Security assessments for software that matters.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Pentest-Report Psono Console UI, REST API & Infra 03.2026

Cure53, Dr.-Ing. M. Heiderich, MSc. H. Moesl-Canaval, BSc. N. Aubry, BSc. M. Astner

Index

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

“Psono is an open source and self-hosted password manager to help keep your data safe. It stores your credentials encrypted and only you can access your data. Access can be shared encrypted with your team. As an open source password manager, Psono comes with a variety of features to manage your data and access your passwords more easily than ever before.”

From https://psono.com/

This report (identifiable as PSO-03 for reference) outlines the detected findings and overall verdict of a penetration test and source code audit against the Psono Console web application, as finalized in CW10 March 2026.

To offer contextual information, the assignment was requested by esaqa GmbH in February 2026 and fulfilled by four seasoned security professionals from Cure53’s talent network. twelve and a half days were invested to reach the coverage expected for this project. The frontend aspects, UI, backend components, API endpoints, and Admin components were all subject to systematic reviews.

Three unique work packages (WPs) were created for the designated testing areas. These were defined as follows:

- **WP1:** White-box pen.-tests & source code audits against Psono Console UI
- **WP2:** White-box pen.-tests & source code audits against Psono Console API
- **WP3:** White-box pen.-tests & source code audits against Psono Console Admin

To ensure a comprehensive and transparent evaluation, the engagement utilized a white-box methodology. Cure53 was granted full access to the application’s source code, target URLs, and administrative test credentials, providing the testing team with complete visibility into the environment’s internal logic.

All foundational preparations were finalized in late February 2026 (CW09), establishing a seamless transition into the active testing phase.

Collaboration was managed via a dedicated Slack workspace shared between the internal and Cure53 teams. This direct communication forum included all relevant stakeholders from both organizations, ensuring real-time alignment.

The cross-team interactions generally expedited the ongoing testing efforts. Minimal clarification questions were required and frequent status updates were provided to keep the internal maintainers in the loop.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Fortunately, no delays or hindering factors were encountered at any stage during the initiatives, testament to the ideal construction of the scope and working parameters. Live reporting was not formally integrated into the workflow for this specific engagement.

The assessors achieved satisfactory coverage across the WP1-WP3 scope items, resulting in the identification of twelve findings with varying impact on the security foundation. The tickets were categorized as three security vulnerabilities and nine miscellaneous issues.

Despite the noteworthy number of discoveries, Cure53 garnered a relatively positive opinion of the scrutinized framework. The majority of tickets pertain to general weaknesses and hardening guidance. Moreover, the maximum assigned severity rating is *Low*, indicating that the Psono Console application is fortified against significant vulnerabilities and threats.

Despite the favorable encompassing impression, the specific findings of this iteration should not be overlooked. Cure53 recommends remediating all identified issues, as even minor weaknesses can be leveraged as stepping stones for sophisticated, multi-stage attacks in the future.

Onward, the report is divided into a number of key chapters for ease of reference. Firstly, the *Scope* provides all general setup information in concise bullet points. The document then lists all security problems in chronological order of detection (rather than degree of severity), grouped into two subcategories: *Identified Vulnerabilities* and *Miscellaneous Issues*. Each corresponding ticket gives a technical explanation, Proof-of-Concept (PoC) and/or steps to reproduce, code snippets, and remedial advice. Lastly, the *Conclusions* section summarizes Cure53's overarching viewpoints of the Psono Console features in focus, as well as substantiates their security effectiveness.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Pen.-tests & code audits against Psono Console web UI, REST API & infra**
  - **WP1:** White-box pen.-tests & source code audits against Psono Console UI
    - **Application URL:**
      - https://console.stagingesaqa.com/
    - **Focus areas:**
      - General issues such as XSS, CSRF
      - Broken session management, incorrect session cookies
      - Other typical UI flaws
  - **WP2:** White-box pen.-tests & source code audits against Psono Console API
    - **API:**
      - /api/v1/*
    - **Focus areas:**
      - Broken authentication, potential flaws in OIDC implementation allowing impersonation
      - RCE
      - Permission flaws
  - **WP3:** White-box pen.-tests & source code audits against Psono Console Admin
    - **API:**
      - /api/v1/admin/*
    - **Application URL:**
      - https://console.stagingesaqa.com/administration
    - **Focus areas:**
      - See WP2
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *PSO-03-001*) to facilitate any future follow-up correspondence.

PSO-03-002 WP2-3: Template injection via billing account & project names *(Low)*

***Fix Note**: This issue has been fixed by the development team and verified by Cure53 to be working as expected. The described issue no longer exists.*

***Note:** The root cause of this issue originates from the third-party Brevo Mail library, rather than the Psono Console core logic. However, it should not be dismissed as an external issue solely. Although this flaw could be categorized as a false positive within the context of Console's direct codebase, implementing the suggested mitigations within the Psono application provides a layer of defense-in-depth. This proactive approach will ensure that the ecosystem remains resilient against future insecure updates or regressions within the Brevo integration.*

While auditing the Psono Console, the testing team noted that the billing account and project names are prone to Server-Side Template Injection (SSTI). In the event that the project or billing account name is set to *{{7*7}}*, the value is correctly displayed in the web browser, yet invitation emails display *49*. This demonstrates that the name is interpreted as code and evaluated to the value *49*, rather than treated as text.

