IT Security Audit

completed by Néosoft

REQUIREMENT YOGI PLUGINS

DETAILED REPORT

12/13/23

Version 1.0

Document history

Version	Date	Author(s)	Description
1.0	12/13/23	Yawavi Jeona Lucie LATEVI Mathieu GRANDMONTAGNE	Document creation

Audit description

Customer	Requirement Yogi
Type	IT Security Audit
Completed by	Néosoft
Start date	12/11/23
End date	12/13/23

Stakeholders

Role	Name and company	Contact
Customer	Requirement Yogi Jérôme RANCATI	Email: jrancati@r-yogi.com
Audit responsible	Néosoft Nicolas GRANDJEAN Audit Division Manager	Email: nicolas.grandjean@neosoft.fr Phone: +33 1 41 46 08 00 / +33 6 23 69 12 82
Auditor	Néosoft Yawavi Jeona Lucie LATEVI IT Security Auditor	Email: yawavijeonalucie.latevi@neosoft.fr
Auditor	Néosoft Mathieu GRANDMONTAGNE IT Security Auditor	Email: mathieu.grandmontagne@neosoft.fr Phone: +33 6 22 89 29 33

Table of contents

1. Context....4
1.1 Objectives....4
1.2 Approaches....4
1.3 Evaluation scales....5
1.4 Service delivery caveats....5
2. Audit scope....6
2.1 General description....6
3. Global synthesis....7
3.1 Audit summary....7
3.2 Compliance tables....9
4. Detailed results....11
4.1 Description of a typical audit finding....11
4.2 Detailed description of the audit findings....12
5. Table of findings....18
6. Table of recommendations....19

1. Context

1.1 Objectives

The purpose of the audit was to perform a technical assessment of the security of the « Requirement Yogi Plugins ». The auditors also had to provide recommendations to fix the identified vulnerabilities.

The objectives could be summarized as follows:

- **Perform a security assessment** of the « Requirement Yogi Plugins » by identifying technical vulnerabilities on the target: the purpose was **to evaluate its robustness** against realistic attacks.
- **Combine the audit notices with appropriate security criteria** by evaluating potential impact level, likelihood level, risks and technical and business impacts of each vulnerability.
- **Provide realistic and applicable technical and/or organizational recommendations** to assist both policy makers and technical teams in arbitration and implementation of corrective measures for short and mid-term.
- **Present audit conclusions in a meeting** as a **technical restitution** with the operational teams.

For this purpose, it was decided to contact an external company specialized in IT security audit and penetration testing. This document is the detailed report of the audit performed by Néosoft in this context.

1.2 Approaches

The audit only consisted in a web penetration test with a black box and grey box approach.

The approaches are the following:

- **"Black box"**: the tests are performed from the position of an external attacker without any prior knowledge of the target, or without other prior knowledge than its name or the list of its exposed services, depending on the chosen approach.
- **"Grey box"**: the auditors have access to limited information allowing them to expand the range of the possible scenarios. For example, they can have access to a global architecture diagram or have valid user accounts.

1.3 Evaluation scales

The following scale was used to evaluate the global security level of the audit scope:

Excellent:
no identified vulnerability or very uncritical vulnerabilities

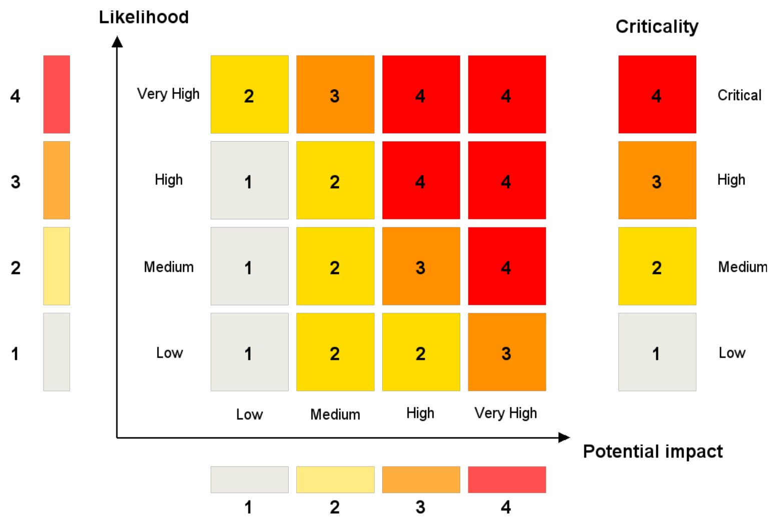
Good:
some identified vulnerabilities but with little real risks

Medium:
multiple identified vulnerabilities some of which deserve immediate attention

