Solutions Architecture Document: Loan Process Automation Service Modernization with Notes

Solutions Architecture Document: Loan Process Automation Service Modernization

Author:	Lex, Principal Infrastructure Engineer and Architect
Date:	August 4, 2025
Version:	1.2
Status:	Draft

1.0 Executive Summary

The "Loan Process Automation" business service is a critical function that currently operates without a structured Software Development Lifecycle (SDLC), formal version control, or adequate security and access controls. This presents significant operational, security, and compliance risks, notably failing to align with NCUA requirements and internal CIS Controls.

This document outlines a proposed future state architecture and a phased action plan to address these deficiencies. The core of the proposal is to introduce foundational SRE and DevOps practices by establishing a secure, role-based file access model, implementing version control for source code, and creating a formal, manifest-driven release process for all application components. This will be supported by deployment automation and a robust data sanitization process for non-production environments.

Successful implementation will result in a more secure, stable, and compliant service while providing a clear and manageable workflow for the development and support teams.

2.0 Current State Analysis

2.1 Identified Problems and Risks

- *No Version Control:* The "Launcher" application source code is maintained on a single developer's machine, representing a critical single point of failure and eliminating change traceability.
- *Insecure Access Control:* A single, legacy Active Directory group grants developers excessive write permissions to all environment files and production data, violating the principle of least privilege.

- *Lack of Separation of Duties:* Developers deploy directly to production (via manual requests) and have direct, unaudited access to production data for development purposes.
- *Uncontrolled Data Flow:* Production data, including Personally Identifiable Information (PII), is manually copied to lower environments without a formal sanitization or masking process, creating a significant data security and compliance risk.
- *Manual & Error-Prone Deployments:* The promotion process for all artifacts is manual (file copies, helpdesk tickets), leading to inconsistencies, a lack of auditability, and a high risk of error.
- *Compliance Gaps:* The current state does not adhere to foundational CIS Controls (e.g., Access Control, Secure Configuration) or OWASP Top 10 principles (e.g., Broken Access Control, Sensitive Data Exposure).

2.2 Current SDLC & Application Flow

The current processes are unstructured and introduce significant risk, as illustrated below.

Current Development & Deployment Flow

```
flowchart TD
    subgraph "Developer Workstation"
      direction TB
      A["Visual Studio: Edit Launcher Code"]
      B["Visual Studio: Edit Excel/Access Artifacts"]
      C["Manual Build of Launcher.exe"]
    end

    subgraph "Single Isilon File Share (All Environments)"
      direction TB
      D["Excel & Access Files (Dev/UAT/Prod)"]
    end

    subgraph "SQL Servers (All Environments)"
      direction TB
      E["Production & Lower Environment Databases"]
    end

    subgraph "Production Endpoints"
      F["Production PCs with Launcher.exe"]
      G["Users accessing Production Artifacts"]
    end

    A --> C
    C -->|1. Helix Ticket| F
    B -->|2. Manual Save/Copy| D
    D --> B
    B -->|3. Direct DB Connection| E
    G --> D

```

Current Application Data Flow (User Perspective)

```
flowchart TB
    subgraph "User's Workstation"
        User["End User"]
        Launcher["Launcher.exe"]
        Artifact["Opened Access/Excel File"]
    end

    subgraph "Isilon File Share"
        FS["Production Artifacts Folder"]
    end

    subgraph "Production SQL Server"
        DB["LoanProcessDB (Production)"]
    end

    User -- "1. Runs" --> Launcher
    Launcher -- "2. Reads Content List From" --> FS
    Launcher -- "3. Opens Selected File" --> Artifact
    Artifact -- "4. Executes Embedded Logic (VBA/VB.NET)" --> DB
    DB -- "5. Returns Production Data" --> Artifact
    Artifact -- "6. Renders UI For" --> User

```

3.0 Proposed Future State Architecture

3.1 Guiding Principles

- *Security First:* Implement the principle of least privilege for all access.
- *Traceability:* Ensure every change is version-controlled and auditable.
- *Automation:* Automate promotions and repetitive tasks to reduce errors and increase velocity.
- *Separation:* Enforce strict separation between production and non-production environments, duties, and data.
- *Atomicity:* Promote application changes as a single, consistent, versioned unit to ensure stability.

3.2 Proposed Solutions

3.2.1. File Share and Permissions Architecture

A structured file share with Role-Based Access Control (RBAC) will be implemented to segregate environments and enforce least privilege.