This finding emanates from the third-party library employed to send invitation emails, Brevo Mail. While the restrictive templating language lowers the impact at present, new features or future vulnerabilities in Brevo Mail may facilitate enhanced risk, mandating the enforcement of input sanitization inside the Psono Console.

Steps to reproduce:

1. Create a new project or billing account and set the name to *{{ 7*7 }}*.
2. Navigate to the user management interface and invite a new user to the project.
3. Observe that the newly-invited user receives an email with the name of the project or billing account displayed as *49* in the subject line and email body.

To mitigate this vulnerability, Cure53 recommends implementing optimal input sanitization for the project and billing account names prior to inserting the values into mail templates.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

PSO-03-007 WP2: Incorrect permission for Psono SaaS custom domain update *(Low)*

***Fix Note**: This issue has been fixed by the development team and verified by Cure53 to be working as expected. The described issue no longer exists.*

During the static and dynamic analysis of the User API, Cure53 detected a broken access control within the SaaS custom domain configuration logic. While the */api/v1/psono-saas-custom-domain* endpoint correctly restricts the creation of custom domains to authorized roles (such as admins or operators), the implementation fails to sufficiently validate authorization for update operations.

Once a custom domain has been established, an authenticated auditor user can successfully issue a PUT request to modify the existing domain value. This access control flaw enables unauthorized users to reconfigure infrastructure settings.

Affected file:
console.esaqa.com/console/api/serializers/psono_saas_custom_domain_update.py

Affected code:
```
[...]
if not Project.has_permission(psono_saas_instance.project_id,
self.context['request'].user.id, 'read'):
    field = 'psono_saas_instance_id'
    msg = 'NO_PERMISSION_OR_NOT_EXIST'
    raise exceptions.ValidationError({field: msg})
```

Steps to reproduce:

1. Determine whether the current user is able to create a custom domain for a specific SaaS instance and verify that this action is restricted.

   Request #1:
   ```
   POST /api/v1/psono-saas-custom-domain/ HTTP/1.1
   Host: console.stagingesaqa.com

   {
   "psono_saas_instance_id":"019cb9a8-2bdd-77b4-b892-811176cd0682",
   "domain":"evil5.com"
   }
   ```

   Response #1:
   ```
   HTTP/1.1 400 Bad Request

   {
       "psono_saas_instance_id": ["NO_PERMISSION_OR_NOT_EXIST"]
   }
   ```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

2. Attempt to update a SaaS instance holding a custom associated domain.
3. Verify that this action is successful.

Request #2:

```
PUT /api/v1/psono-saas-custom-domain/ HTTP/1.1
Host: console.stagingesaqa.com

{
"psono_saas_instance_id":"019cb9a8-2bdd-77b4-b892-811176cd0682",
"domain":"evil5.com"
}
```

Response #2:

```
HTTP/1.1 200 OK

{
    "id": "019cb9af-6604-72be-ba0b-dbdd6a7b414e",
    "create_date": "2026-03-04T16:30:05.828700+00:00",
    "write_date": "2026-03-04T16:36:00.677722+00:00",
    "domain": "evil5.com",
    "status": "initializing",
    "dns_entry_1_name": "evil5.com",
    "dns_entry_1_type": "CNAME",
    "dns_entry_1_value": "forward.psono.net",
    "dns_entry_2_name": "",
    "dns_entry_2_type": "",
    "dns_entry_2_value": "",
    "dns_entry_3_name": "",
    "dns_entry_3_type": "",
    "dns_entry_3_value": ""
}
```

To mitigate this vulnerability, Cure53 recommends altering the permission checks from *read* to either *update* or *configure*, depending on the individuals permitted to update the custom domain of the SaaS instance.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

PSO-03-012 WP2: Role transition leads to administrative lockout *(Low)*

***Fix Note**: This issue has been fixed by the development team and verified by Cure53 to be working as expected. The described issue no longer exists.*

While performing static and dynamic evaluations of the project-level User API, Cure53 located a logic defect in the Role-Based Access Control (RBAC) setup. The system correctly allows users with administrative permissions to manage invites and roles within a project, yet lacks a *Last Admin* validation check. The API enables administrative users to demote their own privileges to a lower-tier role (e.g., auditor) without verifying the existence of at least one other active administrator.

If a project contains only one administrator and this user downgrades their own role, the project assumes an unmanaged state. Since the downgraded user loses the permission to revert this change or invite new admins, the project becomes permanently orphaned, requiring manual intervention by the development team to restore administrative access.

This limitation also applies to billing accounts, whereby the previous administrator can demote their role to user and essentially orphan the project.

