# Back-2-Basics Blog Series – **Part 2:**

# 15 More Security Principles App Owners & Admins Often Forget

**Summary Table of Contents**



## Introduction

In **Part 1** of this Back-2-Basics series, we covered foundational security principles that every organization should be applying—but often isn't. The reality is, most security gaps don't come from a lack of tools or frameworks. *They come from overlooking the basics.*

**Part 2** builds on that foundation with 15 additional practices that are just as critical—but possibly even more frequently neglected in day-to-day operations. These aren't advanced or theoretical concepts. They're practical, real-world disciplines that directly impact how well your organization can prevent, detect, and recover from threats.

From data minimization and secure design to controlling automation, managing secrets, and validating your controls, these principles focus on one thing: turning security from a checklist into something that actually works in practice.

If Part 1 was about *what good security looks like*, Part 2 is about *how it often breaks—and how to fix it*.

## 1. Data Minimization & Lifecycle Management

This is about controlling **how much data you have, how long you keep it, and where it lives**.

**What it involves:**

- Collect only what's necessary (no "just in case" data)
- Define retention policies (e.g., delete after X days/months/years)
- Automatically archive or securely delete old data
- Apply different controls based on data sensitivity

**Where teams fail:**

- Keeping data indefinitely
- Forgetting about backups, logs, exports, and test datasets
- Not deleting data after customer offboarding

**Best practice:**

- Tie data retention directly to legal, regulatory, and business needs
- Automate deletion and anonymization
- Regularly audit for "data sprawl"

## 2. Secure by Design, Not After the Fact

Security must be built into architecture—not bolted on later.

**What it involves:**

- Threat modeling during design phases
- Secure architecture reviews before development
- Embedding security requirements into user stories and pipelines

**Where teams fail:**

- Treating security as a final QA step
- Retrofitting controls after deployment (expensive and incomplete)

**Best practice:**

- Integrate security into the SDLC (DevSecOps)
- Use secure design patterns and reference architectures
- Make security a requirement, not an enhancement

## 3. Human-Centered Security

Security controls must work *with* people, not against them.

**What it involves:**

- Designing usable authentication (e.g., MFA that isn't overly painful)
- Reducing friction that leads to workarounds
- Training based on real-world threats (phishing, social engineering)

**Where teams fail:**

- Overly complex policies (users bypass them)
- One-time annual training with no reinforcement

**Best practice:**

- Design for usability and behavior, not just policy
- Use phishing simulations and continuous awareness
- Minimize reliance on human memory or manual steps

## 4. Verify, Don't Just Trust Controls

Controls must be **tested and validated regularly**.

**What it involves:**

- Testing backups via real restores
- Validating alerting and monitoring systems
- Running tabletop exercises and simulations
- Performing access reviews

**Where teams fail:**

- Assuming controls work because they exist
- Never testing incident response or recovery

**Best practice:**

- Continuously validate controls in real conditions
- Treat testing as part of operations, not a one-time effort

## 5. Resilience Over Prevention

Prevention is important—but failure is inevitable.

**What it involves:**

- Designing systems to:
    - detect quickly
    - contain damage
    - recover fast
- Defining and meeting:
    - RTO (Recovery Time Objective)
    - RPO (Recovery Point Objective)

**Where teams fail:**

- Over-investing in prevention, under-investing in recovery
- No clear recovery strategy during incidents

**Best practice:**

- Assume compromise will happen
- Focus equally on detection, response, and recovery
- Build systems that degrade gracefully, not catastrophically

## 6. Secrets Management Discipline

Credentials are one of the most exploited weaknesses.

**What it involves:**

- Centralized secrets management (vaults)
- No hardcoded credentials in code or configs
- Regular rotation of secrets
- Tight access control and auditing

**Where teams fail:**

- Secrets stored in:
    - source code
    - environment files
    - scripts
- Rarely rotated credentials

**Best practice:**

- Use dynamic or short-lived secrets where possible
- Automate rotation and revocation
- Monitor usage for anomalies

## 7. Context-Aware Access (Adaptive Security)

Access decisions should consider **context, not just identity**.

**What it involves:**

- Evaluating:
  - device health
  - location
  - time of access
  - behavioral patterns
- Adjusting controls dynamically (step-up authentication, blocking, etc.)