- *Structure:*

```
\pfil\*\FileShares

├── LoanAutomation-Prod/
├── LoanAutomation-UAT/
├── LoanAutomation-Staging/
├── LoanAutomation-Dev/
└── LoanAutomation-Archive/
```

```
  └── 2025-08-04_143000_Release-1.2.1/
    ├── manifest.json
    ├── Launcher-v1.5.exe
    ├── LoanApplication.xlsm
    └── RiskAnalysis.accdb
```

- *Permissions Model:* New AD groups will be created to enforce separation of duties.
  - *Developers:* `Read/Write` access to `Dev` and `Staging`. `Read-only` to `UAT`. **NO** write access to `Prod` or `Archive`.
  - *Application Support:* `Read/Write` access to all environments, including the `Archive`, to manage promotions. They are the only group with `Write` access to `UAT`, `Prod`, and `Archive`.
  - *Business Users:* `Read-only` access to `Prod` for execution. `Read-only` to `UAT` for testing.

> *Architect's Note on Permissions for MS Access Lock Files:*
> An investigation was conducted into providing developers with a granular permission set that would allow them to create MS Access lock files (`.ldb`, `.laccdb`) in a read-only production directory without granting broader write access.
>
> **Conclusion:** This is not feasible with standard NTFS permissions. The architecturally sound solution is to provide a fully separate `Dev` environment where developers have the necessary permissions to work effectively, eliminating any need for write access to the production directory.

3.2.2. Implementation Strategy: Parallel Environment Cutover

Recognizing that the proposed changes will impact the development team's workflow, a parallel implementation strategy is recommended.

- The new, structured file shares will be created in parallel to the existing legacy share.
- A defined cutover period will be provided for the development team to transition their workflows to the new `Dev` environment.
- The Application Support team will simultaneously build and test promotion automation using the new structure.
- After the cutover period, the legacy file share will be set to read-only and eventually decommissioned.

3.2.3. Version Control Strategy (Source Code)

- *Tool:* Enterprise GitHub.
- *Target:* All text-based source code, primarily the code for the "Launcher" application.
- *Branching Model:* A simple GitFlow model (`main` , `develop` , `feature/*`) is recommended.

> *Clarifying Question:* Does the Launcher application contain any logic that is dependent on a specific version of an Access/Excel artifact?
>
> *Impact if False:* If the Launcher and artifacts are tightly coupled, our release manifest becomes even more critical. We must ensure that a change to the Launcher code cannot be deployed without the correct corresponding artifact version, and vice versa.

3.2.4. Data Management and Sanitization

A formal, automated process for creating safe, non-production datasets is required.

1. *Identify PII:* Conduct data discovery on the production database to classify sensitive data.
2. *Develop Scripts:* Create automated scripts to extract data from production.
3. *Sanitize/Mask:* The scripts will mask, anonymize, or replace all PII according to compliance requirements.
4. *Load into Non-Prod:* The sanitized data will be loaded into the `Dev` , `Staging` , and `UAT` database environments.
5. *Automate:* This entire process will be automated and scheduled to ensure non-prod environments have recent but safe data.

3.2.5. Release Management and Manifest (Binary Artifacts)

To manage the reality that individual Access/Excel artifacts may be promoted independently, a manifest-driven release process will be implemented. This ensures that even individual promotions are done in a controlled, atomic, and traceable manner.

- **The Release Package:** A release is no longer a single file, but a "package" defined by a manifest. This package represents the complete, known-good state of the application for a given version.
- **The Manifest:** This is a simple file (e.g., `manifest.json`) created by Application Support in collaboration with the requesting developer. It explicitly lists every component and its specific version for that release.

```
{
  "releaseId": "2025-08-04_143000_Release-1.2.1",
  "launcherVersion": "1.5",
  "launcherGitTag": "v1.5-release",
  "artifacts": [
    "LoanApplication.xlsm",
    "RiskAnalysis.accdb",
    "Reporting.xlsm"
  ]
}
```

4.0 Assumptions & Required Clarifications

To validate this architecture, the following assumptions must be verified and questions answered by the development team.

4.1 Key Assumptions

- *Launcher Auth:* The Launcher application performs no internal user authentication and relies solely on underlying file-share permissions.
- *Visual Studio Connections:* Developers are using Visual Studio to connect directly to production databases, likely via embedded connection strings in the Access/Excel artifacts.
- *PII Location:* PII is present in both the front-end Access/Excel files and the back-end SQL databases.
- *Tooling Availability:* Enterprise Application Support is willing and able to own production deployments using Ansible Automation Platform and/or MoveIT.

4.2 Critical Questions for the Development Team

1. **Database Connections:**
    - Where are the SQL database connection strings defined today? (e.g., Inside Access VBA, in the Launcher's code, a config file, or a local ODBC DSN?)
    - How are credentials for different environments managed? Are they hard-coded?
    - What authentication method is used (Windows Integrated, SQL Login)?
2. **Launcher Configuration:**
    - How does the Launcher determine which file share path to open? Is this path hard-coded?
    - Can this path be changed without recompiling the application?
3. **Artifact Logic:**
    - Confirm if the Excel and Access files contain significant business logic (VBA macros, forms, queries) or if they are primarily data viewers. A demonstration

of the most critical logic would be beneficial.