Steps to reproduce:

1. Create a project within the Psono Console application as a user and obtain the default administrative role for this project.
2. Execute the following HTTP request to alter the role from admin to auditor:

   Request #1:
   ```
   PUT /api/v1/project-permission-user/ HTTP/1.1
   Host: console.stagingesaqa.com

   {"project_permission_user_id":"019cbe9f-f5e3-7388-9ef1-
   42d5a66cfb8e","project_role_id":"auditor"}
   ```

   Response #1:
   ```
   HTTP/1.1 200 OK

   {}
   ```

3. Attempt to invite other users via the following HTTP request and verify that the action is unsuccessful:

   Request #2:
   ```
   POST /api/v1/project-invite-user/ HTTP/1.1
   Host: console.stagingesaqa.com
   ```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de


```
{"project_id":"019cbe9f-f5de-7696-821a-
6bfcb61a3d3a","email":"hannes+test123@rs.cure53.de","project_role_id"
:"auditor"}
```


Response #2:

```
HTTP/1.1 400 Bad Request

{
    "project_id": ["NO_PERMISSION_OR_NOT_EXIST"]
}
```

To mitigate this vulnerability, Cure53 recommends incorporating a mandatory server-side validation check within the project-user management logic. The API must verify that at least one user with administrative privileges remains associated with a project or billing account prior to processing any role demotion or user removal.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

PSO-03-001 WP2: User enumeration via login *(Low)*

***Fix Note**: This issue has been fixed by the development team and verified by Cure53 to be working as expected. The described issue no longer exists.*

While assessing the login functionality, Cure53 discovered that the web application returns distinct error messages when attempting to log in with an email address that does not belong to an existing user. As such, attackers can enumerate existing users, which could assist with phishing strategies and password brute forcing.

When a non-registered email address is provided on the login page and the *Next* button is clicked, the application returns the following message: *User doesn't exist or has been deactivated.* Conversely, if the email of an existing user is provided, the *Password* textbox is displayed to the user. This clearly indicates the user's existence status.

Steps to reproduce:

1. Ensure that you are logged out of the application and navigate to the login page.
2. Enter an email address that does not belong to any user.
3. Click the *Next* button.
4. Observe that the application displays the error message: *User doesn't exist or has been deactivated*.
5. Enter an email address that belongs to an existing user and click *Next.*
6. Verify that the application proceeds to the next step and displays the *Password* text input.

To mitigate this issue, Cure53 recommends ensuring that the application forwards the user to the password step without checking the validity of the email address. Once the password is submitted, the validity of both login values should be verified and a generic error message returned if the email address and/or the password is incorrect, regardless of which entity is invalid.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

PSO-03-003 WP1-3: Outdated & vulnerable dependencies *(Info)*

***Fix Note**: This issue has been fixed by the development team and verified by Cure53 to be working as expected. The described issue no longer exists.*

Cure53's analysis of the application's dependencies revealed that several integrated software packages operate obsolete versions, many of which are associated with documented security vulnerabilities. The following components have been identified as outdated and represent a potential risk to the environment's integrity. The exploitation and impact of these vulnerabilities depends on the degree of the functionality's usage within the application under the current construct.

Command:
```
$ trivy fs --severity HIGH,CRITICAL,MEDIUM .
```

Affected files:
frontend-admin/package-lock.json:

Name	ID	Installed Version	Fixed Version
@babel/runtime @babel/runtime-corejs3	CVE-2025-27789	7.24.7	7.26.10, 8.0.0-alpha.17
ajv	CVE-2025-69873	6.12.6	8.18.0, 6.14.0
js-yaml	CVE-2025-64718	4.1.0	4.1.1, 3.14.2
serialize-javascript	GHSA-5c6j-r48x-rmvq	6.0.2	7.0.3
webpack	CVE-2024-43788	5.92.0	5.94.0

frontend/package-lock.json:

Name	ID	Installed Version	Fixed Version
@babel/runtime @babel/runtime-corejs3	CVE-2025-27789	7.24.7	7.26.10, 8.0.0-alpha.17
ajv	CVE-2025-69873	6.12.6	8.18.0, 6.14.0
js-yaml	CVE-2025-64718	4.1.0	4.1.1, 3.14.2
minimatch	CVE-2026-26996, CVE-2026-27903, CVE-2026-27904	3.1.2	10.2.3, 9.0.7, 8.0.6
serialize-javascript	GHSA-5c6j-r48x-rmvq	6.0.2	7.0.3
webpack	CVE-2024-43788	5.92.0	5.94.0

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Requirements.txt:

Name	ID	Installed Version	Fixed Version
django	CVE-2026-25673	5.2.11	6.0.3, 5.2.12, 4.2.29

Significantly, the testing team was unable to estimate the full extent of the potential implications within the restricted time frame of this engagement. The wider security ramifications remain unverified and should be reviewed by the in-house team to identify any prevalent risks.

Establishing robust supply chain security to an optimal standard is inherently complex, as a definitive and singular fix solution is rarely available. Moreover, the effectiveness of any protective framework is highly dependent on the specific versions of integrated libraries and their unique implementation within the environment.

To mitigate these issues, Cure53 recommends upgrading all affected libraries and establishing a policy to ensure that they remain up-to-date moving forward. This approach will ensure that Psono Console can benefit from remediations that have been applied for previously identified vulnerabilities across other external constructs.

PSO-03-004 WP2: Hardcoded extended certificate validity period *(Info)*

***Fix Note**: This issue has been fixed by the development team and verified by Cure53 to be working as expected. The described issue no longer exists.*

While probing the Psono backend utilities, the testers noticed that the validity of the generated self-signed SSL certificates is hardcoded to 100 years, which does not comply with current industry standards and best practices.

Firstly, the extended validity period fails to account for cryptographic obsolescence. Contemporary encryption algorithms such as RSA-2048 are expected to become vulnerable, as computational power and quantum advancements will emerge long before the certificate expires.