Low:
multiple identified vulnerabilities some of which associated to important risks

Poor:
a lot of identified vulnerabilities most of which associated to important risks

The following matrix was used to evaluate the criticality of the findings:



Likelihood		Low	Medium	High	Very High
4	Very High	2	3	4	4
3	High	1	2	4	4
2	Medium	1	2	3	4
1	Low	1	2	2	3
	Potential impact	1	2	3	4

Criticality	
4	Critical
3	High
2	Medium
1	Low

1.4 Service delivery caveats

Néosoft wishes to inform the audit sponsor that absolute completeness cannot be guaranteed. Although the approach proposed here is consistent with the state of the art in terms of penetration testing, the results reflect only what a real attacker could with limited knowledge and time. It therefore seems clear that in certain circumstances, in the case, for example, of an attacker with consequent resources, in terms of time, of people, of money, other attack vectors could potentially, but not necessarily, be identified or even exploited.

2. Audit scope

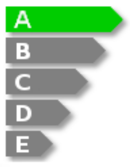
2.1 General description

The audit scope was composed of two Requirement Yogi plugins accessible at the following URLs:

- https://ww1.stg.requirementyogi.cloud/
- https://ww2.stg.requirementyogi.cloud/

3. Global synthesis

3.1 Audit summary



The global security score assigned to the service is **A**, meaning that no critical vulnerability has been identified. In fact, only 2 audit findings have been reported, one of which is more a recommendation for improvement than a real vulnerability.

It is possible to export data in Excel format. However, malicious users (including those with "user" profile) could deliberately enter malicious Excel formulas in these data in order to execute arbitrary code on the workstations of people downloading them. A security warning will probably be displayed when Excel is opened, but most users ignore these warnings. The risk would therefore be to use the application as an attack vector to compromise other users' workstations, to steal the data stored on them, or possibly to increase one's privileges on the application (**NOTICE_01**).

Finally, the Swagger of the API is accessible without authentication (**NOTICE_02**). This is not necessarily a vulnerability, and it may be acceptable if you consider that your users need access to this documentation. However, to limit the attack surface and ensure in-depth security, it may sometimes be a good idea to restrict exposure of the documentation, especially if the API exposes sensitive endpoints that can be attacked based on the documentation.

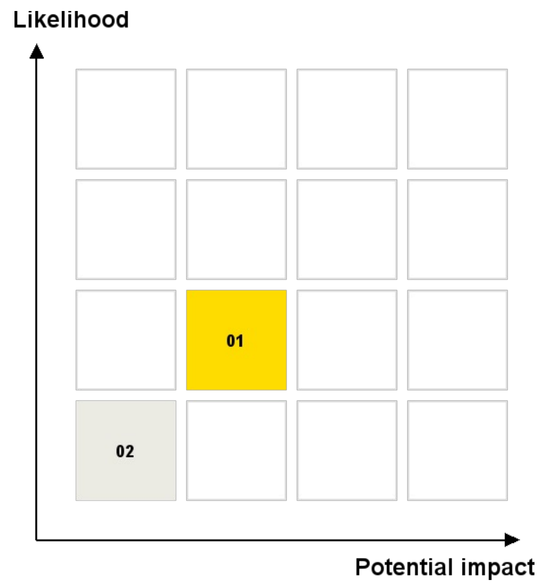
Except for these two points, the plugin's overall level of cybersecurity is very good, and the OWASP tests performed did not result in any JavaScript or server code injection, in access control or session management issues (JWT) or in any configuration problems that could be exploited by an external attacker or malicious user.

Main positive points:

- ✓ Very limited potential attack surface
- ✓ Very few audit findings
- ✓ Non-critical audit findings
- ✓ Very quick fixes proposed
- ✓ JWT session tokens well signed / secured
- ✓ Security good practices generally taken into account

Main identified weaknesses:

- ✗ Excel formula injection
- ✗ Access to the Swagger



Findings by criticality level

In conclusion, we recommend the implementation of an action plan based on the recommendations of this report.

Note: the « Table of recommendations » section of the end of the document presents a list of the main recommendations proposed by the auditors. A view of the risks after application of these measures is also provided.

3.2 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** Some weaknesses identified during penetration testing do not fall into any of the OWASP Top 10 categories. These are usually less frequent attacks but can sometimes be as serious.	
✓ **BROKEN ACCESS CONTROL** 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** 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** 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.	🟨 **NOTICE_01**
✓ **INSECURE DESIGN** 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** 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).	■ **NOTICE_02**
✓ **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*

4. Detailed results

4.1 Description of a typical audit finding

The audit results are reported under the form of technical "audit findings" presented in a standardized manner.