Extras:

#	Assumption	Why we think this	How to validate	Owner	Risk if false
A1	The **Launcher** does not perform auth; it relies on Windows/AD file-share permissions.	Behavioral evidence + typical pattern.	Ask dev: “Does Launcher do any user auth/role checks?” Review code/config.	Dev Lead + App Support	We may need to add auth/role checks or wrap with a service if it’s doing more than launching.
A2	The **Launcher** code can be moved to **enterprise GitHub** without legal/IP constraints.	Internal tool; no third-party license issues expected.	Code scan + legal quick check.	Dev Lead + Security	Delay in version control adoption; deployment automation blocked.
A3	**Excel/Access artifacts** contain embedded logic (VBA/forms/queries), not just data.	Devs use VS/VB; artifacts drive workflow.	Open representative files; list of macros/modules; short demo.	Dev Team	More effort to refactor logic into services; promotion becomes more complex.
A4	**Developers need write on the folder** (not the DB content) due to	Standard Access behavior.	Reproduce with read-only file + write	App Support + Dev	Over-permissioning persists;

	Access **lock files** (`.laccdb`).		on folder; confirm error.		audit findings in Prod.
A5	**Visual Studio** connects to **Prod SQL** via connection strings in code or local settings.	Current behavior reports + lack of SDLC.	Inspect connection strings; check SQL login audit logs.	DB Team + Security	Direct Prod access continues; data leakage & change risk.
A6	**PII** resides in front-end files **and** backend databases.	Loan processing domain reality + your note.	Data classification scan (file share + DB).	Data Governance + DB Team	Data masking scope underestimated; non-prod unsafe.
A7	**Enterprise App Support** will own **Prod** deployments and promotions.	Stated willingness + team charter.	RACI confirmation + SOP draft approval.	App Support Manager	No clear ownership → shadow deployments continue.
A8	**AAP** and **MoveIT** are available for automation and controlled transfer.	You confirmed availability.	Access granted, service accounts created, pilot job runs.	App Support + AAP Eng	Manual steps remain; audit trail weak.
A9	We can **create new AD groups** and phase out the **legacy** one with	Common pattern; manageable scope.	AD/SailPoint report of group membership &	IAM + App Support	Hidden dependencies break access;

	minimal dependencies.		usage; migration plan.		project stalls.
A10	Dev team is **open to change** with guided handoffs; they have limited SDLC maturity.	Your assessment.	Run a short enablement session; gauge tool readiness.	App Owner + Architect	Overly ambitious plan fails; adoption risk.
A11	Non-prod environments (Dev/UAT/Staging) are **reachable** and can host sanitized data copies.	Existing estate has these tiers.	Infra check: capacity, network, SQL instances.	Infra + DB Team	No safe place to iterate; pressure to keep using Prod.
A12	No regulatory blocker to storing **sanitized** Prod extracts in non-prod.	Common practice when masked.	Compliance review; sign-off on masking spec.	Compliance + Security	Rework of data pipelines; approval delays.

- Developer team ideally has **no direct write** access to Prod.
- File deployment and management done by Enterprise Application Support via Ansible Automation Platform.

🟨 Non-Production Environments (UAT, Staging)

Group Name (New AD Groups)	Permissions	Members

`LoanAuto_AppSupport_NonProd_RW`	**Read/Write** (Deploy & Promote files)	Enterprise Application Support Team
`LoanAuto_Developers_NonProd_RW`	**Read/Write** (Build/Test files, artifacts)	Developers, QA/Test Teams
`LoanAuto_BusinessUAT_R`	**Read-only** (Acceptance Testing)	Business Users (for testing and validation)

Note:

- Developers perform most of their testing and validation here.
- Enterprise Application Support owns promotions between UAT, Staging, and Prod.

🟦 Development Environment (Dev)

Group Name (New AD Groups)	Permissions	Members
`LoanAuto_Developers_Dev_RW`	**Read/Write** (Development/Test/Modify freely)	Developers, QA/Test teams
`LoanAuto_AppSupport_Dev_RW`	**Read/Write** (Manage/assist developers if needed)	Enterprise Application Support Team

Note:

- This area provides freedom for developers, with sanitized data.
- Regular sync-downs of sanitized data from prod into this area using automation scripts.

🔴 Phasing Out Legacy AD Group

- Identify existing dependencies of the legacy AD group:
    a. Perform AD group audit (via SailPoint, AD reporting, or Splunk queries) to confirm usage.
    b. Identify all fileshares, applications, and infrastructure accessed by the legacy group.

c. Gradually migrate dependencies to new groups (phase-by-phase).

d. After complete transition, retire the legacy AD group to eliminate confusion and risk.

