for various forms of web resources for controlled management of permitted static and dynamic content.
2. **HTTP Strict Transport Security (HSTS)**
   The HSTS header instructs the browser to use HTTPS exclusively for the domain and its subdomains, prohibiting protocol downgrades for a specified period of time. This mitigates the risk of man-in-the-middle attacks through downgrades to HTTP.
3. **X-Frame-Options & Frame-Ancestors**
   The Security Headers prevent the embedding of specific subpages of a web application, thereby mitigating the risk of users being manipulated into performing unintended sensitive actions, such as profile modifications. In particular, this provides protection against clickjacking, which exploits iframe embedding to overlay malicious content.
4. **X-Content-Type-Options: nosniff**
   This header forces the browser to accept only the Multipurpose Internet Mail Extensions (MIME) type specified in the `Content-Type` response header and to refrain from any automatic sniffing. This reliably prevents both MIME confusion attacks and unintended drive-by downloads.
5. **Referrer-Policy**
   Completely prevents the referer header from being transmitted to external destinations in order to avoid potential leaks of internal path information.
6. **Permissions-Policy**
   Defines a per-feature allowlists (e.g., camera, microphone, geolocation, clipboard) to control access for embedded content. By default, these features

are denied to embeds unless explicitly enabled, which prevents abuse and circumvention of browser permission prompts.

7. **Cross-Origin-Opener-Policy: same-origin**
   Completely separates the window or tab context between external origins to prevent side-channel attacks such as cross-window leaks.

8. **Cross-Origin-Embedder-Policy: require-corp**
   Refuses to embed resources that are not explicitly allowed by the `Cross-Origin-Resource-Policy` (CORP). While the CSP validates the source of the resources, the Cross-Origin Embedded Policy (COEP) focuses on ensuring that the embedded resources from these origins are handled securely.

9. **Cross-Origin-Resource-Policy: same-origin**
   Blocks the loading of resources from external origins unless they are explicitly approved.

4.8.2 Definition of a Content Security Policy

Within Next.js, the middleware provides a suitable entry point for defining CSP headers before content is delivered. This is partly because the CSP configuration is transmitted consistently and uniformly across all routes. Above all, this ensures that existing and newly added routes automatically receive the complete CSP header, thus guaranteeing controlled resource management. In addition, this separates the business logic from the security configuration so that policy changes can be versioned, validated, and automatically tested and rolled out in Continuous Integration/Continuous Deployment (CD) pipelines. In addition, a NONCE can be defined for specific directives to control the execution of permitted scripts and the import of styles. For this concept to work securely, the generated NONCE must be different with each response from the server. It is therefore imperative to define the CSP within the middleware, which enables the generation and transmission of a cryptographically secure NONCE per request, so that a new and unpredictable value is assigned each time a page is reloaded and each time the user navigates (MDN Web Docs, 2025a). A suitably restrictive definition of a content security policy thus enables the zero trust strategy to be enforced by allowing resources to be integrated, executed, or communicated only under clearly defined conditions.

Figure 4.1 provides a schematic representation of the CSP directives and their fallback relationships. Each directive represents a separate resource category for which certain restrictions can be defined.

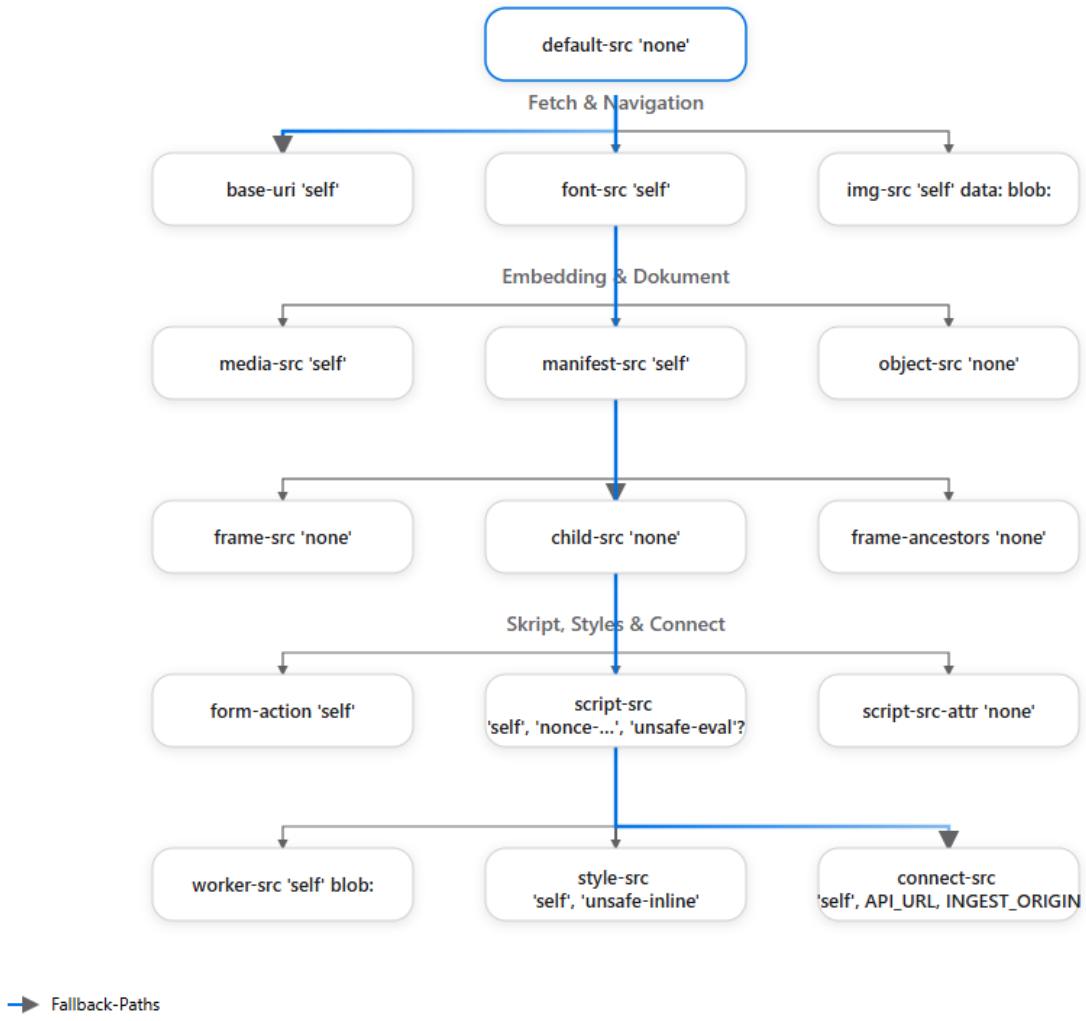

Figure 4.1: Hierarchical representation of CSP directives and their fallback paths

1. `default-src 'none'`

The `default-src` directive with the value `none` forms the foundation of the entire CSP configuration and enables the enforcement of the zero trust strategy. Specifically, the directive ensures the implementation of the following security controls.

Global Fallback Any request for resources such as scripts, stylesheets, images, and many others fall back to default-src, unless a specific directive is defined for that resource category. For example, if no `style-src` directive is specified, the browser refers to `default-src 'none'` and blocks any attempt to load Cascading Style Sheets (CSS). Similarly, the `connect-src` directive works in the same way, whereby any fetch, XMLHttpRequest (XHR), and WebSocket connections are prohibited in this context.

Reduction of the attack surface The restrictive default configuration automatically excludes potential gateways for XSS, data injection, mixed content, and clickjacking attacks, unless explicitly enabled by a subsequent directive.

Security monotony The Content Security Policy is a web standard that continues to evolve over time. At the time of writing this thesis, this standard is at Level 3 with the introduction of NONCE, the `strict-dynamic` directive, the `report-uri` directive, and many more. The fallback hierarchy ensures that even if new directives are added in future CSP levels, the policy remains consistent because unspecified directives do not change the restrictive baseline.

This means that `default-src ‘none’` already enforces a maximum default setting in the browser, which can only be relaxed by precise and isolated exception rules in subsequent directives.

2. Navigation and communication policies

base-uri The `base-uri` directive controls which origins in the `base` element are permitted as the basis for relative URLs on an Hypertext Markup Language (HTML) page. By default, it inherits the global fallback `default-src` analogous to all CSP directives without their own definition and thus blocks all `base` tags unless a whitelist is explicitly specified. The configuration `base-uri ‘self’` provides the following security guarantees in this context.

- **Relative URL integrity:** When resolving paths, the browser only uses its own origin. All relative references to scripts, styles, links, or form action URLs are processed consistently without referring to arbitrary or manipulated base paths.

- **Fallback behavior:** If `base-uri` is missing from the policy, the browser falls back to `default-src 'none'` and automatically blocks every `<base>` tag. However, explicitly allowing `'self'` maintains protection while allowing legitimate applications to continue using relative paths.

form-action The *form-action* directive limits the permitted target URIs for form submissions to an explicitly defined whitelist. In the SCA Tool, the form data is processed in two different ways. First, communication takes place with Ory-Kratos regarding profile changes and authentication actions. Ory-Kratos is located under the development environment and every deployment cluster under the frontend origin with the path */.ory*, which means that internal communication with Ory-Kratos is not restricted by `form-action 'self'`. Second, communication also takes place with the monolith, which, based on the deployment cluster, has a different origin than the frontend. This does not represent a restriction in the current implementation of the SCA Tool, as requests to the monolith are only made by `onSubmitHandlers`, which intercept the form submission and initiate an individual request to the monolith. However, if a form implementation is required in the future that communicates directly to an endpoint of the monolith, it is necessary to extend the form-action directive to include the origin of the monolith.

connect-src The `connect-src` directive defines the permitted destinations for all requests from the browser, including `fetch`, XHR, and WebSockets. In the current implementation, connections are only allowed to the SCA Tool backend API (Monolith) and the monitoring endpoint (SENTRY). Any attempts to send data to unauthorized hosts are blocked by the browser. This provides effective protection against silent data exfiltration.