Each audit finding is described as follows:

[Identifier] – [Title]		
Impacted scope:	[Infrastructure / Server / Application / ...]	[Criticality]
[Security criteria]:	[Titles / Documents / Chapters / Sections / ...]	
Audit findings		
[Short description of the audit findings] Example: "the data entered in these forms are not properly validated before being used to build SQL requests on server-side" (followed by a list of concerned URLs and parameters).		
Impacts/Risks		
[Description of the main associated risks] Example: "a malicious user could enter SQL code in vulnerable forms and attempt to access the application's database containing data XXXX; please note, however, that the likelihood of this scenario is XXXX because the attacker needs to be authenticated with a privileged account."		
Proofs/Examples		
[Proofs, examples and information needed to perform the attack] Example: screenshots, scripts, etc.		
Recommendations *to reduce or remove the identified risks*		
[Summary of the main recommendations] Example: "always validate and encode all user inputs".		
[Deadline]:	[Details of the recommendations] Example: "we recommend using a specialized API to validate and encode all user inputs and thus avoid their direct inclusion in an SQL interpreter; for example, XXXX framework provides appropriate functions which could be systematically used on all HTML forms of the site. "	

4.2 Detailed description of the audit findings

NOTICE_01 – Excel formula injection

Impacted scope:	https://ww1.stg.requirementyogi.cloud/	**Criticality :**	**[2] Medium**
OWASP Top Ten:	A3. Injection	***Potential impact :***	**[2] Medium**
		Likelihood :	**[2] Medium**

Audit findings

It is possible to export data in Excel format. However, malicious users (including those with "user" profile) could deliberately enter malicious Excel formulas in these data in order to execute arbitrary code on the workstations of people downloading them. A security warning will probably be displayed when Excel is opened, but most users ignore these warnings.

Associated risks

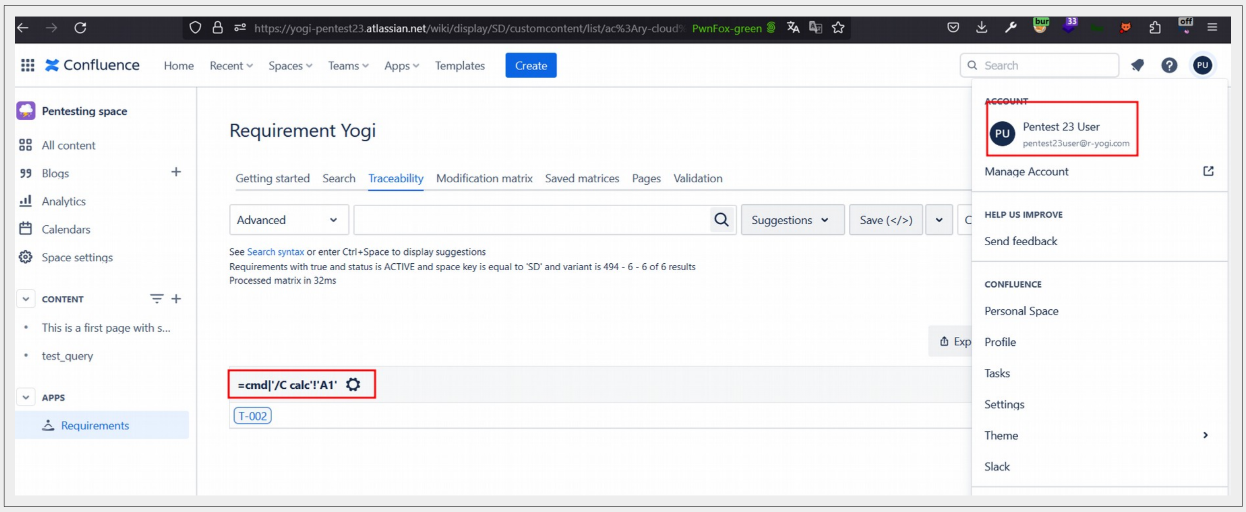
A malicious user with the profile "user" could intentionally inject malicious Excel formulas into data for subsequent export by another user, or by a privileged user such as admin or space admin. This attack can be used to execute arbitrary code on the workstations of people downloading Excel files.

The risk would therefore be to use the application as an attack vector to compromise other users' workstations, to steal the data stored on them, or possibly to increase one's privileges on the application.

Proofs/Examples

The screenshots below show an example of Excel command injection in the traceability feature label. During export, the Excel formula will be exported without any modification, and will therefore be included in the resulting file.

Here we inject a formula that will launch the "calc.exe" command (the Windows calculator) for demonstration purposes, but any other code, including malicious code, could of course be launched instead.