**Where teams fail:**

- Binary access decisions (allowed/denied only)
- No adaptation to risk signals

**Best practice:**

- Implement risk-based authentication
- Trigger stronger controls when anomalies are detected
- Combine with Zero Trust principles

## 8. Third-Party & Supply Chain Awareness

Vendors and dependencies are part of your attack surface.

**What it involves:**

- Assessing vendor security posture
- Limiting vendor access to only what's necessary
- Monitoring third-party integrations
- Managing software dependencies and updates

**Where teams fail:**

- Trusting vendors without verification
- Excessive permissions for integrations
- Ignoring supply chain risks (libraries, packages, plugins)

**Best practice:**

- Maintain an inventory of third parties and dependencies
- Require security standards and reviews
- Continuously monitor—not just at onboarding

## 9. Logging for Detection, Not Just Compliance

Logging should help you **detect and respond**, not just "check a box."

**What it involves:**

- Capturing meaningful, high-value events:
  - authentication attempts
  - privilege changes
  - data access

- Centralizing logs for correlation
- Building actionable alerts

**Where teams fail:**

- Logging too much noise or too little signal
- Logs are stored but never reviewed
- Alerts that no one responds to

**Best practice:**

- Focus on **useful, actionable telemetry**
- Tune alerts to reduce noise
- Regularly review and improve detection logic

## 10. Non-Named Accounts (Service, Shared, Automation) Should Be Minimized, Tightly Controlled, and Heavily Monitored

Non-named accounts are high-risk because they often lack strong authentication, have excessive privileges, and limited accountability—making them a common attack path.

**What this involves:**

- Minimize use; prefer named users or managed/ephemeral identities
- Assign a clear owner and defined purpose for every account
- Enforce least privilege and restrict scope (systems, environments)
- Store credentials securely (no hardcoding) and rotate regularly
- Log and monitor all activity

**Common failure:**

- Too many service/shared accounts with no clear ownership
- Long-lived or hardcoded credentials
- Overprivileged access “for convenience”
- Little to no monitoring or review
- Dormant accounts left active indefinitely

**Best practice:**

- Replace with short-lived (ephemeral) credentials wherever possible
- Require ownership, justification, and periodic review
- Enforce strict access controls and secure secrets management
- Monitor closely and alert on abnormal behavior
- Automatically disable unused accounts

## 11. Review Backup Logs and Test Backup Restoration

Backups are useless if you can’t **reliably restore** from them.

**What this involves:**

- Regularly reviewing backup job logs for:
    - failures
    - partial backups
    - skipped files/systems
- Performing **scheduled restore tests** (not just checking that backups exist)
- Verifying:
    - data integrity
    - recovery time (RTO)

    - recovery point (RPO)

**Common failure:**

Teams assume backups work because jobs say "successful"—but:

- data is corrupted
- restores are incomplete
- recovery takes too long during an incident

**Best practice:**

Treat restore testing like a fire drill:

- automate where possible
- test critical systems more frequently
- document and improve recovery procedures

## 12. Identify and classify data for compliance needs (Ex: PII & PHI privacy data identified for GDPR and HIPAA)

You can't protect data properly if you don't know what it is.

**What this involves:**

- Discovering and labeling sensitive data such as:
    - PII (Personally Identifiable Information)
    - PHI (Protected Health Information)
- Mapping where data lives:
    - databases
    - file storage
    - SaaS platforms
- Aligning controls to regulatory requirements like:
    - General Data Protection Regulation (GDPR)
    - Health Insurance Portability and Accountability Act (HIPAA)

**Common failure:**

- "We encrypt everything" → but don't know where sensitive data actually resides
- Shadow data in exports, logs, backups

**Best practice:**

- Use classification tiers (e.g., Public / Internal / Confidential / Restricted)
- Apply controls based on classification (encryption, access, retention)
- Continuously scan for sensitive data sprawl

## 13. OS and Application Hardening Standards

Out-of-the-box configurations are rarely secure.

**What this involves:**

- Defining baseline configurations for:
    - operating systems
    - databases
    - applications
- Removing or disabling:
    - unnecessary services
    - default accounts
    - unused ports and protocols

- Enforcing:
    - patch levels
    - secure configurations
    - configuration drift detection

**Common failure:**

- "Golden images" created once and never updated
- Inconsistent configurations across environments

**Best practice:**

- Use established benchmarks (like CIS benchmarks)
- Implement **configuration as code**
- Continuously monitor and remediate drift

## 14. Restrict Admin privileges

Admin access is one of the most exploited attack paths.