Alignment with Compliance Frameworks

✅ **CIS Controls Alignment:**

CIS Control Area	How Addressed in File-Share Model
Data Protection (Control 3)	Segregated and access-controlled environments; sanitized data for dev environments.
Secure Configuration (Control 4)	Structured RBAC clearly defined.
Account Management (Control 5)	AD groups with defined roles; least privilege enforced.
Audit Log Management (Control 8)	Logs-and-Audit folders for all access.
Controlled Access Based on Need to Know (Control 14)	Dev teams explicitly limited; Prod environment secured.

✅ **OWASP Top 10 Alignment (Developer Perspective):**

Developers can address OWASP best practices more easily with:

- **Security Misconfiguration (A05)**: Clearly defined share permissions.
- **Sensitive Data Exposure (A02)**: PII data sanitized before moving to lower environments.
- **Broken Access Control (A01)**: Implemented strong RBAC policies limiting prod access.

(Ensure developers receive training to understand these requirements clearly.)

Next Steps (Actionable Items):

- **Step 1: Verify details with Development Team**
  - Confirm launcher’s auth approach.
  - Clarify the exact need for write access to Access DB files.

- **Step 2: Initiate Fileshare Changes**
  - Work with Enterprise Infrastructure team to provision structured file shares.
  - Create and populate new AD groups (via SailPoint request).
- **Step 3: Enterprise GitHub**
  - Establish GitHub repository for Launcher app.
  - Assist the developer in transitioning to using GitHub.
- **Step 4: Automation & Ansible Integration**
  - Involve newly certified AAP engineer in planning automation of deployments/promotions.
  - Develop scripts or workflows to sanitize and replicate production data to dev/test environments.
- **Step 5: Legacy Group Phase-out Plan**
  - Audit and transition legacy AD group to new groups.
  - Communicate and execute phased decommissioning.

High-Level Action Plan (Recommendations)

> Phased, outcome-oriented. Each item notes the primary control objectives it advances.

Phase 0 — Guardrails & Visibility (Weeks 0–2)

- **Freeze ad-hoc Prod writes by devs** (temporary exception process only).
  - *CIS v8:* 6 Access Control Mgmt, 5 Account Mgmt; *OWASP:* A01 Broken Access Control, A05 Security Misconfig.
- **Audit the legacy AD group** usage across shares and SQL. Produce a dependency map.
  - *CIS:* 5 Account Mgmt, 8 Audit Log Mgmt; *OWASP:* A09 Security Logging & Monitoring.
- **Locate connection strings** (Launcher, macros, VS projects) and catalog DB targets.
  - *CIS:* 2 Software Inventory, 3 Data Protection; *OWASP:* A02 Cryptographic Failures (secrets handling).

Exit criteria: Inventory of shares, groups, connection strings, and who writes to what in Prod.

Phase 1 — Structure the File Shares & Access (Weeks 2–4)

- **Create environment-segmented shares** and folders you outlined earlier: `Prod/UAT/Staging/Dev`, each with `Launcher-App`, `Excel-Access-Artifacts`, `Logs-and-Audit`, plus `Test-Data` (non-prod).
- **Stand up new AD groups**:
  - `LoanAuto_AppSupport_Prod_RW`, `LoanAuto_Developers_Prod_R` (strict exceptions), `LoanAuto_BusinessUsers_R`,
  - `LoanAuto_AppSupport_NonProd_RW`, `LoanAuto_Developers_NonProd_RW`, `LoanAuto_BusinessUAT_R`,
  - `LoanAuto_Developers_Dev_RW`, `LoanAuto_AppSupport_Dev_RW`.
- **Apply least-privilege ACLs;** devs lose Prod write; App Support owns promotions.
  - *CIS:* 3 Data Protection, 5 Account Mgmt, 6 Access Control, 14 Controlled Access; *OWASP:* A01, A05.

Exit criteria: Devs can fully work in Dev/UAT; App Support is the only team writing to Prod.

Phase 2 — Version Control & Basic SDLC (Weeks 3–6)

- **Move Launcher code to enterprise GitHub**; define a **simple branching model** (e.g., `main`, `release/*`, `feature/*`).
- **Introduce a lightweight SDLC**:
  - Dev → PR → UAT validation → App Support promotion to Prod.
- **Capture build artifacts** (Launcher exe/config, signed if applicable) in a controlled drop location.
  - *CIS:* 2 Software Inventory, 4 Secure Configurations, 8 Audit Logs; *OWASP:* A08 Software & Data Integrity Failures.

Exit criteria: Every Launcher change is traceable from commit → artifact → deployment.