Requirement Yogi

Getting started | Search | Traceability | Modification matrix | Saved matrices | Pages | Validation

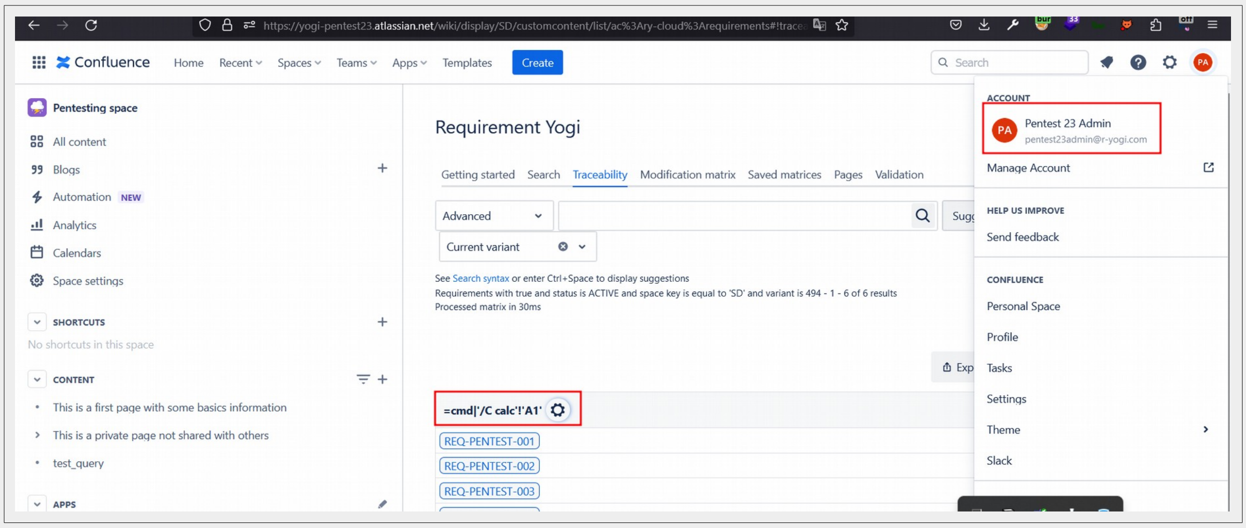
See Search syntax or enter Ctrl+Space to display suggestions
Requirements with true and status is ACTIVE and space key is equal to 'SD' and variant is 494 - 6 - 6 of 6 results
Processed matrix in 32ms

=cmd|'/C calc'!'A1'

T-002

ACCOUNT
Pentest 23 User
pentest23user@r-yogi.com

Command injection into the traceability label by the "user" account (=cmd | '/C calc'!'A1').



Requirement Yogi

Getting started | Search | Traceability | Modification matrix | Saved matrices | Pages | Validation

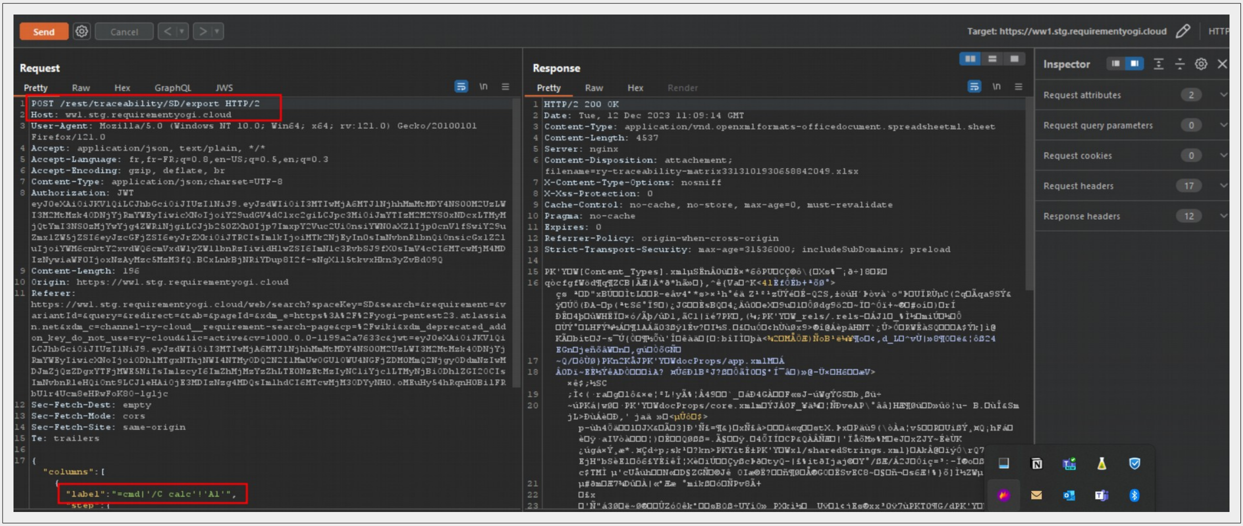
See Search syntax or enter Ctrl+Space to display suggestions
Requirements with true and status is ACTIVE and space key is equal to 'SD' and variant is 494 - 1 - 6 of 6 results
Processed matrix in 30ms