3. Policy for resource access

In addition to controlling navigation and communication, it is necessary to restrict access to fonts, images, media, and manifest files. The directive `font-src 'self'` restricts the loading of web fonts to self-hosted files only. Similarly, the `img-src 'self' data: blob:` directive embeds image resources solely from the user's own origin and via data or blob URIs. The `media-src 'self'` directive implements a similar restriction with a focus on audio and video content, while `manifest-src 'self'` ensures that the web app manifest is protected against manipulative third-party hosting, thus ruling out inconsistent start URIs or icons. In addition, `object-src 'none'` completely prevents the loading of legacy plugins, which may contain potential security vulnerabilities. Overall, these guidelines prevent the uncontrolled integration of external resources, thereby eliminating key vectors for tracking and code injection.

It should be noted that the `img-src` directive is only responsible for loading images and has no effect on triggering events. This is significant because, in the context of XSS, the `img` tag in particular is often misused to trigger malicious event handlers. The attacker sets a deliberately incorrect `src` attribute and uses the `onError` attribute to execute malicious JavaScript code. Defining the `img-src` directive alone would not prevent this attack, and a corresponding definition of the `script-src` directive (see section 4.8.2, point 5) is necessary to prevent inline scripts and insecure event handlers, thus ensuring an appropriate balance between functionality and security.

4. Embedding and context protection

To ensure complete protection against clickjacking and the embedding of application components in external frames or subcontexts, the directives `frame-src 'none'` and `child-src 'none'` are set in the CSP. This prevents the creation of `iFrame` elements and the opening of a child context. In addition, `frame-ancestors 'none'` ensures that the application itself cannot be rendered as an `iFrame` on external domains by instructing the browser to reject all embedding attempts.

A typical clickjacking scenario, which is prevented by the defined directives, works as follows in simplified form. An attacker hosts a website under their own domain that embeds the SCA Tool profile page using a transparent, full-screen iFrame. They place visible controls over it that suggest harmless interaction. When the user clicks on a button, they interact with the invisible iFrame instead of the visible controls. A worst-case scenario here would be, for example, the deactivation of 2FA for the user's SCA Tool profile.

However, two essential conditions must be met for a real attack to occur. First, the user must be in a valid privileged session within SCA Tool at the time of the mouse click. Second, the form for deactivating 2FA must contain a valid CSRF token so that an error-free request can be executed. Without these conditions, the attack will fail. Under certain circumstances, however, these conditions may be met, opening up a potential attack surface. The CSP configuration for embedding protection prevents the SCA Tool frontend from being embedded into third-party iFrames, thus preventing clickjacking scenarios of this kind.

5. Script and work context policies

Since XSS attacks target the unauthorized execution of JavaScript code and DOM changes, a mechanism is required that differentiates between intentional and unintentional scripts and only executes the intended code. This is done using the CSP directive `script-src` with the origin restriction `'self'` and the attribute `'nonce-${nonce}'` . This first specifies that only locally hosted scripts

are allowed and that inline scripts are only executed if they contain the correct, cryptographically secure NONCE in the script attributes, which is updated with each page access.

This prevents classic XSS vectors, as foreign or reused inline code is rejected by the browser due to the lack of a valid NONCE. Since `next.js` requires the JavaScript function `eval()` for fast refresh and debugging in the development environment, a mechanism has been implemented that differentiates between the environments and sets the parameter `'unsafe-eval'` for the development environment. In the production environment, the parameter is strictly disabled to prevent attacks based on the `eval()` function.

In addition, `script-src-attr 'none'` is used to prevent inline event handlers such as `onclick`, `onload`, or `onError` (see section 4.8.2, point 3). Without this restriction, an attacker would be able to inject JavaScript code via manipulated HTML attributes and execute it, even if all external script sources are blocked.

Finally, the `worker-src 'self' blob:` directive specifies that worker scripts may only originate from their own origin or from blob URIs. Workers run in the background and have full access to JavaScript APIs such as `fetch` or `IndexedDB`. If an attacker gains access to reload external workers through potential additional security vulnerabilities, they would be able to exfiltrate data or implement persistent backdoors in the background without this being noticed in the main thread. However, combining this with `script-src` ensures that worker scripts also come from trusted sources.

4.8.3 Additional security headers

In addition to defining a CSP, other HTTP headers can contribute to increasing the overall security of the application. The effect of the following headers is explained in detail in the theoretical basics (see subsection 4.8.1, especially 4., 5., 6., 7., 8., 9.). Taking into account the missing security headers identified by Rosenberger (Rosenberger, 2025) and suitable additions, the definition was made within the Next.js middleware as shown in Listing 4.7.

```
X-Content-Type-Options: "nosniff",
Referrer-Policy: "strict-origin-when-cross-origin",
Permissions-Policy: [
        "camera=()",
        "microphone=()",
        "geolocation=()",
        ...
],

Cross-Origin-Opener-Policy: "same-origin",
Cross-Origin-Embedder-Policy: "require-corp",
Cross-Origin-Resource-Policy: "same-origin",

Server: "",
X-Powered-By: "",

Cache-Control: "no-cache, private",
Pragma: "no-cache",
Expires: "0",
X-Requested-With: "XMLHttpRequest",
X-DNS-Prefetch-Control: "off",
X-Download-Options: "noopen",
```

Listing 4.7: Security header configuration in Next.js middleware

The `Permissions Policy` has been shortened for clarity. The complete list of directives can be found in the appendix (see section B, Listing2).

5 Organizational Safeguards

Comprehensive security results from combining technical measures with organizational controls. The BSI standard 200-1 emphasizes that only a holistic approach that incorporates technical, infrastructural, organizational, and personnel aspects can sustainably increase the level of security (Bundesamt für Sicherheit in der Informationstechnik, 2017b). Building on the technical security measures described in the previous chapter, the following sections discusses organizational controls for monitoring changes to the MFA configuration and the systematic handling of CSP violations.

5.1 Email notifications for changes to the MFA

As part of the development of the MFA in section 4.4, a notification mechanism for security-related changes to the configuration was also integrated. The implementation of a multi-level notification system for changes to the 2FA is a central component of the reactive security architecture. In detail, the following events are recorded and communicated to users.

1. Activation of two-factor authentication
2. Deactivation of two-factor authentication
3. Initial generation and subsequent regeneration of recovery codes

Immediate notification of security-relevant changes enables users to identify unusual or unauthorized activities at an early stage and quickly initiate countermeasures. In particular, the BSI recommends always leaving 2FA enabled, as disabling it significantly reduces account security (Bundesamt für Sicherheit in der Informationstechnik, 2025).

The technical implementation of the notification function is an integral part of the Identity and Access Management (IAM). This not only raises the entry barrier for unauthorized access, but also systematically increases users' security awareness. In accordance with the process-oriented approach of the new IT baseline protection, technical measures must always be embedded in organizational processes in

order to ensure sustainable effectiveness.

The combination of 2FA and notifications links the technical detection of potential attack vectors with organizational response measures. This hybrid approach, which combines technical control based on the second factor and the organizational integration of users in the incident response process, ensures that security incidents are not registered in isolation, but can be tracked and actively addressed by users.

5.2 Reporting violations of the CSP

Building on the implementation of a restrictive CSP discussed in section 4.8.2, a reporting mechanism has been added. The technical foundation for this is the existing policy for controlling integrated resources and their communication, on the basis of which attack-induced and configuration-related CSP violations are to be centrally recorded, correlated, and flagged for remediation. For this purpose, each request to the server sets the `Report-To` header in the response, which communicates browser-side violations of the policy to Sentry's CSP monitoring endpoint. In addition, the deployment environment in which a policy violation occurred is transmitted.

The distinction between attack-induced and configuration-related CSP violations stems from the cause of the violation. If a web-resource-based attack is carried out on the SCA Tool application, such as XSS, this constitutes an attack-induced CSP violation. This is because an illegitimate resource was inserted due to a vulnerability in the application with the aim of executing malicious code. The violation itself does not necessarily pose a threat, but it does require an investigation into the source of the violation. In this case, there is a possibility that a non-validated resource was injected due to insufficient input validation and sanitization. In such a case, the investigation must result in securing the endpoint where the input is not correctly validated. Alternatively, a policy violation can already be triggered when a user or attacker inserts web resources such as scripts or HTML elements into the browser console. This does not indicate a vulnerability in the web application, but should be monitored continuously as it is a potential indicator of the `Reconnaissance` tactic for gathering target information from Adversarial Tactics, Techniques & Common Knowledge (ATT&CK)[1].

A configuration-related violation arises due to an overly restrictive policy and does not pose a security risk, but results in limited functionality. A realistic scenario for a configuration-related violation arises from the continuous development of the SCA tool. The CSP was defined for the current requirements of the application, and additional requirements arise as development progresses.

[1] https://attack.mitre.org/

These new requirements must be taken into account after the development of a new component, and the CSP must be relaxed accordingly. In this context, there is a risk of the policy being relaxed too much, which could lead to security vulnerabilities.

To counter both attack-induced and configuration-related CSP violations and their effects at the organizational level, Sentry offers a centralized, correlated, and detailed view of the violations. Figure 5.1 shows an excerpt from a Sentry report following a CSP violation.

