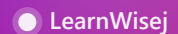


The Wisej.NET Security Model

Lesson Guide · Lab / Exam Guide



Lesson Guide

What the browser can and cannot see in a server-side UI, the session boundary, the threat model, and where each control lives. What this means for a developer securing a production Wisej.NET app — and how to read this module.



Production focus

What the browser can and cannot see in a server-side UI, the session boundary, the threat model, and where each control lives.

1. The production problem

AccessOps is an internal operations portal: departments, employees, confidential documents, approval requests and an admin console. Everyone who can reach it already works for the company, which is exactly why the security work gets postponed — and why one portal ends up holding salary bands, disciplinary letters and the approval trail for every exception the business ever granted. Before a line of authentication code exists you need three answers: what the browser is holding, where the boundaries run, and which of the things that could go wrong are worth engineering against first.

2. Key concepts

Concept	Why it matters
server-side UI model	Controls are C# objects in the server process; the browser holds a rendered mirror and an event channel, so your code and queries are never downloaded.
widget state payload	The property values Wisej.NET serialises so the browser can draw a control. Whatever you assign is on the wire, including on controls the user cannot see.
trust boundary	A line where data passes between parties of different trust. Anything crossing one is re-checked by the more trusted side, because the other side can lie.
session boundary	The frontier between one browser and its own server state in <code>Application.Session</code> . Decisions come from state on the server side of it, never from a browser value.
entry point	Any route in that an attacker can reach: the login page, event handlers, uploads, downloads, custom HTTP handlers, interop calls and the AI assistant's prompt.
STRIDE	Six categories for naming threats — spoofing, tampering, repudiation, information disclosure, denial of service, elevation of privilege — so a review is systematic.
control placement	Choosing the boundary at which a rule can be enforced. A rule that exists only in the UI is a suggestion; the same rule inside a service is a control.

3. Lesson narrative

What the browser actually receives

A Wisej.NET application is an ordinary .NET program. The controls you place on a window are C# objects from the `Wisej.Web` namespace living in the server process, and the browser holds a rendered mirror of them plus a channel for events. A user clicks; the browser reports the click; the server raises the handler in the session that owns the widget; your code runs next to the database; the changed properties are serialised back. No assembly, LINQ query, validation rule or connection string is ever shipped to the client.

What does cross the wire is every property value you assign, because the browser cannot draw what it has not been told: the text of a label, the items of a `ComboBox`, the cells of every row you loaded into a `DataGridView`, a tooltip, a `Tag` parked on a tree node for convenience. Setting `Visible` to false on a column changes what is drawn, not what was sent. The same goes for what you deliberately push out of the process: files streamed with `Application.Download`, files accepted by an `Upload` control, custom HTTP handlers you register, and any JavaScript you ask the client to run with its arguments.

The first AccessOps rule follows: a query decides what leaves the server, not a designer property. The directory grid takes a projection of the columns every viewer may see; the salary band is a second, checked call.

```
// ✗ the whole record crosses the wire
gridEmployees.DataSource = _repo.GetAllEmployees();
gridEmployees.Columns[5].Visible = false;    // Salary

// ✓ the server decides what leaves the process
gridEmployees.DataSource =
    _employees.GetDirectory(_session.CurrentUserId);
```

Four boundaries, one application

The first boundary sits between the browser and the Wisej.NET session. Everything arriving across it is a claim: which widget was clicked, which grid row is current, what was typed, which document id a link mentions. A claim is not a fact, and the session is the first place where a fact can exist.

The second separates one session from the process that hosts all of them. Sessions share memory, so a static field, a singleton service or a cache without a per-user key is visible to everyone signed in to that server. Verified identity therefore belongs in `Application.Session` and never in a static "current user". Background work started with `Application.StartTask` outlives the request that launched it, so it must carry the identity it was given rather than reaching back for whichever session is current when it finishes.

The third and fourth face outward, to the database and to external services: the identity provider you add in Module 4, the document store, the mail relay. Give the database login only the rights `AccessOps` needs and parameterise every command, so a department name typed into a search box cannot become SQL. Name an owner for each boundary, because a control that sits on none of them has nothing to stop.

A threat model for `AccessOps`

A threat model is three lists and a table. The assets first: employee records, the document store, approval decisions, the audit trail that proves them, and the credentials protecting all of it. Then the entry points: the login page, every event handler, the `Upload` control, every download, any custom HTTP handler, and the AI assistant, whose prompt is untrusted input like any other. Then the threats, named by category rather than by fear.