Secondly, excessive lifespans significantly increase the window of opportunity for an attacker following a private key compromise, since a reliable mechanism of enforcing revocation across all clients for a century-long duration is infeasible.

Lastly, hardcoding a 100-year duration bypasses the security benefits of regular identity revalidation and violates modern browser trust policies, which generally enforce a maximum validity of 398 days to ensure operational security agility.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected file:
console.esaqa.com/console/backend/utils/various.py

Affected code:
```
[...]
def generate_self_signed_certificate(domain, country='DE', key_size=2048):
    [...]
    not_valid_before = datetime.now(timezone.utc)
    not_valid_after = not_valid_before + timedelta(days=365 * 100)

    certificate = (
        x509.CertificateBuilder()
        .subject_name(subject)
        .issuer_name(issuer)
        .public_key(private_key.public_key())
        .serial_number(x509.random_serial_number())
        .not_valid_before(not_valid_before)
        .not_valid_after(not_valid_after)
        .add_extension(
            x509.SubjectAlternativeName([x509.DNSName(domain)]),
            critical=False,
        )
        .sign(private_key, hashes.SHA256())
    )
[...]
```

To mitigate this issue, Cure53 suggests monitoring the potential security risks associated with significant SSL certificate validity lengths. Certificates should be regularly reviewed and renewed for optimal security. In general, a 1-2 year validity period would be appropriate in this context.

PSO-03-005 WP2: Non-constant time comparison of access keys *(Info)*

***Fix Note**: This issue has been fixed by the development team and verified by Cure53 to be working as expected. The described issue no longer exists.*

While vetting the HTTP authentication implementation, Cure53 identified inconsistencies in the token comparison protocols. The *CronAuthentication* class utilizes a time-constant compare function to prevent time-based side channel attacks on the authentication token. The *OdooAuthentication*, *ManagementCommandAuthentication*, and *JobServerAuthentication* classes leverage a simple string comparison that halts upon the first character mismatch.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

In theory, this situation can be abused to brute force the token character by character by analyzing the execution time of the authentication function. In practice, this pitfall is non-exploitable (at least as a remote attacker) owing to the inconsistent network delay and latency. Nonetheless, Cure53 deemed it apt to include this ticket in the report with an *Informational* severity rating for completeness reasons.

Affected file #1:
console.esaqa.com/console/odoo/authentication.py

Affected code #1:
```
[...]
if not odoo_access_key or odoo_access_key != settings.ODOO_ACCESS_KEY:
[...]
```

Affected file #2:
console.esaqa.com/console/api/authentication.py

Affected code #2:
```
[...]
if not management_command_access_key or management_command_access_key !=
settings.MANAGEMENT_COMMAND_ACCESS_KEY:
[...]
if not job_server_access_key or job_server_access_key !=
settings.JOB_SERVER_ACCESS_KEY:
[...]
```

To mitigate this issue, Cure53 suggests integrating constant-time comparison functions for all authentication tokens. This will eliminate the risk of potential timing side-channel attacks and align with industry-standard defensive coding practices.

PSO-03-006 WP1-2: Asynchronous payment state for Stripe *(Low)*

***Fix Note**: This issue has been fixed by the development team and verified by Cure53 to be working as expected. The described issue no longer exists.*

Cure53's static analysis of the Stripe integration revealed that the payment lifecycle relies on client-side orchestration. Here, the frontend is responsible for notifying the backend of transaction outcomes via the */update-stripe-payment* endpoint upon an authentication error (e.g., 3D Secure failure).

This architecture creates a trust boundary violation and induces the risk of data desynchronization. If a user's browser session terminates (due to a crash, network timeout, or hardware failure) immediately after a successful 3D Secure authentication but before the frontend can signal the backend, the system enters an inconsistent state.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

In this scenario, the customer is successfully charged by Stripe, yet the internal database record remains unpaid in the backend due to the absent */update-stripe-payment* API call.

Affected file:
console.esaqa.com/frontend/src/views/invoice/invoiceListTable.js

Affected code:

```
const onPay = (rowData) => {
    setBackdropOpen(true)
    setError("")
    invoiceService.payInvoice(rowData[0]).then(
        (result) => {
            setBackdropOpen(false)
            const stripe_payment_id = result.stripe_payment_id;

            if (result.is_successful) {
                loadInvoices();
            } else if (result.hasOwnProperty("provider") && result.provider
=== 'stripe' && result.hasOwnProperty("stripe_payment_error_code")) {
                if (result.stripe_payment_error_code ===
"authentication_required") {

                [...]

                 if (result.paymentIntent.status === "succeeded") {
                        // The payment is complete!

stripeService.updateStripePayment(stripe_payment_id)

                        [...]
                }
            }
}
```

To mitigate this issue, Cure53 recommends asserting that the application completes the implementation of Stripe webhooks, which currently remains in a partially finished state. Cryptographic signature verification must be incorporated using the *stripe-signature* header and official Stripe SDK to prevent attackers from spoofing payment events. By finalizing this integration and ensuring that the handler is idempotent, the system will maintain data integrity even if the client's browser crashes, as Stripe's servers will independently and reliably notify the backend of the final transaction state.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

PSO-03-008 WP1-2: Lack of runtime secret rotation capability *(Info)*

***Fix Note**: This issue has been fixed by the development team and verified by Cure53 to be working as expected. The described issue no longer exists.*

During the static analysis of the Psono Console application and connected Django configuration, Cure53 ascertained that several authentication tokens are utilized that are not managed by Django, such as tokens for *Odoo*, *cron*, *management commands*, and *job server*. These tokens are integrated into the *settings.py* configuration file and are read from environment variables, which are loaded into memory during Django initialization.

