# IT Security Audit

completed by Néosoft

## REQUIREMENT YOGI PLUGINS

### EXECUTIVE REPORT

03/03/25

version 2.0

## Document history

| Version | Date | Author(s) | Description |
|---|---|---|---|
| 1.0 | 01/29/25 | Yassine BOURKI<br>Yawavi Jeona Lucie LATEVI | Document creation |
| 2.0 | 03/03/25 | Nicolas GRANDJEAN | Retest of the 03/03/2025 |

## Audit description

| | |
|---|---|
| **Client** | Requirement Yogi |
| **Type** | IT Security Audit |
| **Completed by** | Néosoft |
| **Start date** | 01/27/25 |
| **End date** | 01/29/25 |

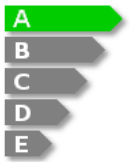
## Stakeholders

| Role | Name and company | Contact |
|---|---|---|
| Customer | Requirement Yogi<br>Jérôme RANCATI | Email:<br>jrancati@r-yogi.com |
| Audit responsible | Néosoft Cyber Data<br>Nicolas GRANDJEAN<br>Audit Division Manager | Email:<br>nicolas.grandjean@neosoft.fr<br>Phone:<br>+33 1 41 46 08 00 / +33 6 23 69 12 82 |
| Auditor | Néosoft Cyber Data<br>Yassine BOURKI<br>IT Security Auditor | Email:<br>yassine.bourki@neosoft.fr |
| Auditor | Néosoft Cyber Data<br>Yawavi Jeona Lucie LATEVI<br>IT Security Auditor | Email:<br>yawavijeonalucie.latevi@neosoft.fr |

## Global synthesis

### Audit summary

**Summary of the retest (03/03/2025):**




The global security score assigned to the service is now **A**.

The retest conducted on 03/03/2025 showed that all previously identified vulnerabilities had been corrected. In fact, the point regarding the absence of “SameSite” attribute on cookies is still active but cannot be corrected at the moment, as the Keycloak solution doesn't support it (https://github.com/keycloak/keycloak/issues/19886). As this issue is associated with a very low criticality, it was decided to rate it as false-positive.

**Summary of the initial audit (29/01/2025):**

The global security score assigned to the service is **B**. Only four vulnerabilities or areas for improvement were identified, with little real risks.

First, the password policy is insufficient: users can choose extremely weak passwords (e.g., a single character), making Brute Force attacks and account compromise through guessing attempts much easier.

Additionally, the password change functionality does not verify the old password, allowing an attacker with temporary access to a user session (e.g., via an unattended device) to modify the password and gain permanent control over the account. Furthermore, the application's cookies, including session cookies, lack the "SameSite" security attribute, increasing the risk of Cross-Site Request Forgery (CSRF) attacks that could exploit unauthorized requests.

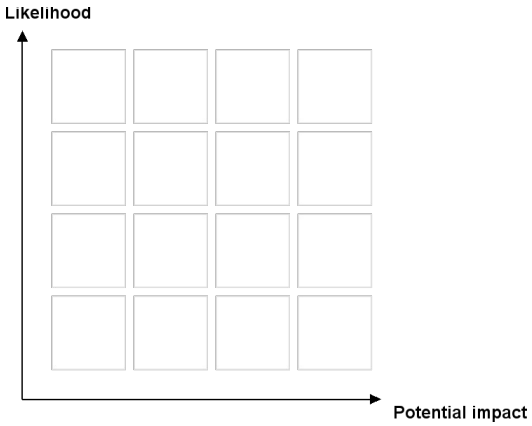
Finally, the server uses outdated TLS protocols (1.0 and 1.1), exposing data exchanges to interception and Man In The Middle (MitM) attacks.

To mitigate these risks, we recommend enforcing a stronger password policy (minimum length, complexity requirements), requiring validation of the old password before allowing a password change, configuring security attributes for cookies, and disabling older TLS versions in favor of modern protocols (TLS 1.2/1.3) with strong encryption suites.

Positive aspects were also identified, such as the absence of injection possibilities like XSS or SQLi and the absence of reported file upload vulnerabilities. In addition, the service generally adheres to security best practices, including correct input validation, secure coding standards and regular updates.