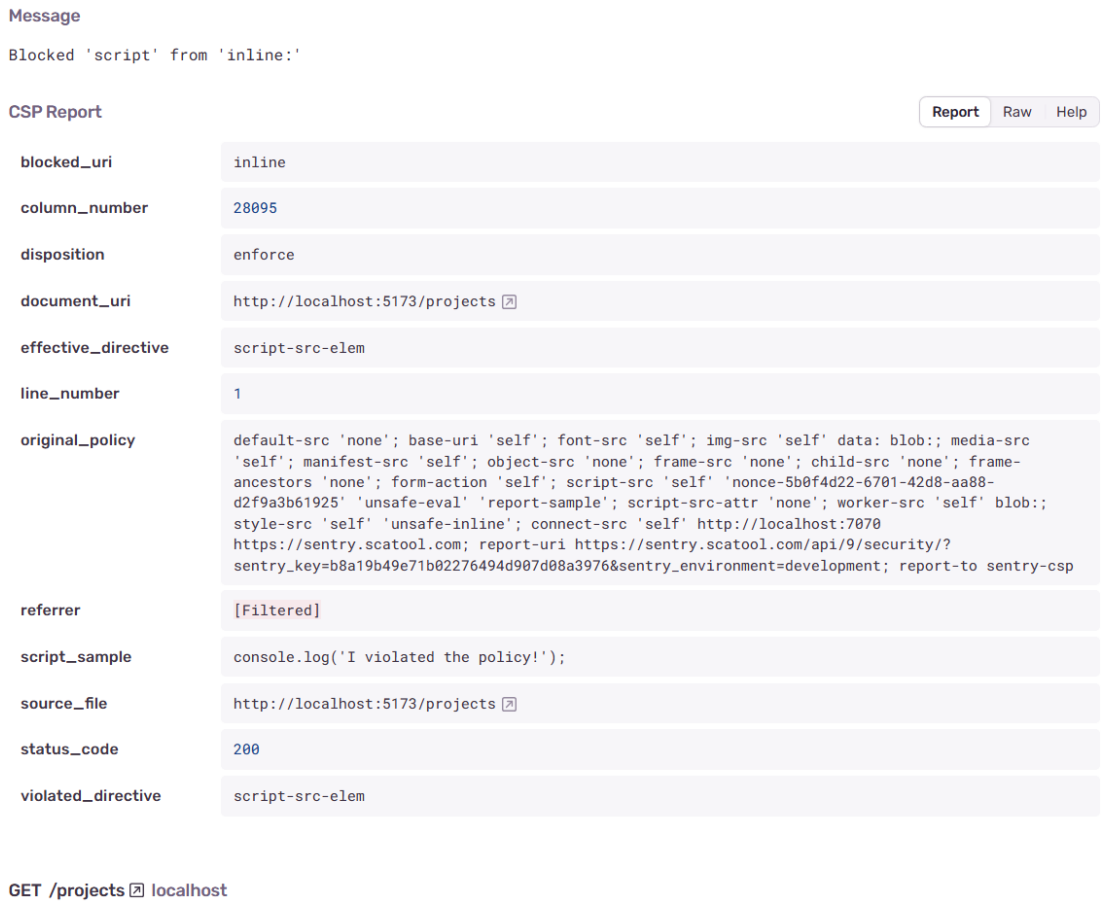

Figure 5.1: Excerpt from a CSP report in Sentry

At the beginning, a brief summary of the incident is described in the *Message* section. In Figure 5.1, a script element from the inline source was blocked. This means that an injected or static script exists in an HTML file that is not CSP-compliant.

This is followed by the *CSP Report*, which describes the policy violation in detail. It begins with the `blocked_uri` and identifies the blocked resource. Alternative values to inline are `eval`, `data`, `blob:`, or a complete URL. The value `eval`

refers to the function in JavaScript, where an unauthorized attempt was made to evaluate a script snippet. In the case of `data`, `blob:`, or a complete URL, a policy violation was detected due to the inclusion of non-validated resources with these origins.

The `blocked_uri` is followed by the `column_number`, which specifies the column position in the source code where the browser detected the violation. In connection with the column position, the line number in the source code is specified by `line_number` field to enable the error to be located precisely.

The third position describes the policy mode using the `disposition` field. The possible values are `enforce` and `report`, which are specified when defining the Content Security Policy. Enforce obliges the browser to block resources that violate the CSP. If the value is set to `report`, the violation is reported to Sentry and the resource is integrated. The exclusive reporting of violations is primarily used for testing before the introduction of a restrictive policy to ensure the existing functionality of the application.

The information in the `document_uri`, `effective_directive`, `original_policy`, and `script_sample` fields is of central importance for the analysis of the *CSP Reports*. The `document_uri` can be used to identify the URI of the document on which the violation occurred. By combining this with the `effective_directive`, which specifies the specific directive that was violated, with the `script_sample` and the `original_policy`, initial conclusions can be drawn as to whether the violation resulted from an attack or from an overly restrictive configuration of the policy itself.

Finally, the source file, the status code of the server response, and the violated directive are documented. While `effective_directive` specifies the normalized directive whose enforcement triggered the violation, `violated_directive` represents the specific policy rule that was violated. Depending on the browser, this may be the corresponding directive or a fallback directive (see Figure 4.1).

After evaluating a CSP violation, especially in the production environment and for *enforce* policies for unexpected or reproducible events, a GitHub issue should be created to maintain the security and integrity of the web application. Sentry allows the creation of a pre-filled GitHub issue directly from the corresponding event.

5.3 Rosenberger Policies

The policies developed by Rosenberger as part of the security audit form the organizational reference framework for the SCA tool and continue to be binding. These include the following policies in particular:

- **Information Security Management Policy**
- **Secure Software Development and Operation Policy**
- **Identity and Access Management Policy**
- **Password Policy**
- **Backup and Recovery Policy**
- **Unified Communications and Collaboration Policy**
- **Security and Compliance Audit Process**

These guidelines were developed in a context-sensitive manner based on the BSI IT baseline protection and address the confidentiality, integrity, and availability of the essential business processes of the SCA tool. Deviations from Rosenberger's specifications should only occur if technical, organizational, or economic reasons make this absolutely necessary and compensating controls are available.

6 Security Measures in Draft

This chapter conceptually elaborates and prototypically implements measures whose final integration depends on product, team, or environmental factors. The focus is on access protection for the GitHub organization and repository, control over access to the deployment environments, and secure password hashing. In addition, the integration of a Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) and multi-layer rate limiting are discussed.

6.1 GitHub Access Control

At the time of the SCA tool evaluation (March 17, 2025), Rosenberger identified various security vulnerabilities and potential solutions related to access control in connection with the GitHub repository. The identified deficiencies include the risk of Account Takeover (ATO) due to a lack of MFA, no allowlists for Internet Protocol (IP) addresses, Adversary-in-the-Middle (AiTM) attacks in insecure networks, and unbound mail domains (Rosenberger, 2025).

To counteract the risk of disclosure and unauthorized access to the repository by ATO, MFA has been enabled for all members of the SCA Tool GitHub organization. This significantly reduces the risk of account takeover through password compromise alone.

After careful consideration, the IP allow list recommended by Rosenberger was not implemented. On the one hand, the full functionality is only available in the *GitHub Enterprise Cloud* plan. Secondly, configuring this list is impractical for a dynamic, location-independent development team. Although the Friedrich Alexander University (FAU)-Virtual Private Network (VPN) makes it possible to access the SCA Tool Repository from the FAU network and thus independently of location, this configuration does not provide sufficient security benefits. GitHub is an external service, which means that only the public egress address of FAU is visible from its perspective. This means that all VPN users are granted access based on an allow list for FAU networks and no identity verification is set.

Personalized maintenance of IP addresses would also not be scalable and would conflict with legitimate GitHub services, such as GitHub Actions.

The reference to AiTM attacks due to working in insecure networks represents a security risk that must be taken into account. As the SCA Tool is being developed exclusively by members of FAU, the use of VPNs is suitable as a situational measure in insecure networks. This results in increased security due to the encrypted data traffic to the gateway. However, the core problem of an AiTM attack is still only addressed to a limited extent. In an AiTM scenario, an attacker positions themselves as an intermediary in a login process and intercepts access data and one-time passwords to create a valid session. In practice, this risk is mainly addressed by phishing-resistant MFA. This involves the use of WebAuthn and Passkeys, which are cryptographically bound to the target domain and thus prevent the passing on of login data. The Landesamt für Sicherheit in der Informationstechnik (LSI) explicitly recommends the use of real-time phishing-resistant MFA for protection against attack vectors such as AiTM, unless there are compelling organizational or technical reasons that speak against the use of the MFA solution (*Leitfaden: Phishing-resistente Multifaktor-Authentifizierung*, 2024).

Finally, the risk of organizational information being exfiltrated via email was also discussed. This risk can be addressed at the technical level with the *GitHub Organizations* plan by using Security Assertion Markup Language (SAML)-Single Sign-On (SSO) to link organizational membership to an identity. This ensures that only accounts authorized via the IdP have access to organizational resources and notifications. If this feature is not available, enforcement must be carried out at the organizational level. In this case, care must be taken to ensure that access is only granted to GitHub profiles with legitimate domains (`@fau.de`, `@scatool.com`, `@group.riehle.org`).

6.2 Deployment-Cluster Access Control

The SCA Tool covers the deployment environments of development, staging, testing, and production. While production is intended for the public, the remaining environments are exclusively for a specifically defined user group. Access to the non-production environments is limited to the FAU network, which still allows a large number of people to access them. In this context, Rosenberger highlights the risk that the environments intended for developers within the network could be attacked by actual attackers or students. The solution proposed is the introduction of an IP allow list within the *traefik* reverse proxy, with the aim of restricting access to the deployments to authorized persons only (Rosenberger, 2025).

While this approach achieves the desired effect, it is problematic in terms of the organizational process of developing the SCA tool. In addition to the static part of the SCA tool developers, it is also expanded by scientific work carried out by students. This means that this dynamic part varies over a semester or in shorter periods. Defining a static IP allow list in *traefik* results in a high, ongoing maintenance effort because the list would have to be adjusted and deployed at regular intervals as students change.

Furthermore, *traefik* is not a suitable entry point for filtering unauthorized users. Before *traefik* receives network traffic, a request is first sent to the Cloudflare edge server, which in turn forwards this request to the origin. User-based filtering is not possible at the *traefik* level, and the block only takes effect after passing through the edge server. In addition, the evaluation of the client IP depends on trusted proxies and the `X-Forwarded-For` header, whereby misconfigurations can lead to circumvention possibilities.