=cmd|'/C calc'!'A1'

REQ-PENTEST-001
REQ-PENTEST-002
REQ-PENTEST-003

ACCOUNT
Pentest 23 Admin
pentest23admin@r-yogi.com

The injected code is saved for all profiles, including the admin.



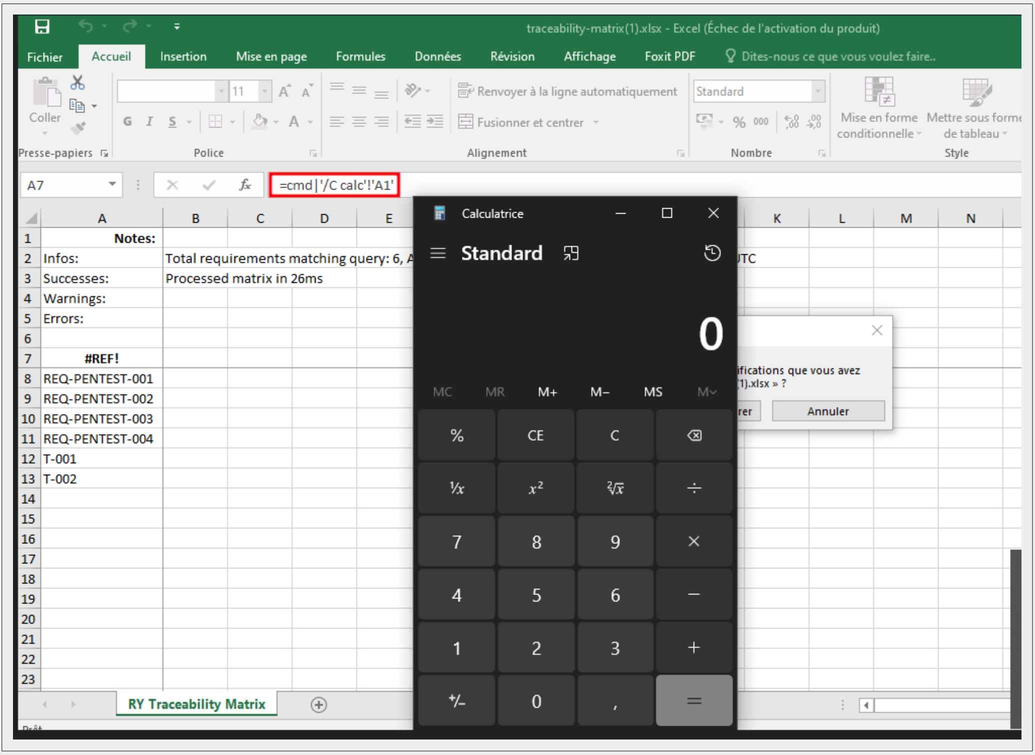
```
POST /rest/traceability/SD/export HTTP/2
Host: wwl.stg.requirementyogi.cloud
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:121.0) Gecko/20100101
Firefox/121.0
Accept: application/json, text/plain, */*
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
Content-Type: application/json;charset=UTF-8
Authorization: JWT
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI3MTIwMjA6MTJlNjhhMmMtMDY4NS00M2UzLW
I3M2MtMzk4ODNjYjRmYWEyIiwicXNoIjoiY29udGV4dC1xc2giLCJpc3MiOiJmYTIzM2M2YS0xNDcxLTMyM
jQtYmI3NS0zMjYwYjg4ZWRiNjgiLCJjb250ZXh0Ijp7ImxpY2Vuc2UiOnsiYWN0aXZlIjp0cnVlfSwiY29u
ZmxlZW5jZSI6eyJzcGFjZSI6eyJrZXkiOiJTRCIsImlkIjoiMTk2NjEyIn0sImNvbnRlbnQiOnsicGxlZ2l
uIjoiYWM6cnktY2xvdWQ6cmVxdWlyZW1lbnRzIiwidHlwZSI6ImNlc3RvbSJ9fX0sImV4cCI6MTcwMjM4MD
IzNywiaWF0IjoxNzAyMzc5MzM3fQ.BCxLnkBjNRiYDup8I2f-sNgX115tkvxHkn3yZvBd09Q
Content-Length: 196
Origin: https://wwl.stg.requirementyogi.cloud
Referer:
https://wwl.stg.requirementyogi.cloud/web/search?spaceKey=SD&search=&requirement=&v
ariantId=&query=&redirect=&tab=&pageId=&xdm_e=https%3A%2F%2Fyogi-pentest23.atlassia
n.net&xdm_c=channel-ry-cloud__requirement-search-page&cp=%2Fwiki&xdm_deprecated_add
on_key_do_not_use=ry-cloud&lic=active&cv=1000.0.0-1199a2a7633c&jwt=eyJ0eXAiOiJKV1Qi
LCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI3MTIwMjA6MTJlNjhhMmMtMDY4NS00M2UzLWI3M2MtMzk4ODNjYj
RmYWEyIiwicXNoIjoiODhlMTgxNThjNWI4NTMyODQ2N2I1MmUw0GU1OWU4NGFjZDM0MmQ2NjgyODdmNzIwM
DJmZjQzZDgxYTFjMWE5NiIsImlzcyI6ImZhMjMzYzZhLTE0NzEtMzIyNC1iYjc1LTMyNjBiODhlZGI2OCIs
ImNvbnRleHQiOnt9LCJleHAiOjE3MDIzNzg4MDQsImlhdCI6MTcwMjM3ODYyNH0.oMEuHy54hRqnHOBilFR
bUlr4Ucm8eHRwFoK80-lgljc
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers

{
  "columns":[
    {
      "label":"=cmd|'/C calc'!'A1'",
      "step":{
```

```
HTTP/2 200 OK
Date: Tue, 12 Dec 2023 11:09:14 GMT
Content-Type: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
Content-Length: 4537
Server: nginx
Content-Disposition: attachement;
filename=ry-traceability-matrix3313101930658842049.xlsx
X-Content-Type-Options: nosniff
X-Xss-Protection: 0
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
Referrer-Policy: origin-when-cross-origin
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload

PK'YDW[Content_Types].xml [illegible]
[illegible]
```