Django settings are immutable following process commencement; as such, rotating a compromised token requires a full application service restart. This creates a security-vs-availability conflict, whereby an organization may delay rotation to avoid service downtime, thereby extending the window of opportunity for an attacker to exploit leaked credentials.

Affected file:
console.esaqa.com/console/console/settings.py

Affected code:

```
def config_get(key, *args):
    if 'ESAQACONSOLE_' + key in os.environ:
        val = os.environ.get('ESAQACONSOLE_' + key)
        try:
            json_object = json.loads(val)
        except ValueError:
            return val
        return json_object
    if key in config:
        return config.get(key)
    if len(args) > 0:
        return args[0]
    raise Exception("Setting missing", "Couldn't find the setting for
%s ..." % (key,))

[...]
ODOO_ACCESS_KEY = config_get('ODOO_ACCESS_KEY', '')
JOB_SERVER_ACCESS_KEY = config_get('JOB_SERVER_ACCESS_KEY', '')
CRON_ACCESS_KEY = config_get('CRON_ACCESS_KEY', '')
MANAGEMENT_COMMAND_ACCESS_KEY = config_get('MANAGEMENT_COMMAND_ACCESS_KEY',
'')
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

To mitigate this issue, Cure53 recommends adopting a dynamic secret management system (i.e., secure vault or encrypted database) to replace static environment variables. Decoupling sensitive credentials from the Django process lifecycle ensures that compromised tokens can be rotated immediately upon discovery. This approach facilitates rapid remediation of security incidents without interrupting the operational uptime of Psono Console services.

PSO-03-009 OOS: Reauthentication not required for authentication change *(Low)*

***Fix Note**: This issue has been mitigated by the development team and verified by Cure53 to be working as expected.*

***Note:** This issue has been categorized as Out-Of-Scope (OOS). Nevertheless, Cure53 decided to include a dedicated ticket in the report to provide a holistic security overview.*

Cure53 discovered that a user with Two-Factor Authentication (2FA) can amend their email address and alter/disable their second factor without needing to reauthenticate. In general, web applications should force users to reauthenticate when performing sensitive operations such as amending authentication methods.[1]

For users with 2FA enabled, reauthentication must request their password and a one-time code, neither of which are stipulated under the current implementation. This beneficial approach will ensure that attackers are not privy to these values, thus constraining the threat of account takeovers via session cookie theft or physical computer access.

Steps to reproduce:

1. Log in as any user.
2. Navigate to the *Account* section by clicking on the icon in the top-right corner.
3. Set up 2FA in the *Two Factor* tab (if not already configured).
4. Disable the second factor and observe that neither the password nor one-time code is required.
5. Alter the email address in the *Profile* tab and note that neither the password nor one-time code is required to complete this action.
6. Verify that an attacker can successfully take over the account via the *Forgot password* functionality by amending the associated password.

To mitigate this issue, Cure53 recommends forcing users to re-navigate through the login process for any authentication method modification. The process must also require the one-time code for users with 2FA enabled, in alignment with the typical login protocols already deployed.

[1] https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

PSO-03-010 WP2: Partial DB scheme enumeration via *orderBy* *(Low)*

***Fix Note**: This issue has been fixed by the development team and verified by Cure53 to be working as expected. The described issue no longer exists.*

While exploring the Admin dashboard, Cure53 determined that the sort-by-column functionality on multiple tables directly adopts the column name from the HTTP GET *ordering* parameter without any sanitization or whitelist filtering. This enables ordering the tables with hidden columns that are not returned to the UI, as well as columns from other tables with references to the original table.

The ORM framework throws an exception when a non-existent column name is specified, resulting in an internal server error (HTTP status code 500). This behavior facilitates enumerating hidden fields from the affected table and all related tables, which is achievable by brute forcing the *ordering* parameter with common table and column names (e.g., *password*).

Request #1 (existing hidden column):

```
GET /api/v1/admin/user/?page=0&page_size=10&ordering=password HTTP/2
Host: console.stagingesaqa.com
Cookie: csrftoken=[...]; __stripe_mid=[...]; sessionid= [...];
__stripe_sid=[...]
User-Agent: [...]
Referer: https://console.stagingesaqa.com/administration/user/
X-Csrftoken: [...]
Content-Type: application/json
```

Response #1:

```
HTTP/2 200 OK
Date: Fri, 06 Mar 2026 09:17:31 GMT
Content-Type: application/json
Server: cloudflare

{"count":8,"results":[{"id":"019c8e78-8dbb-70df-a77c-
d8590036a768","email":"user@example.com","is_email_verified":true,"is_super
user":false,"is_staff":false,"is_active":true},{"id":"019cae9d-6bd1-7352-
bd67-8a455d116c65","email":"nicolas+psono-
ato@rs.cure53.de","is_email_verified":true,"is_superuser":false,"is_staff":
false,"is_active":true},[...]
```

Request #2 (non-existent column):