A target-oriented alternative is to define a policy on the Cloudflare Edge server. This can be implemented, for example, via Cloudflare Access and Web Application Firewall (WAF) rules to enforce identity- or network-based filtering. Due to a lack of administrative rights at Cloudflare, this work is limited to conceptual specifications. It is recommended to implement this using Cloudflare Access through IdP group control in conjunction with WAF rules.

6.3 Secure Password Hashing

As part of the previous security audit, it was noted that bcrypt with cost=8 is used in the testing deployment, which creates an unrealistic load profile in development and promotes configuration drifts toward staging or production (Rosenberger, 2025). In addition to Rosenberger's finding, the use of lookup secrets in 2FA necessitates the provision of a suitable hash function (see section 4.4.2).

To ensure adequate security, the cost factor for the testing environment was raised to *12*. In the medium to long term, switching to `Argon2id` is advisable, as the hasher offers strong protection against offline password cracking attacks and is more scalable (Rosenberger, 2025). In terms of user experience targets, the project team decided to carry out the switch in the following security thesis under controlled conditions. This is necessary because the use of the `Argon2id` hashing algorithm requires appropriate calibration depending on the corresponding deployment environment and the underlying hardware.

To determine the appropriate `Argon2id` parameters, calibration should be performed in the target environment using the Kratos Command Line Interface (CLI). Calibration is performed locally to the pod using the following command:

```
kubectl -n <ns> exec -ti <kratos-pod> -- kratos hashers
   ↪ argon2 calibrate 120 --min-duration 600ms --format
   ↪ yaml
```

Listing 6.1: Argon2id Calibration

Kratos automatically combines various `Argon2id` parameters, including memory, iterations, and parallelism, to achieve the defined target values. The target values include the expected login load in the form of the number of requests per minute for the Kratos pod. It should be noted that this load refers exclusively to requests that cause server-side password hashing. This therefore only affects flows that contain a password, such as registration, login, or password changes. Secondly, the target hash duration per password is defined, which is crucial for achieving a suitable balance between security and UX. Ory-Kratos recommends a duration between 500ms and one second to ensure this balance (Ory Corp., 2025b). In the calibration command, the example values of 120 requests per minute in combination with a hashing duration of 600ms were used. These values may vary based on the target environment.

As part of the migration to `Argon2id`, existing `bcrypt` hashes remain valid and will be replaced by an `Argon2id` hash the next time a user successfully logs in.

6.4 Turnstile-Verification via Ory-Kratos-After-Hooks

The current state of the SCA Tool implementation provides login, registration, and recovery pages as pre-authentication interfaces. This means that any user on the Internet can access these interfaces without further restrictions and initiate an unlimited number of requests. In terms of system availability and database authenticity, this situation poses an enormous potential risk.

The availability of the system can be compromised by attackers who automatically send a large number of requests to these endpoints, potentially from distributed devices. This attack vector is known as a Distributed Denial of Service (DDoS) attack, which aims to increase the resource consumption of a system to such an extent that parts of the system or the entire system fails.

In this context, various factors can compromise the authenticity of the system's database. On the one hand, the lack of further validation of requesting users means that an unlimited number of user profiles can be registered. If this is done using an automated script, the database storage grows very rapidly and resource consumption is increased by background jobs, such as sending an email

verification code for registration. In addition, sending verification emails reduces the reputation of the sending email address and worsens the spam score. On the other hand, the created accounts can be used to carry out further attacks by exploiting them for following attack scenarios, such as spam or phishing.

To counter this security risk, a combination of security measures must be implemented. To prevent mass registrations, fake accounts, and misuse of the login process through attack vectors such as credential stuffing, these endpoints must be protected by a CAPTCHA. Since the application is operated via the Cloudflare infrastructure, the integration of the Turnstile CAPTCHA provided by Cloudflare is suitable.

The basic principle of CAPTCHA is based on collecting signals via the browser and the user's environment. Among other things, browser fingerprinting, mouse movements, timing patterns, and network characteristics are taken into account to determine whether the user is a real person or a bot. In addition, invisible challenges with adaptive difficulty are presented to the client at various levels. Turnstile challenges include mathematical calculations, cryptographic tasks, and timing-based tests. The complexity of the challenges depends on factors such as the trustworthiness of the IP address or the number of request attempts. At the first level, signals are passively collected when the page is loaded. On the second level, the client is presented with JavaScript-based challenges that they must pass. If the first two levels do not allow for a concrete conclusion, the third level requests visible interaction from the user as a fallback. After completing the challenges, Cloudflare analyzes the results and, if successful, issues a digitally signed, time-limited, single-use token to the client, which must be transmitted when submitting a form to the SCA Tool server. Finally, an internal implementation must ensure that this token is validated by a Cloudflare endpoint before executing a specific request. A positive result from the Cloudflare endpoint must lead to the continuation of the requested action, while a failure must lead to its termination.

It is already clear that the main objective of CAPTCHA is to ensure that the protected actions are performed manually by a real person. By rejecting invalid or abusive requests at an early stage, resource consumption is reduced and an implicit contribution is made to protecting the availability of the system. However, CAPTCHA still requires server validation of the token and thus server capacity, which means that the availability of the system at unauthenticated endpoints remains vulnerable to attack. For this reason, rate limiting is a suitable addition to the integration of Turnstile CAPTCHA to ensure comprehensive protection against denial-of-service attacks. The design of rate limiting in the context of SCA Tool is discussed in more detail in the following section.

To integrate Turnstile CAPTCHA into the existing self-service flows for login, registration, and verification, it is first necessary to implement a reusable UI

component. This will also protect future endpoints from automated bot activity. In addition, this approach enables the generic integration of CAPTCHA and thus the triggering of the first validation instance using the Cloudflare Challenge platform. The widget must then be integrated into the UI components to be protected.

In the second step, validation is enforced on the server side within the authentication and registration flows. When the form is submitted, the turnstile token is passed to Ory-Kratos as transient additional information (transient_payload). Kratos then calls a downstream webhook, which is executed before the session is issued and the identity is persisted. The webhook performs a server-side verification process in which the transmitted token is validated via the Turnstile Verify endpoint. If validation is successful, the Kratos flow continues and the action intended by the user is executed. If the validation fails, the webhook responds with an HTTP error status and the process is terminated. This ensures that the security mechanism is enforced, as direct requests to the corresponding API endpoint cannot bypass token validation.

To strengthen the robustness of the validation process, not only is a successful validation of the Turnstile Verify endpoint checked, but a context-bound check is also performed. Here, the *action* attribute, which describes the context of the respective form submission, is compared on the server side with a predefined *expectedAction* value. This prevents a token issued for a specific flow from being reused in a different context. This reduces the risk of an attacker using a valid login token to bypass potential protection mechanisms in the registration process.

In summary, there is a clear separation of responsibilities, with the frontend initiating the turnstile challenge and transporting a short-lived token, while the final decision is made on the server side within the Kratos flow. Secrets, such as the turnstile secret, must be stored as sealed secrets in the same way as the previous secrets of all SCA Tool deployment environments. If an illegitimate error occurs, such as a genuine user's token expiring, the system behaves in a *fail-secure* manner. This means that although validation fails and the security of the system is guaranteed, the user is notified of the problem and asked to initiate the process again. Due to the prioritization of other security measures, a complete implementation was not provided within the scope of this thesis, but a detailed conceptual implementation was provided with draft pull request *#1537*.

Finally, it should be emphasized that although CAPTCHA enforcement prevents abusive behavior at the application level, it does not offer protection against DoS attacks. For comprehensive protection, rate limiting should be implemented as a supplementary security measure. This combination results in a protection concept that both prevents unwanted bot actions and improves the availability of the system.

6.5 Rate-Limiting

While turnstile CAPTCHA already provides a security measure for identifying bot activity, combining it with multi-layered rate limiting is a good idea as part of a defense-in-depth strategy. Rate limiting is a mechanism for controlling incoming and outgoing network traffic. Among other things, it is designed to minimize the likelihood of a successful brute force attack and prevent attacks such as DoS, which target excessive resource consumption and thus application failure.

The goal of this measure is to protect specific endpoints of the application from excessive access by both bots and legitimate users. Taking into account the SCA Tool architecture, this goal is achieved by implementing rate limiting on multiple layers.

Firstly, the use of Ory-Kratos for user management means that requests to the authentication, registration, and verification endpoints at Cloudflare's edge must be limited. Limiting to the application layer is insufficient for two reasons. First, limiting at the application layer is too late because TLS, the reverse proxy, WAF, and Kratos itself have already been loaded, resulting in high rejection costs. Second, this creates the risk that the endpoints to be protected can be contacted independently of the web application, thereby circumventing the limiting. Consequently, rate limiting must be implemented at the Cloudflare's edge. To this end, Cloudflare provides the option of creating specific rules to control rate limiting. The rules include the parameters of table 6.1, which can be set either via the Cloudflare dashboard or the API.

`Parameter`	**Purpose**
`Matching Expression`	Condition for counting/mitigation (e.g. path match for `/self-service/*`).
`Characteristics`	Aggregation key of the counter (e.g. IP, Header value, Cookie, Query-Value, Host, Path, Country, ...)
`Threshold`	Maximum number of requests allowed per counting window
`Period`	Length of the counting window
`Mitigation Action`	Reaction when exceeded, e.g. *Managed Challenge*, *Block* (HTTP 429) or *Log only*.
`Mitigation Timeout`	Duration for which mitigation remains active (e.g. 15–30 min Lock/Increased friction).

Table 6.1: Parameter overview for Cloudflare WAF rate-limiting rules

The definition of a rule begins with the specification of a matching expression, which determines the conditions under which the rule should apply. In the context of Ory-Kratos authentication endpoints, it is appropriate to set the match-

ing expression to three specific paths. These include */self-service/login**, */self-service/registration** and */self-service/recovery**.