Phase 3 — Data Safety for Lower Environments (Weeks 4–8)

- **Define a data masking/sanitization spec** for PII (names, SSNs, DOB, account #s).
- **Automate sanitized refreshes** into Dev/UAT (frequency based on need).
- **Block direct VS connections to Prod**; provide **non-prod service accounts** and approved connection profiles.
    - *CIS:* 3 Data Protection, 6 Access Control; *OWASP:* A02 Cryptographic Failures, A01 Broken Access Control.

Exit criteria: Dev/UAT have fresh, safe data; no routine dev access to Prod data.

Phase 4 — Deployment Automation & Auditability (Weeks 6–10)

- **Use AAP** for promotions:
    - Jobs to: copy/verify artifacts → checksum → place in `Prod/Launcher-App` → set permissions → write to `Logs-and-Audit`.
- **Use MoveIT** (or AAP modules) for **controlled, auditable file transfer** where required.
- **Centralize logs** (e.g., Splunk): promotions, file writes, and access events by group/user.
    - *CIS:* 8 Audit Log Mgmt, 4 Secure Configurations; *OWASP:* A09 Logging & Monitoring.

Exit criteria: Buttoned-up, auditable deployments; zero manual, untracked Prod writes.

Phase 5 — Hardening & Continuous Improvement (Weeks 8–12)

- **Secret management** for connection strings (vaulted variables, CyberArk, etc.).
- **Macro/Access governance**:
    - Inventory macros/add-ins; sign macros; set policy for what's allowed in Prod.
- **Training & enablement**:
    - Short sessions: SDLC basics, environment usage, how to request promotions, OWASP Top 10 in practice.
- **Retire the legacy AD group** after all dependencies are migrated.
    - *CIS:* 4 Secure Configs, 7 Vulnerability Mgmt (for outdated components), 5/6 Access/Account Mgmt; *OWASP:* A05 Misconfig, A06 Vulnerable Components, A08 Integrity.

Exit criteria: Secrets are vaulted, macros controlled, team trained, legacy group removed.

Recommended RACI (condensed)

- **Architect (you):** Strategy, standards, approvals.
- **Dev Lead:** Launcher code, macros inventory, non-prod readiness.
- **App Support:** Promotions, AAP jobs, share/ACL enforcement, audit trail.
- **DB Team:** Non-prod DBs, masking pipelines, service accounts.
- **IAM/SailPoint:** New AD groups, legacy group retirement.
- **Security/Compliance:** Masking spec approval, logging oversight, CIS/OWASP alignment.

Success Metrics (objective, exec-friendly)

- **0** routine developer write operations to **Prod** shares/databases.
- **100%** of Launcher changes **in Git** with PRs and approvals.
- **100%** of Prod promotions executed by Application Support (audited). (AAP Could be used)
- **≤ X days** to refresh **sanitized** non-prod data per schedule.
- **Legacy AD group** fully decommissioned by **<date>**.
- **Training completion**: Dev & App Support, **>90%** attendance; short assessment ≥ **80%**.

Open Questions to Close Before the Call

- Do we need **code signing** for the Launcher? (recommended if distributed widely)
- Are there any **external vendors** tied to these artifacts or shares?
- Preferred **logging taxonomy** (index, fields) for promotions/access events?
- Is **macro signing** feasible given current Office policies?

If you'd like, I can create a 1–2 slide “Assumptions” page and a 3–4 slide “Action Plan” with CIS/OWASP callouts for your meeting. Want me to draft the PowerPoint outline next?

What does “≤ X days to refresh sanitized non-prod data per schedule” mean?

- **The “≤ X days” part:**
  This sets a maximum target for how quickly your development and test (non-prod) environments should receive **fresh, sanitized copies** of production data.

    - **“X” is a placeholder** for the actual number of days your team agrees is reasonable (for example, 3 days, 7 days, etc.).
- **“Refresh sanitized non-prod data”:**
  This involves taking a current snapshot of production data, removing or masking sensitive information (such as names, SSNs, and account numbers), and then loading the **sanitized copy** into your lower environments (Dev, UAT, QA).
- **“Per schedule”:**
  Refers to the regular cadence you decide—could be weekly, monthly, or aligned with sprint cycles or release milestones.

Why does this metric matter?

- **Keeps test environments useful:**
  Developers and testers work with data that’s close to real, which helps catch bugs before they hit production.
- **Protects sensitive info:**
  Sanitization ensures PII isn’t leaking into places it shouldn’t be (e.g., dev/test, where security is often weaker).
- **Enforces process discipline:**
  Regular, timely refreshes demonstrate that your organization takes data governance seriously, which is critical for audits and compliance.

How do you use this metric in practice?