- Spoofing — someone signs in as the finance manager: authentication and session identity in Module 2, federated login in Module 4.
- Tampering and elevation — a non-manager approves their own request: authorization inside the service, Module 3, not a hidden button.
- Information disclosure — a document reaches someone outside its department, or an injected search term returns forbidden rows: Module 5.
- Repudiation — nobody can prove who approved what, or when: the audit trail and correlation IDs, Module 6.
- Denial of service — the login page becomes a guessing endpoint, or an oversized upload fills the disk: Modules 2 and 7.

Rate every row twice, for likelihood and for impact. An internal audience changes those numbers without removing them: a curious colleague is likelier and milder than an outsider, a departing employee taking the document store both plausible and severe. The most useful column is the last one, naming the module

that implements the mitigation — a row with none against it is either accepted risk, which should say so in writing, or a gap you found for free.

Putting each control where it is enforceable

A control only counts at a boundary it can watch. Removing the Admin tab, disabling the Approve button, filtering a grid to the current department — each happens on the untrusted side of the session boundary, so each can be worked around by asking for an event the UI no longer offers. They are worth doing, because a UI that offers work the user cannot perform is a bad UI, but they are user experience, not security.

AccessOps therefore puts every decision that matters in a service on the server side of that boundary:

`DocumentService` decides whether this caller may read this document, `ApprovalService` whether this caller may approve this request, and both read identity from the session, not from a parameter the screen supplied. The UI reflects those decisions and never makes them. It also makes the rules testable: a service can be called with hostile arguments in a unit test.

The lab's control placement map is one row per control: the rule, the boundary it sits on, the class that enforces it, and how you would prove it works. It turns the diagram into a plan the remaining six modules tick off line by line.



Security habit

The browser is never trusted. Every decision that matters — who the user is, what they may do, which record they may see — is made on the server in a service, and the UI only reflects it. If a rule exists only as a hidden button, it does not exist.



Learning objectives

By the end of this module you'll be able to:

- ✓ Describe what a Wisej.NET client actually receives — rendered widgets and events, not your code or data — and what still crosses the wire.
- ✓ Identify the trust boundaries in a Wisej.NET app: browser, session, server process, database and external services.
- ✓ Build a threat model for a line-of-business app: identity, authorization, data exposure, injection, session abuse and denial of service.
- ✓ Place each security control at the boundary where it is enforceable.

Lab / Exam Guide

What you'll build in the hands-on lab, the suggested approach, and the required deliverables.



Hands-on lab

Create the AccessOps solution and write its threat model: draw the trust boundaries, list the assets (employee records, documents, approvals), enumerate the entry points (login, uploads, downloads, custom endpoints, the AI assistant), rate each threat, and map every mitigation to the module of this course that implements it. Build the shell with a placeholder login page.

Starting point

You start from nothing but an empty Wisej.NET Web Application project named AccessOps, its generated start-up window and the default configuration files. In this module you create the shell — a placeholder login page and the navigation for the departments, employees, documents, approvals and admin areas the rest of the course fills in — and the written threat model that every later module is measured against.

Suggested approach

- ✓ Create the AccessOps solution and a placeholder login page with a user name box, a password box and a Sign in button whose handler does nothing yet except set a status label, plus empty screens for departments, employees, documents, approvals and the admin console.
- ✓ Draw the trust boundary diagram with four regions — browser, Wisej.NET session, server process, and the database and external services — and mark on it exactly what crosses each line, including uploads, downloads, custom handlers and JavaScript interop.
- ✓ Write the asset and entry point inventory: employee records, documents, approval decisions, the audit trail and the credentials on one list; the login page, every screen's event handlers, the Upload control, downloads, custom HTTP handlers and the AI assistant on the other.
- ✓ Build the threat table by walking each entry point through spoofing, tampering, repudiation, information disclosure, denial of service and elevation of privilege, then rate each row for likelihood and impact and name the mitigation and the module that delivers it.
- ✓ Produce the control placement map: one row per control giving the rule, the boundary it sits on, the class that will enforce it, and the evidence you would show a reviewer that it works.

Required deliverables

- ✓ AccessOps solution with a placeholder login page
- ✓ Trust boundary diagram
- ✓ Asset and entry point inventory
- ✓ Threat table with likelihood, impact and mitigation
- ✓ Security control placement map

Student review questions

- ✓ Your employee grid is bound to full employee records and the salary column is hidden in the designer. Describe what the browser is holding, and name the change that actually removes the salary from the wire.
- ✓ Two people are signed in to AccessOps on the same server. Explain which pieces of state they share by default, and what would have to be true for one of them to see the other's current department.
- ✓ Pick the threat in your table with the highest impact and argue the opposite case: what would have to be true about AccessOps for that mitigation to be a waste of the team's time?



Security habit

The browser is never trusted. Every decision that matters — who the user is, what they may do, which record they may see — is made on the server in a service, and the UI only reflects it. If a rule exists only as a hidden button, it does not exist.