The IP address is considered a characteristic by default, which can be supplemented by optional parameters such as specific headers or the geolocation of the IP address. Based on these characteristics, the frequency of requests is tracked in order to generate a mitigation to them.

The threshold parameter defines the circumstances under which Cloudflare applies a mitigation to an exceedance. There are two options that can be selected depending on the subscription plan. The basic threshold counts the requests within a time window and reacts accordingly. Alternatively, there is the option of defining a complexity-based threshold, where exceeding the threshold is measured based on the complexity or cost of processing the request. The period parameter only exists in combination with the request-based threshold and specifies the time frame in which the threshold must be exceeded for a mitigation to be applied. If the violation condition is met, the mitigation action specifies the type of countermeasure to apply.

One possible mitigation is to block subsequent requests with the same characteristics. Cloudflare also offers three variations of challenges. With the Managed Challenge, Cloudflare automatically selects a challenge type based on signals, ensuring minimal disruption to the process. In an Interactive Challenge, the user is asked to solve a task. This offers the most security within the framework of rate limiting, but also results in the highest friction. The third variation is the JavaScript-based challenge without user interaction. When a user visits a page, a JavaScript code is injected and executed to solve a task. A less invasive response to a threshold being exceeded is to log these violations. This provides an overview of access violations and additional analysis, but does not provide security and should only be used as a supplementary measure.

Finally, the mitigation timeout for a rule is defined, specifying how long the selected mitigation action should remain in effect after an exceedance. There are two behavior modes. The first mode is “Perform action during the selected duration” and triggers the action for all matching requests during the mitigation timeout window, which must be greater than zero. In the second mode, “Throttle requests over the maximum configured rate”, the mitigation timeout is set to zero and the action is applied only to requests above the limit (Cloudflare, Inc., 2025).

The conceptual definition of rate limiting rules is summarized in Table 6.2 and serves as an initial safeguard for Ory-Kratos authentication endpoints at Cloudflare’s edge via WAF rate-limiting rules.

Matching Expression	Char.	Thre.	Peri.	Mitigation Action	Mitigation Timeout
`/self-service/login`	IP	5	60 s	Managed Challenge	0 (Throttle)
`/self-service/registration`	IP	3	60 s	Managed Challenge	0 (Throttle)
`/self-service/recovery`	IP	3	120 s	Managed Challenge	0 (Throttle)
`/self-service/verification`	IP	3	120 s	Managed Challenge	0 (Throttle)
`/self-service/`	IP	200	1 s	Block (HTTP 429)	60 s

Table 6.2: Initial Cloudflare rate limiting rules

Building on the rate-limiting rules, an implementation of a rate limiter at the application layer is also suitable as a secondary control. Unlike edge limiting, an in-app limiter can access authenticated identities and action contexts, allowing it to enforce detailed restrictions on costly actions. When implementing an in-app limiter, it is important to ensure that HTTP code 429 with a Retry-After value is passed when a threshold is exceeded, and additional attention is paid to neutral error texts to avoid disclosing sensitive information.

7 Evaluation

This chapter evaluates the effectiveness of the technical and organizational security measures implemented for the SCA tool as part of this work. With reference to the requirements defined in Chapter 3, any gains or potential losses in security are examined. In addition, both the software level and organizational aspects are considered.

For the evaluation, each measure is assigned to the requirements and reviewed based on measurable criteria.

7.1 Dependencies

In line with the reevaluation from requirement **R-1**, the first step in section 4.1 was to check whether the vulnerabilities identified by Rosenberger are still current and pose a risk to the SCA tool. In this context, the criterion for measuring the security gain is based on the fact that the vulnerable dependencies have all been upgraded to a stable version and no longer pose a threat to the SCA tool. To this end, the dependencies were scanned again to compare the current status with the identified CVEs. This revealed that the libraries discovered by Rosenberger are already available in stable versions and that there are no associated CVEs.

The rescan revealed additional packages with CVE findings, some of which are classified as *HIGH* or *CRITICAL*. This finding was documented as a GitHub issue to fulfill requirement **R-5** and released for discussion with the development team. This discussion revealed that GitHub Dependabot[1] directly addresses the existing vulnerabilities with regard to requirement **R-7**. With one exception, all existing CVEs were thus dealt with and a significant security gain was achieved.

GitHub Dependabot handles transitive risks via top-level upgrades, which means that deeply nested dependencies can remain if the version limits of the chain prevent them from being removed. In this regard, the discussion concluded that the SCA tool should be responsible for the structural monitoring of CVEs in the

[1] https://github.com/dependabot

future. Until then, transitive dependencies will be handled manually.

In summary, it can be concluded that all vulnerabilities found have been closed and the application therefore has no open CVEs according to the SBOM scan dated August 12, 2025.

7.2 Measures focusing on user flows

This section discusses measures for increasing the overall security of user-specific flows. This includes secure configuration for user passwords, implementation of MFA and all related functionalities, as well as measures for securing sensitive user actions in the SCA tool architecture.

7.2.1 Deployment-specific Password Policy

Section 4.2 discusses the strict enforcement of a restrictive password policy, taking into account the recommendations of `NIST SP 800-63B` and OWASP as well as requirement **R-3**.

In accordance with **R-2** (*Practical evaluation of the approaches*), the security benefits and UX impact have been weighed up. This assessment shows that while a mandatory minimum length of 15 characters and an HIBP check demonstrably reduce the risk of weak or compromised passwords, hard rejections in the event of network errors and strict composition rules create measurable friction according to studies. For this reason, a temporary risk acceptance was chosen with regard to requirement **R-6** until the UX concept and integration variant were finalized. The decision is documented in issue #1453 in accordance with **R-5** (*Traceability and transparency*) and, with draft pull request #1454, provides the appropriate password configuration taking into account the deployment environment.

The remaining residual risk due to the current configuration is mitigated by compensatory measures. On the one hand, the mandatory AAL2 elevation makes it more difficult to perform security-critical actions. On the other hand, the effective attack window is shortened due to the enforcement of re-authentication, making it more difficult to misuse the user profile even with suboptimal passwords.

In conclusion, it can be stated that the measure and the associated requirements have been partially fulfilled. The policy has been configured and tested in terms of content with draft pull request #1454. Enforcement has been deliberately postponed in line with the concept, and supplementary controls reduce the impact of an attack in the event of an ATO due to insufficiently complex passwords. In order to comply with the recommendations of NIST and OWASP and to meet the requirement **R-9** for effectiveness monitoring, the enforcement variant in combination with monitoring of violations is suitable. The resulting findings can

be used to optimize the process flow with regard to user guidance and to avoid disproportionately impairing the UX (**R-8**).

As the implementation of the policy was postponed, it was not possible to validate any criteria for its security effectiveness. During implementation and monitoring, the criteria in Table 7.1 must therefore be taken into account, depending on the integration variant.

Metric	Description
Password quota	Percentage of passwords set during registration or password change that meet the minimum length requirement ≥ 15 (**R-3**, **R-9**)
HIBP Hit-Rate	Percentage of entered or suggested passwords that have at least one HIBP match (**R-3**, **R-9**)
HIBP Error-Rate	Percentage of registration or login attempts affected by a network error related to HIBP (**R-8**, **R-9**)
User Friction	The rate of interruptions in dialogues concerning the password (**R-8**, **R-9**)
Effect of sudo reauthentication	Average time between the last (re-)authentication and the execution of a security-critical action (**R-3**, **R-9**)

Table 7.1: Criteria for evaluating the password policy

7.2.2 Multi-Factor Authentication

For protection against password-based attacks and session hijacking, Section 4.4 dealt with the introduction of MFA and all associated sub-processes. This addresses key risks in the context of the authentication path and increases the resilience of these endpoints.

In accordance with the requirement to evaluate the identified vulnerabilities (**R-1**) in terms of their relevance and exploitability, it was determined that all of the attack vectors mentioned are still possible due to the lack of MFA. Based on this, the solution proposed by Rosenberger was reviewed for its practicability and potential limitations in accordance with **R-2**. The solution involves activating TOTP-based MFA within Ory-Kratos and is correct except for a few syntactic deviations. MFA does not impose any limitations on the performance of the application or on users, as it is a setting that users can manage themselves.

The implementation of MFA based on TOTPs with recovery codes for restoring

the account and enforced AAL2 elevation in security-critical actions closes the security gap identified in the pre-audit. The setup of an additional factor forces sensitive operations to only take place in a privileged, time-limited context. This reduces the effective attack window in the event of a session compromise and prevents the decoupling of MFA from the user profile without recently verified authentication. The protection extends to the login flow, through the entry of the TOTP, and to critical endpoints within the settings area, thus protecting the *Authentication and account lifecycle flows* according to **R-3**. Furthermore, this implementation follows the defense-in-depth approach and thus fulfills the requirement for the implementation of prioritized findings (**R-7**).

In view of requirement **R-4** for the protection of critical public endpoints, access to the user profile is restored using recovery codes. In addition, all parallel sessions are invalidated after confirmation of the second factor, minimizing the impact of session tokens that have already been compromised.

To ensure traceability and transparency (**R-5**), architecture and configuration decisions were documented as part of a pull request. Due to the lack of MFA and rate limiting, and the fact that access to a user account also grants access to vulnerabilities in the user's repositories, the risk was classified as very high and risk mitigation measures were initiated. The risk is mitigated by implementing MFA and designing rate limiting, thus fulfilling the requirement *Risk assessment and prioritization* (**R-6**).

Maintainablity, configurability, and efficiency (**R-8**) are achieved by implementing a modular structure, embedding it in existing user flows to reduce risk, and making it configurable via the Kratos configuration.