- **Pick an “X” that’s realistic:**
  Example: “Non-prod environments must be refreshed with sanitized data from Prod every 7 days or less.”
- **Track & report:**
  Each time you refresh, log the date. If you slip past your target (say, it takes 10 days instead of 7), you know you need to improve.
- **Automate if possible:**
  Tools like Ansible Automation Platform or database scripts can make this hands-off.

Example in Context:

Suppose you set “≤ 7 days.”

- **Monday:** Prod data is masked & copied to Dev/UAT.
- **Next refresh deadline:** The following Monday.
- **If you miss that,** you're out of compliance with your own metric.

SDLC Compliance: Relevant Controls (with Links)

CIS Controls v8

- **Control 2: Inventory and Control of Software Assets**
  Version control, inventorying artifacts, and change management ensure you know what's deployed and where.
- **Control 3: Data Protection**
  Sanitizing data in non-prod environments and limiting PII exposure.
- **Control 4: Secure Configuration of Enterprise Assets and Software**
  RBAC, secure file shares, and config management for apps/artifacts.
- **Control 5: Account Management**
  Tight control and monitoring of who has access to what in Dev/UAT/Prod.
- **Control 6: Access Control Management**
  Least privilege: only App Support deploys to prod, devs don't write to prod, etc.
- **Control 8: Audit Log Management**
  Logging promotions, file access, and code changes for traceability and compliance.
- **Control 14: Controlled Access Based on the Need to Know**
  No prod DB access for devs; strong separation of duties.

OWASP Top 10 (2021)

- **A01:2021 – Broken Access Control**
  Strict RBAC, no direct prod access, formal promotion paths.
- **A02:2021 – Cryptographic Failures**
  Protecting credentials and connection strings; sanitizing data.
- **A05:2021 – Security Misconfiguration**
  Correct permissions, properly segmented environments, hardened defaults.
- **A08:2021 – Software and Data Integrity Failures**
  Version control, code reviews, traceable promotion/deployment.

- **A09:2021 – Security Logging and Monitoring Failures**
  Audit trails for deployments, code changes, access to sensitive environments/data.

You can copy/paste this into your companion guide or notes. If you want, I can provide a markdown version with the same links for direct inclusion in documentation.

You said:

can you validate those links?

ChatGPT said:

Sure! I've verified the official resources for both CIS and OWASP. Here's the quick-reference list with properly formatted HTML links and embedded citations for your internal doc:

SDLC Compliance: Relevant Controls (with Verified Links)

CIS Controls v8

- **<a href="** CIS Control 2: Inventory and Control of Software Assets **">Control 2: Inventory and Control of Software Assets</a>**
  Manages version control, artifact tracking, and software inventory to ensure authorized application deployments.
  cas7.1.docs.cisecurity.org+1CIS+1Wikipedia+10CIS+10CIS+10
- **<a href="https://www.cisecurity.org/controls/data-protection-control-3/">Control 3: Data Protection</a>**
  Focuses on data classification, masking, and minimizing PII exposure. CIS
- **<a href="https://www.cisecurity.org/controls/secure-configuration-of-enterprise-assets-and-software-control-4/">Control 4: Secure Configuration of Enterprise Assets and Software</a>**
  Addresses RBAC, secure file shares, and hardened default configurations.
  OWASP+10CIS+10CIS+10
- **<a href="https://www.cisecurity.org/controls/account-management-control-5/">Control 5: Account Management</a>**
  Ensures proper provisioning and de-provisioning of access rights across environments.
  OWASP
- **<a href="https://www.cisecurity.org/controls/access-control-management-control-6/">Control 6: Access Control Management</a>**

  Implements least-privilege access and group-based permissions aligned with roles. OWASP+14CIS+14cas7.1.docs.cisecurity.org+14

- **<a href="https://www.cisecurity.org/controls/audit-log-management-control-8/">Control 8: Audit Log Management</a>**
  Enables traceability of deployments, file access, and policy violations. CISCIS
- **<a href="https://www.cisecurity.org/controls/controlled-access-based-on-the-need-to-know-control-14/">Control 14: Controlled Access Based on the Need to Know</a>**
  Restricts exposure of sensitive environments and data to only authorized roles. cas7.1.docs.cisecurity.org+2cas7.1.docs.cisecurity.org+2cas7.1.docs.cisecurity.org+2OWASP Cheat Sheet Series+1OWASP+1

OWASP Top 10 (2021)