Excel file export request.



traceability-matrix(1).xlsx - Excel (Échec de l'activation du produit)

A7 =cmd|'/C calc'!'A1'

	A	B
1	Notes:	
2	Infos:	Total requirements matching query: 6, A
3	Successes:	Processed matrix in 26ms
4	Warnings:	
5	Errors:	
6		
7	#REF!	
8	REQ-PENTEST-001	
9	REQ-PENTEST-002	
10	REQ-PENTEST-003	
11	REQ-PENTEST-004	
12	T-001	
13	T-002	

We can see that the injected code runs when we open the Excel file on our workstation.

Recommendations

to reduce or remove the identified risks

Filter Excel formulas in exported files

Quick Wins:	We recommend that you clean up the entries at export time by adding a single quote at the beginning of each field. This way, Excel will interpret the cell content

Deadline: 1/3	as a string, even if it contains an "=" character. We also recommend filtering the following characters: "+ - = @". You can also set up a finer filter by checking the content of each field against the content expected as input.

NOTICE_02 – Access to the Swagger

Impacted scope:	https://ww2.stg.requirementyogi.cloud/swagger-ui/index.html	Criticality :	[1] Low
OWASP Top Ten:	A5. Security Misconfiguration	*Potential impact :*	[1] Low
		Likelihood :	[1] Low