7.2.3 Sensitive Operations

The configuration of Ory-Kratos' self-service flows for mandatory enforcement of AAL2 and a privileged session window of five minutes addresses security-critical actions. This enforces recently verified authentication for sensitive operations and reduces the exploitation window for session hijacking, replay, and CSRF-based attacks on the privileged session duration (**R-3**, **R-4**, **R-8**). The deviation from the original 1-second window is not implemented due to the impairment of application functionality and the restriction of user flows (**R-2**). Security restrictions cannot be determined and the recommendations of NIST are followed with this approach.

For monolith endpoints outside the Kratos flow, the annotation-based Sudo mechanism provides a semantically equivalent protection mechanism. This ensures that security-relevant actions, such as the management of API keys, are canceled outside the privileged session and reauthentication is requested. This consistently hardens API-based operations and future critical routes against session abuse and

shoulder surfing without disproportionately restricting usability (**R-3**, **R-4**, **R-8**).

7.3 Session and access control

As part of session management in Ory Kratos, an absolute session lifetime was introduced in section 4.6. This limits the previous default window of 24 hours to a defined, shorter period of eight hours and reduces the maximum usable duration of stolen tokens accordingly (**R-3**, **R-7**).

In addition to the general session, the NIST and OWASP recommend integrating an idle timeout of between 15 and 30 minutes. Without invalidation due to a longer period of inactivity, there remains a risk that a stolen session could be misused until the end of its absolute lifetime. However, this risk is mitigated by the fact that security-critical events can only be performed on the basis of reauthentication. In addition, authentication by a user causes all previously active sessions to be invalidated, which means that the stolen session may be deactivated before the absolute lifetime expires under certain circumstances. The deviation is documented in a GitHub issue and this thesis and is therefore acceptable until implementation.

In addition, a missing configuration for the CORS headers sent by the server was identified. The default values constitute a violation of SOP and lead to exploitable vulnerabilities in legacy browsers in the form of CSRF attacks, session hijacking, and many others. In modern browsers, this results in a contradictory misconfiguration that can lead to inconsistencies and errors.

To counteract these shortcomings, an environment-specific whitelist for origins, methods, and headers is defined within the CORS header. This blocks authenticated cross-site requests from unauthorized domains and only allows authorized methods and headers within the own domain. In accordance with requirement **R-4**, attacks in the form of SOP bypasses on sensitive endpoints are prevented. By explicitly defining the values for each deployment environment, awareness of the configuration is created in accordance with **R-10**. The procedure was discussed and documented in pull request #1248 (**R-5**).

To protect browser-side authentication information, the definition of attributes for security-critical cookies was discussed in section 4.6.4. For the `staging` and `production` deployment environments, the attributes *HttpOnly* and *Secure* ensure the confidentiality and integrity of cookies. `SameSite` set to `Lax` ensures a balance between CSRF protection and the requirements of top-level navigation and redirection after authentication with the IdP.

By securing critical endpoints in the form of authenticated cross-origin requests,

requirements **R-3** and **R-4** are fulfilled. Cookie attributes are documented in a context-specific manner, and their implementation in accordance with the requirement for traceability and transparency (**R-5**) can be traced in pull request #1248.

Risk assessment and prioritization (**R-6**) is only partially fulfilled. Although CSRF-based attacks are a serious attack vector, this attack is primarily based on the HTTP method *POST*. With the value `Lax`, the risk is thus reduced to *GET*-based CSRF attacks resulting from modified links in the context of phishing. All *GET* methods must be `safe` and only perform read operations on the server. This further limits the risk to the incorrect implementation of *GET* methods that can be exploited by an attacker. Such methods are not present in the SCA tool at the time of evaluation. However, due to the continuous development of the SCA tool, there remains a small residual risk that the exploitation of this attack vector will be enabled due to the disregard of the provision of safe *GET* methods.

7.4 Evaluation of Multiple Active Sessions

Section 4.7 addresses the risk of compromised cookies due to parallel valid sessions identified in the pre-audit.

In combination with AAL2 elevation after 2FA setup, the configuration prevents outdated AAL1 sessions from performing privileged operations. The requirement to secure *account lifecycle flows* (**R-3**) and investigate critical endpoints (**R-4**) is met by invalidating previous sessions in the login, registration, identity verification, password reset/recovery, and session management flows. This measure addresses a prioritized vulnerability identified in the pre-audit in accordance with **R-7**. The implementation and alternative approaches have been fully documented in pull request #1262 (**R-5**).

Furthermore, maintainability and UX were taken into account during the implementation of the measure (**R-8**). From the user's perspective, revoked sessions do not result in error states, but lead to a reauthentication request and redirection to the original destination.

A short latency until the user's next interaction, which causes a redirection to the authentication endpoint, remains as a residual risk. This discrepancy occurs exclusively at the frontend level and does not allow an attacker to gain additional information after their session has been invalidated on the server side. From the moment the session is invalidated on the server side, all subsequent requests are answered with `Unauthorized`. The residual risk only includes all information obtained up to this point and a usability restriction that requires a subsequent interaction for redirection.

7.5 Security Header Evaluation

With regard to the definition of security headers in section 4.8, this provides extensive protection against web-based attacks. The strictly formulated content security policy ensures that only explicitly defined resources are used. Any deviating resources are blocked by the policy and reported to the reporting endpoint. This strengthens the client-side attack surface against XSS, clickjacking, MIME sniffing, and cross-origin data leaks. In addition, communication between resources is restricted, the embedding of external frames is prevented, and only permitted connection targets are loaded.

The measures thus remedy the weaknesses identified in the preliminary findings (**R-1**) and, in line with the practical evaluation of Rosenberger's solution (**R-2**), move the provision of headers to the Next.js middleware to avoid any loss of functionality. To ensure the traceability of the chosen approach and transparency regarding the extended components, the implementation was documented in accordance with **R-5** in GitHub Issue #1410 and Pull Request #1444. Taking usability into account, an individual policy was formulated for the different deployment environments to avoid restrictions in the development process. Building on this, the central provision of the CSP ensures consistent transmission of the headers across the entire web application (**R-8**). The implementation follows the defense-in-depth approach by providing a comprehensive layer of protection for the client side in addition to the server-based protection mechanisms. This means that the failure of one layer does not immediately lead to a security incident (**R-7**).

A connection to Sentry's reporting endpoint was established to measure the effectiveness of the CSP and monitor violations against it. This enables the aggregated evaluation of policy violations depending on the deployment environment. Misconfigurations and regressions can thus be detected at an early stage, and any restriction of functionality or security incident can be identified in good time and prevented by taking appropriate measures (**R-9**).

In addition to CSP, a number of security-related headers (see section 4.8.3) have been introduced. Among other things, HSTS ensures that the production environment is automatically accessed via HTTPS. This mechanism protects users from man-in-the-middle attacks by ensuring that communication with the production environment of the SCA tool is exclusively encrypted. The restrictive permissions policy prevents the misuse of embedded components such as the camera, microphone, or clipboard. Consequently, the additional security headers provide a complementary layer of protection that makes it significantly more difficult to violate the confidentiality of the SCA tool and misuse scripts (**R-7**).

7.6 Requirements Coverage Summary

The table 7.2 assigns the requirements to the measures implemented in this thesis. The status *Fulfilled* describes the complete implementation of the measure, taking into account the associated requirement. *Partially* indicates a conscious compromise or the need for additional work to fully implement the measure. The status *Not applicable (N/A)* means that the requirement is not meaningful or possible in the context under consideration. *Deferred* describes one or more measures whose realization has been postponed and whose productive implementation is therefore outside the scope of this thesis.

Req.	Measure(s)	Status
R-1	Libraries re-scanned and upgraded; security headers reviewed; session configuration reworked; MFA gap closed.	**Fulfilled**
R-2	Password policy drafted (min length/HIBP); privileged session window set to 5 min (1 s rejected on UX grounds).	**Fulfilled**
R-3	TOTP and recovery codes; AAL2 elevation on sensitive flows; sudo re-auth for monolith; 8 h absolute session.	**Fulfilled**
R-4	CORS allowlist; cookie hardening (HttpOnly/Secure/SameSite=Lax); re-auth on critical settings; session revocation.	**Fulfilled**
R-5	Issues/Pull-Requests for every change; decisions and risks documented.	**Fulfilled**
R-6	High risks mitigated (MFA, sessions, headers); temporary acceptance for Password enforcement and transitive chains.	**Partially**
R-7	Pre-audit priorities implemented; Dependabot active; manual handling for deep transitive dependencies.	**Fulfilled**
R-8	Environment-specific configuration; central middleware; modular Ory Kratos configuration; reusable @RequiresSudo aspect.	**Fulfilled**
R-9	CSP report collection active; MFA change alerts in place; Password policy metrics defined for future enforcement.	**Partially**
R-10	Explicit defaults for CORS/methods/headers and cookies; CI guard for logging configuration.	**Fulfilled**

Table 7.2: Requirements Coverage by Implemented Measures

Beyond covering the requirements, Table 7.3 provides an overview of targeted measures that have been deliberately deferred for reasons of prioritization or underlying context.

Topic	Status	Rationale
Server-side idle timeout	**Deferred**	No native idle invalidation in Ory Kratos; concept provided; to be implemented as server middleware and a periodically running background job (Section 4.6.2).
Password policy enforcement	**Deferred**	Draft Pull-Request #1454; pending UX integration & monitoring to avoid undue friction (Section 4.2, **R-9**).
Structural monitoring of transitive CVEs	**Deferred**	Manual handling until structural monitoring for the SCA Tool backed is integrated (Section 4.1).
Edge rate limiting and secondary app-level limiting	**Deferred**	Edge rules concepted and in-app limiter planned for identity-aware throttling (Section 6.5).
Turnstile CAPTCHA	**Deferred**	Concept and draft Pull-Request #1537; to be finalized post-CSP tuning (Section 6.4).
Password hashing migration to Argon2id	**Deferred**	Calibration procedure defined and staged migration planned (Section 6.3).
Organizational and infrastructure controls (Cloudflare Access/IP allow-lists)	**N/A**	Constraints related to feasibility, licensing, or operation; alternatives suggested (Sections 6.1, 6.2).