**What this involves:**

- Limiting who has administrative rights
- Separating:
    - standard user accounts
    - privileged/admin accounts
- Using:
    - Just-in-Time (JIT) elevation
    - approval workflows
    - session monitoring

**Common failure:**

- Users with permanent admin rights "for convenience"
- Admin accounts used for email/web browsing

**Best practice:**

- Enforce **least privilege strictly**
- Require MFA for all privileged access
- Log and monitor all admin activity

Regularly review and remove unnecessary privileges

## 15. Restrict RPA, macros and other automated services such as data integration

Automation is powerful—*but also a major hidden attack vector.*

**What this includes:**

- Robotic Process Automation (RPA) bots
- Office macros (e.g., Excel, Word)
- Data integration tools and scripts
- Scheduled jobs and workflows

**Risks:**

- Often run with **high privileges**
- May use **hardcoded credentials**
- Can execute malicious actions at scale
- Frequently bypass normal user controls

**Common failure:**

- "It's just automation" → treated as low risk
- No visibility into what bots/scripts actually do

**Best practice:**

- Treat automation like **service accounts**:
    - assign least privilege
    - avoid shared credentials
    - use secure secret storage
- Disable macros by default; allow only signed/approved ones
- Monitor behavior and execution logs

Restrict where automation can run and what it can access

## Conclusion

Security failures rarely come from missing tools or frameworks—they come from inconsistent execution of fundamentals. Across these 15 principles, a clear pattern emerges: strong security depends on how well organizations manage data, control access, validate assumptions, and operationalize discipline over time.

Whether it's minimizing data exposure, hardening systems by default, securing automation, or ensuring backups actually work when needed, each principle reinforces the same idea: security is only real when it is continuously applied, tested, and enforced in practice—not just defined in policy.

By consistently returning to these basics, app owners and administrators can significantly reduce risk, improve resilience, and build systems that are prepared not just to prevent issues—but to withstand and recover from them when they occur.

## 15 Best Practices - Reference Summary:

**1. Data Minimization & Lifecycle Management** 🗂️🔒 - Only collect and keep data that is truly needed, and securely delete it when it is no longer required.

**2. Secure by Design, Not After the Fact** 📐🛡️ - Build security into systems from the start instead of trying to fix it after deployment.

**3. Human-Centered Security** 👤🔐 - Design security controls that are easy for people to use correctly without creating workarounds.

**4. Verify, Don't Just Trust Controls** 🔍✅ - Regularly test and validate that security controls actually work as intended.

**5. Resilience Over Prevention** 🧱🔄 - Focus on detecting, containing, and recovering from incidents, not just trying to prevent them.

**6. Secrets Management Discipline** 🔑🧰 - Store and manage credentials securely, and avoid hardcoding or exposing sensitive keys.

**7. Context-Aware Access (Adaptive Security)** 🌐🧠 - Adjust access decisions based on risk signals like device, location, and behavior.

**8. Third-Party & Supply Chain Awareness** 🔗🚚 - Manage and monitor vendor and dependency risks as part of your security posture.

**9. Logging for Detection, Not Just Compliance** 📜🚨 - Collect and use logs to actively detect threats, not just to meet audit requirements.

**10. Non-Named Accounts Controlled & Monitored** 👥👁️ - Minimize shared and service accounts, and tightly control and monitor their use.

**11. Review Backup Logs & Test Restoration** 💾🔁 - Regularly check backup success and test restores to ensure data can actually be recovered.

**12. Data Classification & Compliance** 🏷️📊 - Identify and label sensitive data so appropriate privacy and compliance controls can be applied.

**13. OS & Application Hardening Standards** 🖥️🛡️ - Secure systems by removing unnecessary features and enforcing secure configurations.

**14. Restrict Admin Privileges** 👷🚫 - Limit administrative access and use it only when necessary under strict controls.

**15. Restrict RPA, Macros & Automation Services** 🤖⛔ - Control automation tools tightly to prevent misuse or hidden malicious activity.

---