Audit findings

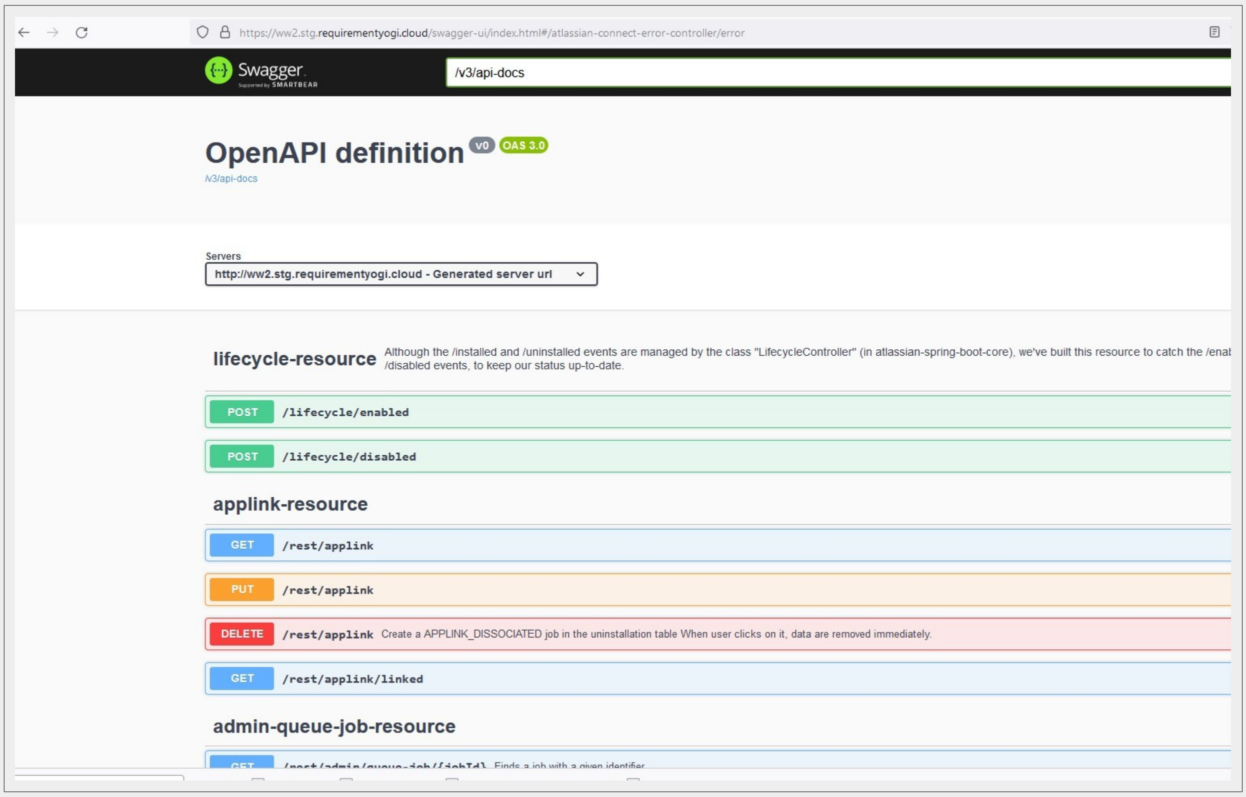
The Swagger of the plugin is accessible without authentication at the following URL: "https://ww2.stg.requirementyogi.cloud/swagger-ui/index.html".

Associated risks

This is not necessarily a vulnerability and it may be acceptable if you consider that your users need access to this documentation. However, to limit the attack surface and ensure in-depth security, it may sometimes be a good idea to restrict exposure of the documentation, especially if the API exposes sensitive endpoints that can be attacked based on the documentation.

Proofs/Examples

The screenshot below shows access to the Swagger:


https://ww2.stg.requirementyogi.cloud/swagger-ui/index.html#/atlassian-connect-error-controller/error

Swagger Supported by SMARTBEAR
/v3/api-docs

OpenAPI definition v0 OAS 3.0
/v3/api-docs

Servers
http://ww2.stg.requirementyogi.cloud - Generated server url

lifecycle-resource Although the /installed and /uninstalled events are managed by the class "LifecycleController" (in atlassian-spring-boot-core), we've built this resource to catch the /enab
/disabled events, to keep our status up-to-date.

POST /lifecycle/enabled
POST /lifecycle/disabled

applink-resource

GET /rest/applink
PUT /rest/applink
DELETE /rest/applink Create a APPLINK_DISSOCIATED job in the uninstallation table When user clicks on it, data are removed immediately.
GET /rest/applink/linked

admin-queue-job-resource

GET /rest/admin/queue-job/{jobId} Finds a job with a given identifier

Access to the Swagger.

Recommendations
to reduce or remove the identified risks

Filter access to the Swagger	
Quick Wins: Deadline: 1/3	We recommend checking if end users actually need to have direct access to the Swagger. If not, access to the Swagger should be blocked or at least limited to certain kinds of authenticated users.

5. Table of findings

Identifiers	Audit findings
🟨 **NOTICE_01**	Excel formula injection
⬜ **NOTICE_02**	Access to the Swagger

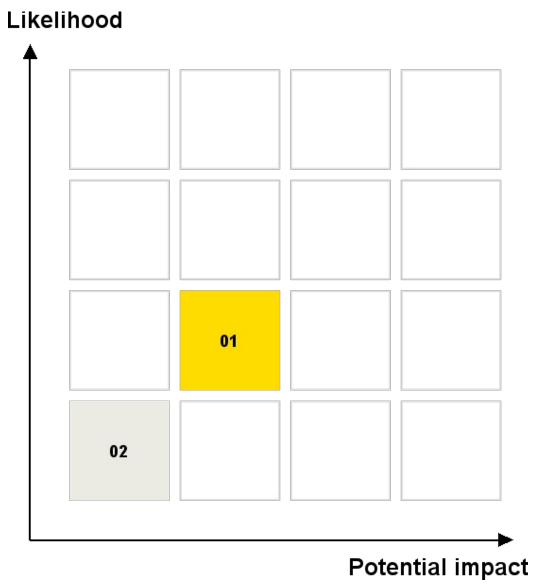
6. Table of recommendations

Identifiers		Summary of the proposed measures	QW	ST	MT
🟨	**MEASURE_01** [01]	Filter Excel formulas in exported files	x		
⬜	**MEASURE_02** [02]	Filter access to the Swagger	x		