```
GET /api/v1/admin/user/?page=0&page_size=10&ordering=nonExistentColumn
HTTP/2
Host: console.stagingesaqa.com
Cookie: csrftoken=[...]; __stripe_mid=[...]; sessionid=[...]
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
User-Agent: [...]
Referer: https://console.stagingesaqa.com/administration/user/
X-Csrftoken: [...]
Content-Type: application/json
```

Response #2:

```
HTTP/2 500 Internal Server Error
Date: Fri, 06 Mar 2026 09:20:17 GMT
Content-Type: text/html; charset=utf-8
Server: cloudflare

<!doctype html>
<html lang="en">
<head>
  <title>Server Error (500)</title>
</head>
<body>
  <h1>Server Error (500)</h1><p></p>
</body>
</html>
```

Additionally, information can be implicitly leaked from the tables by analyzing the results of the ORM-created table joins when ordering with a column via another table or the result orders. To demonstrate this activity, Cure53 ordered users by *id* from the *billing_account_permission* table, which is related to the user table. This query joins the two tables and leaks the number of billing accounts per user. The response contains *admin@example.com* twice, disclosing that two billing accounts are active.

Request #3:

```
GET /api/v1/admin/user/?
page=0&page_size=10&ordering=billing_account_permission_users__billing_acco
unt__id HTTP/2
Host: console.stagingesaqa.com
Cookie: csrftoken=[...]; __stripe_mid=[...]; sessionid=[...]
User-Agent: [...]
X-Csrftoken: [...]
Content-Type: application/json
```

Response #3:

```
HTTP/2 200 OK
Date: Fri, 06 Mar 2026 10:12:44 GMT
Content-Type: application/json
Server: cloudflare

{"count":8,"results":[{"id":"57d72e2b-4d8b-44c0-b9a6-
03d8b9e9309b","email":"admin@example.com","is_email_verified":true,"is_supe
ruser":true,"is_staff":true,"is_active":true},{"id":"019cae9d-6bd1-7352-
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
bd67-8a455d116c65","email":"nicolas+psono-
ato@rs.cure53.de","is_email_verified":true,"is_superuser":false,"is_staff":
false,"is_active":true},{"id":"57d72e2b-4d8b-44c0-b9a6-
03d8b9e9309b","email":"admin@example.com","is_email_verified":true,"is_supe
ruser":true,"is_staff":true,"is_active":true},[...]
```

This technique can be employed to enumerate vast segments of the database structure. Referencing related tables in the *ordering* parameter allows itemizing corresponding relationships and columns. Moreover, threat actors could analyze the order of the results by creating projects with names such as *aaa*, *aab*, and *aac* to leak contents of specific database fields (e.g., the password hash).

Albeit, Cure53 could not craft other advanced payloads due to time constraints. In addition, the affected endpoints require admin privileges, explaining the ticket's severity rating of *Low*.

Affected file #1:
console.esaqa.com/console/api/v1/admin/user.py

Affected code #1:
```
[...]
if ordering:
       users_qs = users_qs.order_by(ordering)
[...]
```

Affected file #2:
console.esaqa.com/console/api/v1/admin/psono_license.py

Affected code #2:
```
[...]
if ordering:
       psono_licenses_qs = psono_licenses_qs.order_by(ordering)
[...]
```

Affected file #3:
console.esaqa.com/console/api/v1/admin/psono_saas_instance.py

Affected code #3:
```
[...]
if ordering:
       psono_saas_instances_qs = psono_saas_instances_qs.order_by(ordering)
[...]
```

While the audit team was only able to confirm the aforementioned entities owing to restrictions in the staging environment and time constraints, all table sorts on the administration dashboard are likely affected by this activity.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

To mitigate this issue, Cure53 recommends limiting the input values of the *ordering* parameter via a whitelist containing the column names displayed on the webpage exclusively. This will prevent perpetrators from enumerating additional fields of the utilized table, as well as other tables referenced from this counterpart.

PSO-03-011 WP2-3: SaaS-Instance admin password displayed in cleartext *(Info)*

***Fix Note**: This issue has been mitigated by the development team and verified by Cure53 to be working as expected.*

***Note:** the client confirmed that this behavior is intentional and does not represent a security issue from the perspective of Psono Console. Nevertheless, Cure53 opted to include this ticket with an* Informational *severity rating for completeness reasons*.

In the event that a new Psono-SaaS-Instance is created, a password for the administrator account is generated and visible in the web UI, which is accessible via the API for all users with the *read* permission on the respective project.

All project users should be able to login as an administrator in the instance. However, the storage and retrieval of the password in cleartext, as well as the absence of a forced password modification upon the first login, exacerbates the risk of leaks and disclosure to unauthorized users.

Furthermore, the */api/v1/psono-saas-instance/<instance ID>/* API endpoint also returns a *private_key_pem* field containing a private key. While the generation and sharing of this private key is necessary to allow non-technical users to set up the SAML connection, it is returned in cleartext via the API to all users with *read* access, which also raises the probability of disclosure to unauthorized users.

Steps to reproduce:

1. Create a new SaaS-Instance as a user with project administrative permissions.
2. Wait until the instance is ready (status: *enabled*), then log in as a user with the auditor role for the project.
3. Navigate to the *Psono SaaS* page and click the instance's edit button.
4. Review the *Default credentials* section and note that the password for the administrator can be viewed in cleartext by the current non-admin user.

Requests:

```
GET /api/v1/psono-saas-instance/019cb917-b8b6-7283-a2fa-ca6d7f55308b/
HTTP/2
Host: console.stagingesaqa.com
Cookie: csrftoken=<REDACTED>;sessionid=<REDACTED>
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:148.0)
Gecko/20100101 Firefox/148.0
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
X-Csrftoken: <REDACTED>
Content-Type: application/json
```

Response:

```
{
       "Id":"019cb917-b8b6-7283-a2fa-ca6d7f55308b",
       "create_date":"2026-03-04T13:44:25.529464+00:00",
       [...]
       "webclient_url":"https://mysaas.stagingesaqa.com",
       "server_url":"https://mysaas.stagingesaqa.com/server",
       "admin_portal_url":"https://mysaas.stagingesaqa.com/portal",
       "Public_key":"",
       "default_admin_username":"admin@mysaas.stagingesaqa.com",
       "default_admin_password":"Aspgg<REDACTED>",
       [...],
       "private_key_pem":"-----BEGIN PRIVATE KEY-----\nMIIEv<REDACTED>\
       n-----END PRIVATE KEY-----\n",
       [...]
}
```

Affected file:
console.esaqa.com/console/api/v1/psono_saas_instance.py

Affected code:

```
def get(self, request, psono_saas_instance_id=None, *args, **kwargs):
    """
    Returns all or a specific Psono SaaS instance

[...]

        return Response({
            'id': str(psono_saas_instance.id),
            'create_date': psono_saas_instance.create_date.isoformat(),
[...]
            'default_admin_username':

psono_saas_instance.default_admin_username,
            'default_admin_password':

psono_saas_instance.default_admin_password,
[...]
            'certificate_pem': psono_saas_instance.certificate_pem,
            'private_key_pem':