- **<a href="** A01 Broken Access Control - OWASP Top 10:2021 **">A01:2021 – Broken Access Control</a>**
  Mitigates risk by enforcing strict RBAC and denying direct dev access to Prod. Wikipedia+9OWASP+9OWASP+9OWASP+3OWASP+3OWASP+3
- **<a href="** A02 Cryptographic Failures - OWASP Top 10:2021 **">A02:2021 – Cryptographic Failures</a>**
  Focuses on secure handling of credentials, secrets, and sanitized data. OWASP+9OWASP+9OWASP+9
- **<a href="** A05 Security Misconfiguration - OWASP Top 10:2021 **">A05:2021 – Security Misconfiguration</a>**
  Prevents unsecured ACLs, misconfigured shares, and environment mixing. OWASP Cheat Sheet Series+3OWASP+3OWASP+3
- **<a href="** A08 Software and Data Integrity Failures - OWASP Top 10:2021 **">A08:2021 – Software and Data Integrity Failures</a>**
  Adds software integrity through Git, code reviews, and promotion pipelines. OWASP Cheat Sheet Series+3OWASP+3OWASP+3
- **<a href="** A09 Security Logging and Monitoring Failures - OWASP Top 10:2021 **">A09:2021 – Security Logging and Monitoring Failures</a>**

Ensures robust audit logging for deployments, access, and configuration changes.
[Wikipedia+3OWASP+3OWASP+3]

Clarifications & Technical Explanations:

1. Launcher Application

- **What it seems to do:** Opens a centralized menu or interface for launching various Excel spreadsheets and Access databases.
- **Auth/permissions:**
  - It likely relies on users' existing Windows permissions for files. The launcher probably does **not** handle file permissions directly.
  - Confirm this explicitly with the developer:
    "Does the Launcher handle any user authentication or authorization internally, or does it simply rely on users' existing Windows file share permissions?"
- **Deployment:**
  - Placing the launcher on enterprise GitHub is a solid and necessary first step to introducing version control.

---

2. Access & Excel Front-End (Logic vs. Data Container)

This differentiation is critical:

Type	What it means	Questions to clarify with devs
Logic Container	Files contain VBA macros, custom scripts, or business logic embedded directly within them. They execute business logic (calculations, queries, forms) directly inside the Excel or Access application itself.	"Do these Excel and Access files contain embedded VBA macros, queries, forms, or other logic that runs locally? Or are they purely for data storage and retrieval?"
Data Container Only	Files are simply containers to store or retrieve data (usually temporarily). They have	"Are the Excel and Access files mostly used to store or retrieve

	minimal logic and mainly point to external databases.	data without local processing or embedded logic?"

You **likely have "Logic Containers"** here, given they're using Visual Studio (Apache2 Ubuntu Default Page: It works). Apache2 Ubuntu Default Page: It works code could mean external applications, but they might also be writing VBA inside Access/Excel. Clarify explicitly.

Clarify with the team:

- *"Is Visual Studio used only to build the launcher application, or are you using* Apache2 Ubuntu Default Page: It works *to automate or embed logic directly in the Access/Excel files?"*

---

3. Access Write Permissions Misunderstanding

- Generally, Access databases (`.mdb` or `.accdb`) only require **read permissions** if just viewing data.
- The confusion probably arises from the fact that:
  - Access creates temporary **lock files (** `.ldb` **/** `.laccdb` **)** when opening databases.
  - If the databases are opened directly on the share, the user needs permission to **create/write** these lock files on the folder itself.
  - It's not "write permissions" to the database **content** itself that's needed—just the ability to create and manage temporary lock files.

To clarify explicitly:
"Do you mean you need folder-level write permissions just to open Access files, or are you actually modifying (writing to) data in these Access databases from the frontend?"

A better approach is generally:

- Move copies of databases locally or to dedicated folders for temporary access.
- Implement proper file share permissions separating folder vs. file-level access.

---

4. Visual Studio and DB Connection

- Visual Studio (VS) is typically used to write code that connects programmatically to databases via connection strings.

- Developers likely have connection strings configured in their 🔗 Apache2 Ubuntu Default Page: It works code to connect to SQL servers.
- Currently, they're directly hitting production databases. This is risky and not best practice.

Typical DB Auth Setup (recommended future state):

- **Launcher app or** 🔗 Apache2 Ubuntu Default Page: It works **App:**
  - App connects to databases via a **service account**.
  - Users authenticate to the app, not directly to the DB.
  - Connection strings are stored securely in config files or secrets management solutions (e.g., CyberArk, Azure Key Vault).
- **Developers in Visual Studio (Dev Environment):**
  - Use individual or shared credentials specifically provisioned for dev environments only.
  - Ideally, move entirely away from direct production DB connections. Instead, use sanitized databases for development/testing.

---

5. Data Management & PII Considerations

Since PII exists in your databases, you must establish **data masking** or **data sanitization** when promoting or replicating data from prod to non-prod environments:

- Scrub PII data (names, SSNs, account numbers, etc.).
- Create sanitized datasets for lower environments to comply with NCUA and general security best practices.