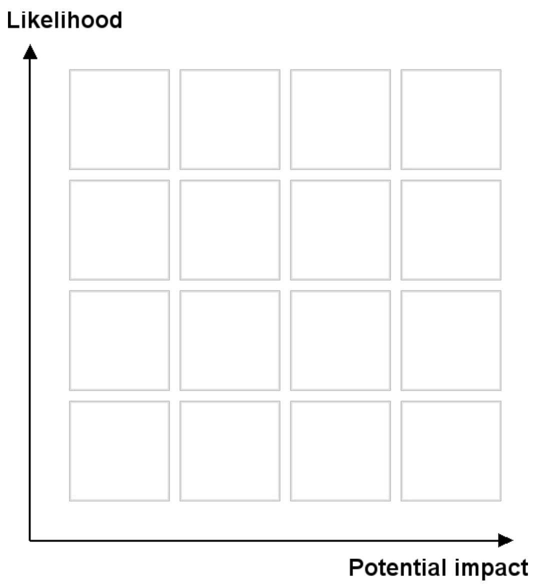
* ***QW*** **=** ***Quick Wins***: *measures likely to improve the level of security in the very short term*
* ***ST*** **=** ***Short-Term***: *measures to be applied as soon as possible*
* ***MT*** **=** ***Mid-Term***: *measures to be applied in the medium to long term*

Distribution of the findings after implementation of the Quick Wins

At the time of the audit:

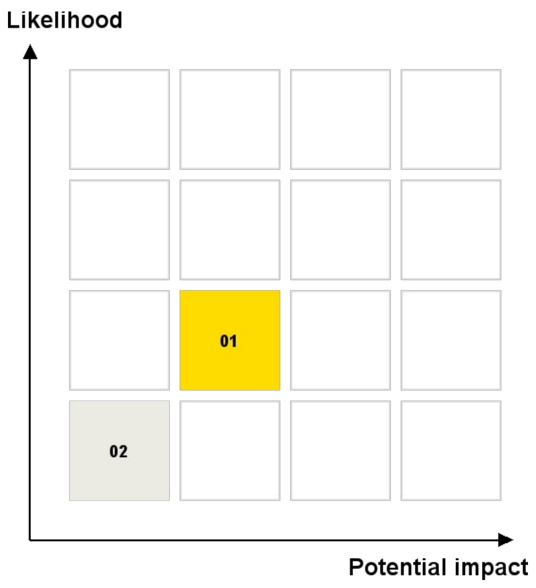


After implementation of the Quick Wins:



Distribution of the findings after implementation of the short-term measures

At the time of the audit:



After implementation of the short-term measures:

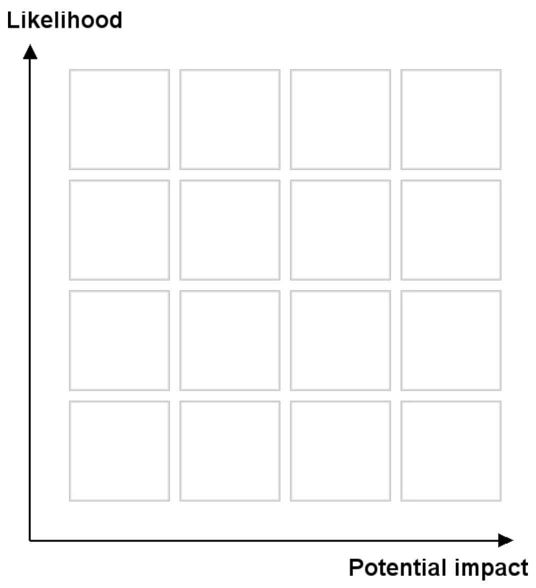


Table of contents

1. Context..4
1.1 Objectives..4
1.2 Approaches..4
1.3 Evaluation scales..5
1.4 Service delivery caveats..5
2. Audit scope..6
2.1 General description..6
3. Global synthesis..7
3.1 Audit summary..7
3.2 Compliance tables..9
4. Detailed results..11
4.1 Description of a typical audit finding..11
4.2 Detailed description of the audit findings..12
5. Table of findings..18
6. Table of recommendations..19