PH7X SYSTEMS ®

0.1.0 · CHAPTER 17

Farm solutions that run with elevated privileges

SharePoint Migration Field Guide · Chapter 17

pH7x Systems

2026-11-01

17. Farm solutions that run with elevated privileges

> Elevation decides which account acts, never who may ask.

Most farm solution code crosses to Subscription Edition like any other component. Code that acts with more rights than whoever starts it needs its own decision, because an upgrade checks that it is present, not what it lets people do.

> **Which operations in your farm solutions act with more rights than whoever starts them, and which of them can cross to Subscription Edition as they are?**

The decision this chapter leads to is a disposition for each of those operations (keep, adapt, replace or eliminate) and the six answers that justify it.

Which code this chapter is for

Farm solutions on a SharePoint Server 2016 or 2019 farm moving to Subscription Edition: extensions whose "custom logic calls the SharePoint server object model and runs with full trust", supported "in SharePoint on-premises only" (development models, checked on 1 November 2026).

An operation, here, is one elevated call reached from one entry point. Elevated means the `SPSecurity` method `RunWithElevatedPrivileges`, which runs a delegate "in the context of an account with higher privileges than the current user" (elevation of privilege, checked on 1 November 2026), or a site opened with another account's user token. Code that does not elevate belongs to Chapter 15.

Find every elevated operation

With the source code, you search it for the four names in the script below. Without it, the assemblies still name every method they call: a package "has a .cab-based format but a .wsp file extension" (build farm solutions), `expand` extracts it, and the IL Disassembler writes each assembly out as text (all three checked on 1 November 2026). From a Visual Studio Developer PowerShell:

```
# Run in an empty folder holding a copy of one package. Changes nothing on any farm.
Copy-Item .\Requests.wsp .\package.cab
Write-Host "Extracting Requests.wsp and disassembling its assemblies"
expand .\package.cab -f:* .
foreach ($d in Get-ChildItem -Filter *.dll -Recurse) {
    ildasm $d.FullName /out="$($d.FullName).il" }
Get-ChildItem -Filter *.il -Recurse |
    Select-String 'RunWithElevatedPrivileges|SPUserToken|AllowUnsafeUpdates|ValidateFormDigest'
```

Each match names an `.il` file and a line, and the calling method is the nearest `.method` above it. You trace each method to the entry points that reach it and record one operation per entry point.

Six answers for each operation

Each operation gets six answers, the columns of the migration pack's elevated operations sheet:

ANSWER	WHAT IT ESTABLISHES, AND WHERE IT COMES FROM	AN ANSWER THAT STOPS IT CROSSING AS IT IS
1. Caller	The entry points, and who can reach or schedule each one	Anyone who can open a page, with nothing narrower
2. Permission	The specific right on the specific object that the caller lacks	Full Control of the site collection, because it was simpler
3. Identity	The account acting on each resource, in SharePoint and outside it, as set out below	An account, or a resource, that nobody listed
4. Authorisation	How the code decides this caller may ask: a check before the elevated block, against data the caller cannot supply	No check, or a check of a value that arrives with the request
5. Form digest	`SPUtility.ValidateFormDigest()` or `SPWeb.ValidateFormDigest()`, called before any elevated write	A write on a GET request with `AllowUnsafeUpdates` set and no digest
6. Record	What the solution writes, and what Modified By, version history and the audit log show on the target build	Nothing that names the person who asked

Microsoft's current guidance is that elevation "should be used with care. You should not expose direct, uncontrolled mechanisms for people with low privileges to circumvent the permissions granted to them", with a form digest check before any elevated write (elevation of privilege, checked on 1 November 2026). Microsoft's 2010 developer guidance (checked on 1 November 2026) adds that elevated operations "always succeed regardless of the user's permissions", so the code checks with `DoesUserHavePermissions` or `CheckPermissions`; that elevated objects "must not be returned outside" the block; and that `AllowUnsafeUpdates` "must never be used to work around a canary validation error". It was written for 2010, so it says what to test, not what the target build does. Questions 3 and 6 need the rehearsal farm: the account depends on the process, the record on the build.

Whose rights the elevated code uses

The same 2010 guidance says an elevated block runs "by using service account credentials", and that a site opened inside it "notes that the current user is the System Account". We would read that service account as the account of the process the code runs in:

WHERE THE ELEVATED CODE RUNS	THE ACCOUNT OF THAT PROCESS	WHAT MICROSOFT'S ACCOUNT GUIDANCE GIVES IT
Pages, web parts and web services of a content web application	The web application pool account	`SP_DATA_ACCESS` on that web application's content and search databases
Timer jobs and Central Administration	The farm service account	`dbcreator` and `securityadmin` on SQL Server, `db_owner` on every SharePoint database
A PowerShell session or a console program	The administrator running it	Whatever that administrator's account holds

Sources: account permissions and security settings and administrative and service accounts, checked on 1 November 2026. The rehearsal farm confirms the middle column when the delegate logs `WindowsIdentity.GetCurrent().Name` (GetCurrent, checked on 1 November 2026).