---

6. Enterprise Application Support & Tooling

- Great news that **Enterprise Application Support** can help handle the production deployment.
- Using your newly trained Ansible Automation Platform (AAP) resource would be highly beneficial:
  - AAP can automate file movement and deployments.
  - MoveIT can be leveraged for secure, auditable file transfers.

---

7. Governance Standards (Unanswered Question)

You didn't explicitly address governance. Typically, credit unions have specific internal IT standards or compliance standards beyond NCUA (e.g., CIS controls, internal audit compliance, Sarbanes-Oxley controls).

- Verify if such internal standards or audits apply. Ensure alignment in your action plan.

Structured Problem Space (Summary):

Current Issues

- Lack of defined SDLC.
- No version control for critical software.
- Insecure/unstructured production file access.
- Uncontrolled prod data in non-prod environments.
- Developers directly accessing production databases.

Future State Goals

- Established, simple SDLC with clear separation of duties.
- Version-controlled Launcher app on GitHub.
- Secure fileshare structures with clear access management.
- Sanitized, safe datasets for development environments.
- Removal of direct developer prod DB access, replaced with controlled service accounts.
- Enterprise App Support managing deployments via automation (Ansible).

Next Steps & Actionable Questions for Developers:

Use the questions below to clarify technical details explicitly:

- **Launcher:**
  a. Does the launcher handle any authentication or authorization itself?
  b. Does the launcher contain hard-coded connection strings or configurations?
- **Excel/Access Files:**
  a. Are Excel/Access files just data containers, or do they have embedded logic (macros/forms)?
  b. What exactly happens when a user opens an Access DB—do lock files appear in the folder?

   c. Why specifically do you claim you need write permissions—can you explain/show the exact error or limitation?
- **Visual Studio & DB:**
   a. Does your Apache2 Ubuntu Default Page: It works code directly query/update production databases?
   b. Can we use non-prod databases with masked data to perform your day-to-day tasks?
- **Data Handling:**
   a. What sensitive data (PII) resides in your files, and how do you currently handle this data moving down environments?
- **Governance:**
   a. What compliance or audit standards does your team currently need to follow? (NCUA, internal policies, etc.)

Immediate Next Action:

- **Discuss these specific clarifying questions** directly with the dev team to get definitive answers.
- **After you confirm these details**, I'll help you create the structured recommendation document, including a phased action plan.

Let me know if any additional clarifications are needed!

Simplified Slide Deck

Slide 1: Title & Kickoff

Loan Process Automation: Dev Team Alignment
Date | Your Name/Role

> *"Moving from ad-hoc to governed SDLC"*

Slide 2: Why We’re Here

- Align on what we know & clarify what we don’t
- Set immediate priorities for Launcher, Artifacts, DBs
- Agree on next steps toward SDLC & compliance

Slide 3: Where We Stand Now

- **Launcher:** Single dev, manual builds, no Git
- **Artifacts:** Excel/Access in one share, devs copy/play in Prod
- **DBs:** Direct Prod connections, no data sanitization

> *(Add a small visual/flowchart if you have time)*

Slide 4: What’s Broken

- No version control → untraceable changes
- Over-privileged access → security & audit risks
- PII in non-prod → compliance gaps
- Manual, error-prone deployments

Slide 5: Where We’re Going

- Version control (Git) for Launcher
- RBAC with new AD groups and environment splits
- Sanitized, safe data in non-prod
- Automated, auditable deployments

Slide 6: What Changes for You

- Dev work (incl. Visual Studio) moves to Dev/UAT, not Prod
- New AD groups = permissions based on environment
- Fresh, sanitized test data in lower environments
- Code and artifacts go through App Support for Prod
- All changes are tracked, reviewable, and auditable

Slide 7: File-Share & Permissions Preview

Environment	App Support	Developers	Business Users
Prod	RW	R (strict)	R
Non-Prod	RW	RW	R
Dev	RW	RW	–

> *(See guide for details; just highlight separation here.)*

Slide 8: Key Action Steps

- **0–2 wks:** Audit shares, code, connection strings, AD groups
- **2–4 wks:** Stand up new shares/permissions, restrict Prod
- **3–6 wks:** Move Launcher to Git, define SDLC model
- **4–8 wks:** Automate data refresh/masking in non-prod
- **6–10 wks:** App Support pipelines, secrets management, training

---

Slide 9: Next Steps & Q&A

- Confirm assumptions and open questions
- Schedule deep dive on connection strings, share mapping
- Share companion guide for details
- Q&A

That’s it—9 slides, every section actionable.
If you want, you can combine Slides 4 and 5 (“What’s Broken/Where We’re Going”) into a “Challenges & Goals” slide to make it even shorter.
Let me know if you want it in outline, PowerPoint format, or with icon ideas!