Table 7.3: Measures deferred or not applicable to the context

8 Conclusion

The aim of this thesis was to increase the overall security of the SCA tool by implementing technical and organizational security mechanisms.

To this end, the vulnerabilities identified by Rosenberger and the proposed countermeasures were reviewed for their relevance and feasibility. Based on these findings, security mechanisms consistent with a defense-in-depth strategy were integrated. Furthermore, additional vulnerabilities were identified and solutions were designed to address them. Based on the requirements formulated in chapter 3, a defense model was implemented in which the interaction of the system components was taken into account for each measure. This primarily ensured that components which are fundamentally independent of each other, such as Ory-Kratos and the monolith, provide consistent protection.

The evaluation in Chapter 7 shows that the combination of measures achieves a noticeable gain in security. All vulnerable and outdated libraries were upgraded to a stable version, the security of user profiles was increased by introducing MFA based on TOTP codes, and the logging of sensitive information by Ory-Kratos was prevented at the technical and organizational level. In addition, the execution of security-critical actions was limited to recently verified sessions, both in the context of identity management and within the scope of the SCA tool's functionality. To ensure the confidentiality and integrity of sessions throughout their lifetime, browser-side security attributes were defined for the cookies to be protected. In connection with cookie protection, an explicit CORS definition was used to provide technical enforcement of specific origins, methods, and headers on the client side. To further strengthen the client side, security headers in the form of a restrictive CSP specification and supplementary headers for encrypted communication , JavaScript-API restrictions, and the prevention of caching of security-relevant server responses were added.

With a focus on ensuring ongoing security measures, monitoring measures have been established to detect security-related events at an early stage and initiate countermeasures. This actively involves the user in the security process by informing them of any changes to their MFA configuration. CSP reports relieve

the burden on SCA tool administrators and developers by reporting violations at an early stage and enabling the identification of potential attack patterns.

Despite the proven security effect, there are still noticeable limitations to the measures implemented due to conflicting objectives. In the context of MFA, `recovery codes` are used to restore a user profile, but are vulnerable because they are stored by the user. Unauthorized access to these codes can lead to the takeover of the user profile and thus to the viewing of organizational information. The strict implementation of a security-conscious password policy can lead to potential usability losses, which in turn can lead to circumvention measures on the part of the user. These circumvention measures can lead to a disproportionate reduction in the security of a user profile, which means that a suitable design must be created in accordance with the factors to be considered in Section 4.2.

In summary, it can be concluded that IT, information, and data security is a continuous process. The security measures introduced provide a significant increase in security, but also require additional measures to avoid negative side effects. In addition, many measures have a complementary effect and only lead to a robust security architecture when combined. This thesis provides a starting point and a clear framework for further safeguards.

9 Future Work

This chapter highlights starting points for further work that have emerged from the results or limitations of this thesis. The aim is to address potential improvements to the SCA tool and increase overall security. The chapter 6 on security measures in draft provides additional detailed information on the implementation of the necessary security measures and, where possible, presents conceptual implementations in the form of pull requests.

9.1 Idle-Timeout for the Session-Management

Ory Kratos does not natively provide a central configuration option to enforce server-side invalidation of inactive sessions. According to the recommendations of OWASP and NIST, sessions should be invalidated after 15 to 30 minutes of inactivity (see section 4.6.2). Therefore, a timeout must be implemented to protect against the misuse of stolen session cookies. Care must be taken to determine a suitable benchmark for user activity or inactivity as to prevent the erroneous invalidation of active sessions.

For a customized implementation concept, the endpoints `deleteIdentitySessions`[1] and `disableSession`[2] endpoints of the Ory Kratos Identity API can be used to delete or invalidate an inactive session. The information required to evaluate user activity can be obtained from specific interactions in the frontend in combination with the return data of the `whoami` endpoint. Based on this information, it is then necessary to invalidate the session in the backend and communicate to the frontend that the user should be redirected to the authentication endpoint. This concept secures sessions while taking usability into account, as the invalidation does not lead to an error state due to missing permissions.

[1] https://www.ory.sh/docs/reference/api#tag/identity/operation/deleteIdentitySessions
[2] https://www.ory.sh/docs/reference/api#tag/identity/operation/disableSession

9.2 Turnstile-CAPTCHA

In section 6.4, the integration of a Turnstile-CAPTCHA was conceptually prepared and a detailed implementation approach was provided as part of pull request #1537. The next step is to implement the concept in a productive manner.

When implementing the challenge, it is particularly important to ensure that the defined CSP (see section 4.8.2) does not restrict any necessary functions of the CAPTCHA. Specifically, it is necessary to enable Cloudflare's challenge endpoint for both the origin of scripts and frames. The Cloudflare origin for frames is necessary as the user requires direct or indirect interaction with the CAPTCHA, which takes place within a frame. The script source ensures that the Cloudflare code for the challenge can be executed.

In addition, it is necessary to consider maintaining the protected endpoints in case of failures in the context of CAPTCHA. Legitimate users must not be locked out of their profiles in the event of an error. This means that a mechanism must be implemented that allows interaction with the protected endpoint independently of CAPTCHA. In order to secure the authentication endpoint, one option is to develop a risk-based approval process. In the event of a CAPTCHA failure, additional information such as device or IP address matching or recent successful sessions can be used to determine whether the user is legitimate.

9.3 Multi-layered rate limiting

Section 6.5 presented the necessity and a detailed concept for the first layer of rate limiting. Specifically, initial edge rules were defined for Cloudflare to ensure the protection of publicly accessible endpoints. Building on this, the implementation of a rate limiter at the application level is suitable as a second layer. This layer ensures the limiting of requests, taking identities into account, and provides additional protection for particularly computationally intensive endpoints. When implementing rate limiting in different layers, care must be taken to avoid both redundancies and conflicting configurations between the layers.

Overall, the rate limiter at the edge level is intended to protect the infrastructure, while the limitation at the app level is intended to protect against misuse by legitimate users.

9.4 Phishing resistant MFA

In section 6.1, the risk of an AiTM attack was discussed alongside other security risks for the protection of the SCA Tool GitHub repository. In this attack,

attackers redirect users to invalid login pages without their knowledge and intercept credentials and time-sensitive one-time passwords in real time. This class of attacks shows that knowledge- or code-based factors can be passed on and reused under certain circumstances. The risk also exists for authentication in the SCA tool.

Phishing-resistant MFA based on passwordless authentication addresses this security risk by cryptographically binding the authentication factor to the domain of the web application.

The implementation of phishing-resistant MFA requires consideration of the User Experience. Mandating the use of this authentication mechanism takes away users' freedom to decide on their user profile and potentially leads to rejection. Instead, it is more appropriate to offer this option and point out that it makes the authentication process simpler and faster. For technical implementation, Ory Kratos supports phishing-resistant MFA in the form of the `WebAuthn`[3] (Web Authentication) standard, in which cryptographic key pairs are stored on the user's device and serve as proof of identity. `WebAuthn` enables users to authenticate themselves using biometric factors, hardware security keys, or platform-based authenticators.

However, when prioritizing this security measure, requirement 6.3.3 of the OWASP ASVS must be taken into account, which emphasizes that the implementation of phishing-resistant MFA is intended for the third level (OWASP Foundation, 2025b). This level describes the highest security level in terms of the ASVS (see section 2.3). For this reason, it is recommended to prioritize ensuring that basic security requirements of the lower levels are met before implementing advanced protection mechanisms.

9.5 Abuse detection for the MFA

As an organizational measure, a notification mechanism was implemented in section 5.1 that communicates security-related changes to the MFA to the user. In addition to enabling immediate response to incidents, this is intended to integrate the user into the security process.

The context of this measure offers the possibility of supplementing the mechanism with additional anomaly signals. The access pattern can be checked for deviations from the norm by reporting unusual user agents, unknown devices, or velocity checks, in which the usual location is compared with the current access location, taking into account the time elapsed since the last authentication. In addition, the analysis of behavior-based signals, such as frequent failed attempts to enter

[3] https://www.w3.org/TR/webauthn/

the second factor, is suitable as an indicator of user profile theft.

To address the anomalies, pure reporting should first be integrated so that valid users are not restricted. Once reliable detection has been established, automated mechanisms can be integrated to perform additional verification in cases of high risk probability or to automatically invalidate a user session in the event of critical actions.

From a long-term perspective, the use of a Security Information and Event Management (SIEM) system should be considered, as it enables the central aggregation, correlation, and analysis of security-related events from different sources. By using a SIEM system, the detection of anomalies within the framework of MFA can be contextualized with additional security-critical events to ensure comprehensive threat detection.

9.6 Verifiable ASVS mapping

In order to ensure verifiable traceability of the achieved security level and targeted further development, the security measures of the SCA tool should be systematically mapped to the OWASP ASVS controls (see section 2.3). This is to ensure that the security measures are measurable and verifiable. In addition, this approach enables the identification of potential vulnerabilities in existing configurations that require additional security measures.

In detail, the first step is to define a scope and the desired level of security. Next, the inventory must be collected in the form of existing artifacts. This means that relevant code snippets, configurations, policies, test cases, and pipeline reports must be recorded. The third step is to establish a verification strategy. Depending on the artifacts recorded, code review checklists, unit and integration tests, or negative tests are possible. The measures identified as open or partially fulfilled must then be formulated as GitHub issues and prioritized based on risk. In addition to this, it is advisable to extend the existing security policy in the CI/CD pipeline to automatically validate security-conscious configurations and report critical misconfigurations.