The second row changes conclusions: Microsoft says to "treat the securityadmin role as equal to the sysadmin role" (least-privileged administration, checked on 1 November 2026). A helper called from a web part and from a timer job borrows two very different accounts with no difference in its code. Event and feature receivers run, in our experience, in the process that made the change: a row for each.

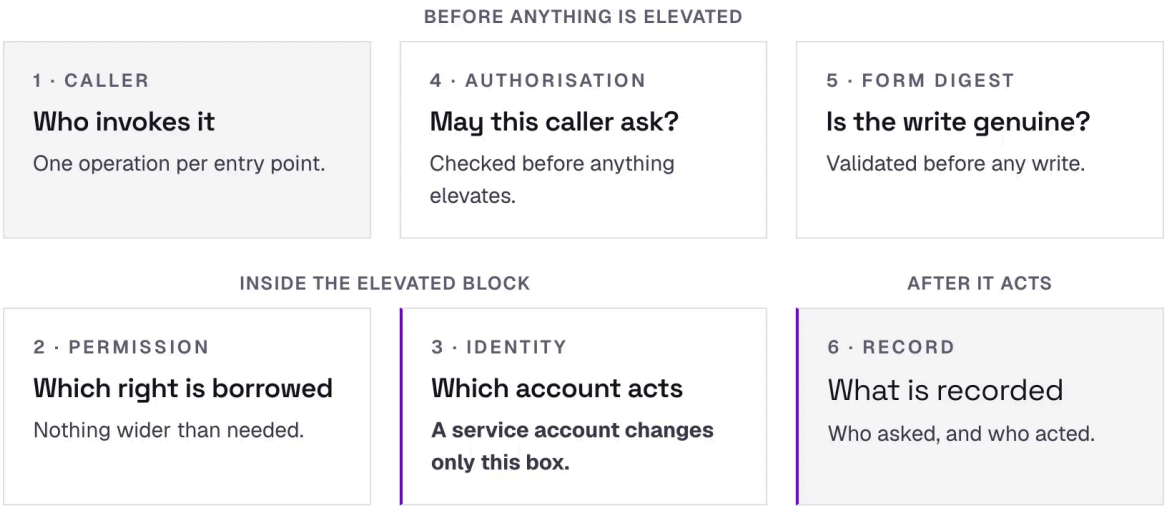


Figure 17.1. Six answers along one request. A service account in place of the elevated call changes one of them.

Keep, adapt, replace or eliminate

Keep an operation when all six answers already hold, and test it like any kept component.

Adapt it when it is needed and an answer fails: check the caller before the block, validate the digest, record the requester, open the site inside the block and let nothing elevated leave it, and elevate only the lines that need it.

Replace it when it can run outside the farm, through the client APIs, with an identity of its own. On premises, app-only policy through server-to-server trust "only works in on-premises SharePoint environments", and does not reach search on premises, most User Profile operations through CSOM or taxonomy writes, which need "a specific service account" (app-only and elevated privileges, checked on 1 November 2026).

Eliminate it when nothing depends on it any more, or when the people it serves should hold the permission it borrows. The second is the site owner's decision, recorded as one.

Swapping the elevated call for a service account is none of these. It changes question 3, leaves the other five as they were, and can make question 6 worse: "the Modified By column for the list items will display the name of the service account" (same source), so the person who asked disappears unless the solution records them. We recommend choosing the identity last: the narrowest account that holds exactly the right question 2 names.

Prove each kept operation on the rehearsal farm

For each operation kept, adapted or replaced, run four tests and keep their evidence beside its row: a caller who may ask succeeds and is recorded; one who may not is refused before anything elevated runs; a write without a valid form digest is refused; and the account it runs as, with what Modified By, version history and the audit log show, is written down. A test that only proves success proves that elevation works, which was never in doubt.

Out of scope

Other customisations are Chapter 15; client-side code, add-ins, authentication and integrations are Chapters 16, 18, 21 and 22; installing what you keep is Chapter 23. This is not a secure coding course or a penetration test. The reasoning behind the six answers is in our analysis of privileged server-side code.

Before you continue

- [ ] Every elevated call, by delegate or by token, is listed once per entry point
- [ ] Each operation names who can invoke it and the permission they lack
- [ ] Each operation names the account acting on each resource, confirmed on the rehearsal farm
- [ ] Each operation checks the caller before elevating, against data the caller cannot supply
- [ ] Every elevated write validates the form digest
- [ ] Who asked is recorded, and Modified By and the audit log were read on the target build
- [ ] Each operation has a disposition and a reason, and its identity was chosen last