**Positive points:**

- ✔ Very limited potential attack surface
- ✔ No injections identified (XSS, SQLi, SSTI, ...)
- ✔ No access control issues found
- ✔ JWT session tokens well signed / secured
- ✔ No file upload vulnerabilities found
- ✔ Security good practices generally taken into account
- ✔ Quick implementation of the identified improvements




**Findings by criticality level:**

All the identified vulnerabilities have been fixed after the retest of the 03/03/2025.

## Compliance tables

The tables below show a recap of the compliance level of the scope with respect to the applicable standards.

The controls are listed in the left column, and the findings associated to each control are reported in the right one. In this way, a control with at least one associated audit finding will be considered as not compliant and will appear preceded by a red cross. On the contrary, a control which is not associated to any finding will be considered as compliant.

**Gaps with the "OWASP Top Ten":**

| Controls | Associated findings |
|---|---|
| OUT OF TOP 10 WEAKNESSES * | |
| **BROKEN ACCESS CONTROL**<br>Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits. | |
| **CRYPTOGRAPHIC FAILURES**<br>The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra protection, mainly if that data falls under privacy laws, e.g., EU's General Data Protection Regulation (GDPR), or regulations, e.g., financial data protection such as PCI Data Security Standard (PCI DSS). | |
| **INJECTION**<br>An application is vulnerable to attack when: user-supplied data is not validated, filtered, or sanitized by the application; dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter; hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records; hostile data is directly used or concatenated; the SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures. Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all interpreters. | |
| **INSECURE DESIGN**<br>Insecure design is a broad category representing different weaknesses, expressed as “missing or ineffective control design.” Insecure design is not the source for all other Top 10 risk categories. There is a difference between insecure design and insecure implementation. We differentiate between design flaws and implementation defects for a reason, they have different root causes and remediation. A secure design can still have implementation defects leading to vulnerabilities that may be exploited. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks. One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required. | |
| **SECURITY MISCONFIGURATION**<br>The application might be vulnerable if the application is: missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services; unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges); default accounts and their passwords are still enabled and unchanged; error handling reveals stack traces or other overly informative error messages to users; for upgraded systems, the latest security features are disabled or not configured securely; the security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values; the server does not send security headers or directives, or they are not set to secure values; the software is out of date or vulnerable (see A06:2021-Vulnerable and Outdated Components). | |

### VULNERABLE AND OUTDATED COMPONENTS

You are likely vulnerable: if you do not know the versions of all components you use (both client-side and server-side), this includes components you directly use as well as nested dependencies; if the software is vulnerable, unsupported, or out of date. This includes the OS, Web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries; if you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use; if you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion, this commonly happens in environments when patching is a monthly or quarterly task under change control, leaving organizations open to days or months of unnecessary exposure to fixed vulnerabilities; if software developers do not test the compatibility of updated, upgraded, or patched libraries; if you do not secure the components' configurations (see A05:2021-Security Misconfiguration).

### IDENTIFICATION AND AUTHENTICATION FAILURES

Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. There may be authentication weaknesses if the application: permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords; permits Brute Force or other automated attacks; permits default, weak, or well-known passwords, such as "Password1" or "admin/admin"; uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers," which cannot be made safe; uses plain text, encrypted, or weakly hashed passwords data stores (see A02:2021-Cryptographic Failures); has missing or ineffective multi-factor authentication; exposes session identifier in the URL; reuse session identifier after successful login; does not correctly invalidate Session IDs; user sessions or authentication tokens (mainly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

### SOFTWARE AND DATA INTEGRITY FAILURES

Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise. Lastly, many applications now include auto-update functionality, where updates are downloaded without sufficient integrity verification and applied to the previously trusted application. Attackers could potentially upload their own updates to be distributed and run on all installations. Another example is where objects or data are encoded or serialized into a structure that an attacker can see and modify is vulnerable to insecure deserialization.

### SECURITY LOGGING AND MONITORING FAILURES *

### SERVER-SIDE REQUEST FORGERY (SSRF)

SSRF flaws occur whenever a Web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL). As modern Web applications provide end-users with convenient features, fetching a URL becomes a common scenario. As a result, the incidence of SSRF is increasing. Also, the severity of SSRF is becoming higher due to cloud services and the complexity of architectures.

** Not applicable in the context of the current audit*