Appendices

A Ory Kratos Configuration

```
version: v1.3.0

dsn: "postgres://ory:ory@postgres:5432/ory?sslmode=disable&
   ↪ max_conns=20&max_idle_conns=4"

serve:
public:
base_url: http://localhost:4433/
cors:
enabled: true
admin:
base_url: http://kratos:4434/

selfservice:
default_browser_return_url: http://localhost:5173/
allowed_return_urls:
- http://localhost:5173

methods:
password:
enabled: true
config:
haveibeenpwned_enabled: false
oidc:
enabled: true
config:
providers:
- id: github
provider: github
client_id: xxx
client_secret: xxx
mapper_url: "file:///etc/config/kratos/oidc.github.jsonnet"
scope:
- read:user
- user:email
- id: gitlab
provider: gitlab
issuer_url: https://gitlab.com
client_id: xxx
client_secret: xxx
```

```
40 mapper_url: "file:///etc/config/kratos/oidc.gitlab.jsonnet"
41 scope:
42 - read_user
43 - openid
44 - profile
45 - email
46
47 flows:
48 error:
49 ui_url: http://localhost:5173/auth/error
50 settings:
51 ui_url: http://localhost:5173/settings
52 privileged_session_max_age: 15m
53 required_aal: highest_available
54 recovery:
55 enabled: true
56 ui_url: http://localhost:5173/auth/recovery
57 use: code
58 verification:
59 enabled: true
60 ui_url: http://localhost:5173/auth/verify
61 use: code
62 after:
63 default_browser_return_url: http://localhost:5173/
64 logout:
65 after:
66 default_browser_return_url: http://localhost:5173/
67 login:
68 ui_url: http://localhost:5173/auth/login
69 lifespan: 10m
70 registration:
71 lifespan: 10m
72 ui_url: http://localhost:5173/auth/register
73 after:
74 password:
75 hooks:
76 - hook: session
77 - hook: show_verification_ui
78 oidc:
79 hooks:
80 - hook: session
81
82 log:
83 level: debug
84 format: text
```

```
leak_sensitive_values: true

secrets:
cookie:
- PLEASE-CHANGE-ME-I-AM-VERY-INSECURE
cipher:
- 32-LONG-SECRET-NOT-SECURE-AT-ALL

ciphers:
algorithm: xchacha20-poly1305

hashers:
algorithm: bcrypt
bcrypt:
cost: 8

identity:
default_schema_id: default
schemas:
- id: default
url: file:///etc/config/kratos/default.schema.json

courier:
smtp:
connection_uri: smtps://test:test@mailslurper:1025/?
   ↪ skip_ssl_verify=true
```

Listing 1: *kratos.yaml*

B Complete security header configuration

```
"Permissions-Policy": [
        "camera=()",
        "microphone=()",
        "geolocation=()",
        "payment=()",
        "usb=()",
        "bluetooth=()",
        "accelerometer=()",
        "gyroscope=()",
        "magnetometer=()",
        "autoplay=()",
        "encrypted-media=()",
        "fullscreen=()",
        "picture-in-picture=()",
        "screen-wake-lock=()",
        "web-share=()",
        "xr-spatial-tracking=()",
        "clipboard-read=()",
        "clipboard-write=()",
        "gamepad=()",
        "hid=()",
        "idle-detection=()",
        "midi=()",
        "otp-credentials=()",
        "publickey-credentials-get=()",
        "serial=()",
        "storage-access=()",
]
```

Listing 2: Complete Permissions-Policy directive

References

Bundesamt für Sicherheit in der Informationstechnik. (2017a, October). *BSI Standard 200-2: IT-Grundschutz-Methodik* (tech. rep.). BSI. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/BSI_Standards/standard_200_2.pdf?__blob=publicationFile&v=2

Bundesamt für Sicherheit in der Informationstechnik. (2017b, October). *BSI-Standard 200-1: Managementsysteme für Informationssicherheit (ISMS)* (tech. rep. No. 200-1). BSI. Bonn. Retrieved October 10, 2025, from https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/BSI_Standards/standard_200_1.pdf?__blob=publicationFile&v=2

Bundesamt für Sicherheit in der Informationstechnik. (2023a). *IT-Grundschutz-Kompendium. Edition 2023*. Retrieved October 6, 2025, from https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/IT-GS-Kompendium/IT_Grundschutz_Kompendium_Edition2023.pdf

Bundesamt für Sicherheit in der Informationstechnik. (2023b, February). *Con.10 – entwicklung von webanwendungen* (Technischer Bericht). IT-Grundschutz-Kompendium, Edition 2023. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/IT-GS-Kompendium_Einzel_PDFs_2023/03_CON_Konzepte_und_Vorgehensweisen/CON_10_Entwicklung_von_Webanwendungen_Edition_2023.pdf?__blob=publicationFile&v=4

Bundesamt für Sicherheit in der Informationstechnik. (2023c, July). *Zero Trust* (Technische Leitlinie). BSI. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeLeitlinien/Zero-Trust/Zero-Trust_04072023.pdf

Bundesamt für Sicherheit in der Informationstechnik. (2025, May). *Zwei-Faktor-Authentisierung*. Retrieved May 28, 2025, from https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Informationen-und-Empfehlungen/Cyber-Sicherheitsempfehlungen/Accountschutz/Zwei-Faktor-Authentisierung/zwei-faktor-authentisierung_node.html

Cloudflare, Inc. (2025, July 29). *Rate limiting parameters*. Retrieved September 10, 2025, from https://developers.cloudflare.com/waf/rate-limiting-rules/parameters/ Last updated: July 29, 2025.

Grassi, P. A., Fenton, J. L., Lefkovitz, N. B., Danker, J. M., Choong, Y.-Y., Greene, K. K., & Theofanos, M. F. (2017, June). *Digital identity guidelines – enrollment and identity proofing requirements* (NIST Special Publication No. 800-63A). National Institute of Standards and Technology (NIST). https://doi.org/10.6028/NIST.SP.800-63a Includes updates as of March 2, 2020.

KajsaEklof. (2022, May 6). *No validation for `totp_code` when unlinking totp 2fa method*. GitHub. Retrieved May 4, 2025, from https://github.com/ory/kratos/issues/2450

Komanduri, S., Shay, R., Kelley, P. G., Mazurek, M. L., Bauer, L., Christin, N., Cranor, L. F., & Egelman, S. (2011). Of passwords and people: Measuring the effect of password-composition policies. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*, 2595–2604. https://doi.org/10.1145/1978942.1979321 CHI 2011 Honorable Mention.

Leitfaden: Phishing-resistente Multifaktor-Authentifizierung (tech. rep.). (2024, June). Landesamt für Sicherheit in der Informationstechnik (LSI) Bayern. Nürnberg. https://www.lsi.bayern.de/mam/aktuelles/leitfaden_phishing-resistente-mfa_v1-1.pdf Management Summary: Empfehlung FIDO2/WebAuthn; Abschnitt 3.2: Echtzeit-Phishing und robuste MFA.

MDN Web Docs. (2025a). *HTTP Content Security Policy (CSP) Guide*. Retrieved July 11, 2025, from https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP%5C#nonces

MDN Web Docs. (2025b). *Verwendung von HTTP-Cookies*. Retrieved July 21, 2025, from https://developer.mozilla.org/de/docs/Web/HTTP/Guides/Cookies

National Institute of Standards and Technology. (2017, June). *Digital Identity Guidelines: Authentication and Lifecycle Management (nist sp 800-63b)* (Special Publication No. 800-63B). National Institute of Standards and Technology. Gaithersburg, MD. https://doi.org/10.6028/NIST.SP.800-63b Includes updates as of March 2, 2020.

Ory Corp. (2025a, July 22). *Configuring cookies*. Retrieved July 22, 2025, from https://www.ory.sh/docs/kratos/guides/configuring-cookies

Ory Corp. (2025b, September 2). *Kratos hashers argon2 calibrate*. Retrieved September 2, 2025, from https://www.ory.sh/docs/kratos/cli/kratos-hashers-argon2-calibrate

Ory Corp. (2025c, September 30). *Logs and audit trails*. Retrieved October 10, 2025, from https://www.ory.sh/docs/self-hosted/operations/logging

OWASP Cheat Sheet Series. (2025, May 7). *Session management cheat sheet*. Retrieved May 7, 2025, from https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

OWASP Foundation. (2021). OWASP Top 10. Retrieved July 10, 2025, from https://owasp.org/Top10/

OWASP Foundation. (2025a). *Authentication cheat sheet* [Owasp cheat sheet series]. Retrieved August 13, 2025, from https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

OWASP Foundation. (2025b, May 30). *OWASP Application Security Verification Standard (ASVS)* (Standard). OWASP Foundation. Retrieved October 8, 2025, from https://github.com/OWASP/ASVS/tree/v5.0.0/5.0

Perl, H. (2025, April 29). *Lookup secrets - a MFA fail-safe*. Ory Corp. Retrieved October 10, 2025, from https://www.ory.sh/docs/kratos/mfa/lookup-secrets

Rosenberger, A. (2025, March 17). *Sca tool security audit* [Master's thesis]. Friedrich-Alexander-Universität Erlangen-Nürnberg, Faculty of Engineering, Department Computer Science, Professorship for Open-Source Software.

Temoshok, D., Fenton, J., Choong, Y.-Y., Lefkovitz, N., Regenscheid, A., Galluzzo, R., & Richer, J. (2025, July). *Digital identity guidelines: Authentication and authenticator management* (NIST Special Publication No. 800-63B-4). National Institute of Standards and Technology (NIST). Gaithersburg, MD. https://doi.org/10.6028/NIST.SP.800-63b-4 Revision 4.

Ur, B., Kelley, P. G., Komanduri, S., Lee, J., Maass, M., Mazurek, M. L., Passaro, T., Shay, R., Vidas, T., Bauer, L., Christin, N., & Cranor, L. F. (2012). How does your password measure up? the effect of strength meters on password creation. *Proceedings of the 21st USENIX Security Symposium (USENIX Security '12)*, 65–80. https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final209.pdf