decrypt_with_db_secret(psono_saas_instance.private_key_pem)
                                if psono_saas_instance.private_key_pem else
'',
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
[...]
        }, status=status.HTTP_200_OK)
```

To mitigate this issue, Cure53 recommends removing the default administrator password entirely and replacing it with a single-use invitation link sent to the administrator user via email upon instance creation. The administrator should be forced to set up a password for their account. Subsequent users should gain access via invitation links that the initial administrator user sends by email and follow a similar onboarding process as the account password configuration. Accordingly, passwords with extended lifetimes would never be visible to multiple users, neutralizing the risk of compromise.

The current default password implementation could be retained; however, the password should only be visible for administrators and solely usable for the first login. Here, the administrator must be forced to set a password (in alignment with the process described above).

Moreover, the internal team should reevaluate the necessity of returning the *private_key_pem* field to all project users via the API, as well as consider all associated risks. A single-use download link for the private key could be employed to avoid long-term server storage. The ability to generate a new key pair should be provided in the event of a download malfunction or user loss.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

This security evaluation marks the third assessment conducted by Cure53 against the Psono ecosystem.

The engagement was commissioned for granular reviews of the Console application, organized into three distinct WPs to ensure encompassing coverage of the attack surface. WP1 targeted the frontend and session management layers, specifically investigating UI-driven vulnerabilities such as XSS and CSRF, as well as the integrity of authentication mechanisms and OIDC implementation. WP2 shifted focus to the user-facing API, in which the premise was scrutinized regarding the logic governing data access and potential permission flaws. WP3 was dedicated to the administrative components, whereby Cure53 appraised the security posture of the management interface and associated privileged API endpoints.

To facilitate an efficient and in-depth audit, the client provided definitive access to the source code and highlighted specific areas of concern prior to initiation. This transparency proved highly beneficial, as it allowed the testers to bypass the initial discovery phase and quickly familiarize themselves with the internal logic. By leveraging this white-box approach, the assessment sought to locate surface-level and deeply entrenched vulnerabilities. Moreover, complex architectural concerns were addressed, ensuring that the findings reflected a thorough understanding of the Psono Console's underlying codebase and operational environment.

The collaboration with esaqa GmbH was characterized by first-rate operational efficiency. The engagement was supported by a dedicated Slack channel, which facilitated immediate technical assistance and streamlined communication both during the preparatory phase and throughout the active security audit.

The conclusory outcomes of this project primarily indicate adequate robustness and sound design implementations. Twelve limitations were documented in total; three of the tickets were categorized as exploitable vulnerabilities with only minor severity, while nine entail defense-in-depth suggestions.

While investigating the Psono Console UI (WP1), Cure53 identified that the backend architecture is realized as a Django application written in Python, while the frontend utilizes the React framework. The use of React inherently neutralizes several classes of common web vulnerabilities; for instance, traditional DOM-based XSS is significantly harder to achieve, as the framework does not leverage HTML-esque strings for rendering and automatically escapes data by default.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The static code review did not detect any usage of insecure client-side functions (e.g., *dangerouslySetInnerHTML*), which would otherwise bypass React's built-in protections. By strictly adhering to standard JSX data binding, the frontend effectively ensures that all dynamic content is automatically escaped before being rendered into the DOM. This avoidance of direct HTML injection sinks significantly hardens the client-side application against common injection vectors, as it forces all user-supplied data to be treated as literal text rather than executable code.

The OIDC integration into Django was diligently analyzed to ensure optimal safeguarding of the authentication handshake and subsequent session management. The implementation successfully deters common delegation flaws, such as session fixation, state parameter manipulation, and authorization code injection. The exchange of identity tokens and transition to local session states conform to industry best practices, effectively preventing unauthorized account takeovers and authentication flow hijacking.

In addition to the specific functional audits, the general Django configuration was rigorously probed regarding the middleware stack and enforcement of security headers, such as X-Content-Type-Options and Content-Security-Policy. The setup is sufficiently hardened; Cure53 did not locate any malpractices or omissions in this realm. The uniform security controls demonstrate a mature approach to framework-level defense, ensuring that the underlying infrastructure provides a secure foundation for the application logic.

The application was extensively tested for other injection vulnerabilities, e.g., template injection. Similarly, no erroneous behaviors of this ilk were observed in the application. However, the encoding of user input prior to web page insertion facilitates integrating arithmetic and logic operations into project invite emails (PSO-03-002). The observed flaw can be attributed to the third-party Brevo Mail dependency. While Psono Console utilizes this library for mail services, the weakness exists within the external module's implementation, rendering it a downstream security concern rather than a native defect of the Console ecosystem.

The development team utilizes tools such as Trivy and Bandit to pinpoint outdated and vulnerable third-party software components. Nevertheless, certain libraries are susceptible to known risks (PSO-03-003). Pertinently, one of the vulnerabilities was published during the active review stage.

All requests that perform state-changing operations avoid leveraging the GET method. Moreover, all requests of this nature are adequately protected against CSRF attacks via the adoption of the Django framework's built-in CSRF protections.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The session management protocols are properly implemented, with session cookies invalidated upon user logout. However, alterations to authentication methods such as the email address and second factor do not require reauthentication, which deviates from security best practices (PSO-03-009). Albeit, this deficiency was deemed OOS for this PSO-03 task.

Regarding WP2, the API was thoroughly reviewed via white-box tactics, with particular scrutiny of the deployed access control mechanisms. Dynamic testing was applied to verify pitfalls detected during the code audit. In summary, the API is secure and the vast majority of endpoints enforce RBAC in an astute manner. Nevertheless, the endpoint to update custom domain information of SaaS instances allows write operations for all users with read access (PSO-03-007).

The database interactions were also vetted. The application utilizes Django's ORM framework and refrains from constructing SQL queries via string concatenation. The consistent and correct usage of the ORM specific syntax to read, insert, and update database entries nullifies the plausibility of SQL injection. The white-box review confirmed that the application relies on the ORM framework to handle malicious inputs. An adversary can manipulate SQL queries by specifying arbitrary column names in the result sorting parameter. This allows enumerating parts of the database and could potentially leak contents, although this impact could not be proven during the allocated analysis time frame (PSO-03-010).

Elsewhere, an RBAC-related fault was located concerning project permission management, whereby administrators are granted the ability to revoke their own privileges. As a result, a project could be completely locked out due to the lack of administrator oversight, which also affects billing account permissions (PSO-03-012). Besides the described activities, no other shortcomings were detected from an RBAC perspective.

The only other limitation pertained to insecure authentication token comparisons in the backend source code (PSO-03-005). Generally, the avoidance of major detriments in this area is commendable.

With respect to the Psono Console Admin components (WP3), Cure53 determined that the API returns the default password of the SaaS instance administrator in cleartext to all users with read access (PSO-03-011). While the in-house team confirmed that this is intentional, the implementation could be augmented for airtight security.

The administration API was explored for potential input validation deficiencies and critical execution flaws. Testing verified that the endpoints are suitably shielded against common injections; no viable pathways for RCE were detected.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Dynamic evaluations of the platform's access control mechanisms did not identify any Insecure Direct Object References (IDOR) vectors. The application consistently validates authorization boundaries, preventing users from performing unauthorized actions even when supplied with valid UUIDs belonging to other users, billing accounts, or projects to which they have not been granted access. This robust isolation ensures that object identifiers alone are insufficient to circumvent security constraints. Furthermore, the granular validation of functional roles within individual projects is effective and correctly limits privileged operations to authorized personnel. Non-administrative project members are successfully blocked from executing restricted actions, confirming that the backend appropriately enforces the principle of least privilege across all tested scopes.

Lastly, WP3 encompassed close inspections of several auxiliary system interfaces, including those facilitating communication with Odoo, the internal cron scheduling system, various management commands, and the job server. These endeavors confirmed that the components maintain a high degree of structural integrity and isolation. Beyond negative practices related to the handling of static authentication tokens (PSO-03-008) and the omission of constant time comparison (PSO-03-005), the testing team was unable to identify any other weakness or logic mishaps within these API implementations.

To finalize, Cure53 is pleased to report that the Psono Console yielded a favorable verdict during this initiative. None of the findings exceeded an impact score of *Low*. A multitude of typical errors and vulnerabilities are successfully negated, reflecting the development team's acute awareness of web application security.

Cure53 would like to thank Sascha Pfeiffer from the esaqa GmbH team for his excellent project coordination, support, and assistance, both before and during this assignment.