Titles:

* Don’t Write Secure Code; Design Secure Systems
* Stop Writing Security Code Every Day! (Just Write It Once)
* Put Security In Your Framework, Not Your Features
* Don’t “Code” For Security, Architect For It
* Spend Less Time Writing Secure Code

Abstract:

The best way to build secure systems is to stop writing security-related code on a daily basis.

Developers have their hands full with complex systems, confusing business rules, technical edge cases, responsive UIs, etc. Security requirements, when they even exist, are repetitive to implement, hard to test, and often get crowded out by other demands. When developers handle security on a feature-by-feature basis, the result is a wildly inconsistent mess of security holes.

In this session developers and architects will learn real-world techniques for designing security into the application framework itself, rather than leaving it up to individual features. You’ll see how to implement access control in your data access layer, declaratively handle permission checks with Attributes and Annotations, automate security testing with static analysis, and more. Come learn how secure software *design* can dramatically reduce the day-to-day burden of secure *coding*.

Code samples will be in C# and ASP.NET MVC, but the focus will be on techniques and concepts that easily generalize to other platforms.

Notes:

By “framework” I mean “the custom code you write that provides the foundation for your feature code”. I generally like to separate my general purpose code (like security) from my feature-specific app code, and that approach lends itself well to handling security in a more global way.

The main point of this talk is that if you ask devs to do the same 5 things over and over again for every feature, they become numb to those things. Things get overlooked, or forgotten, or implemented differently because of copy/paste/modify, or QA gets lax in testing them, and you end up with no real consistency of security. However, if you bake security into the application framework then devs don’t have to *remember* to do as much, and once it’s built they get security on new features more or less “for free”. It’s the whole “pit of success” thing.

I intend to draw from my existing application code examples like these:

* Data access methods that implement access control by taking both the ID of the thing they are retrieving, and the ID (or credential set) of person making the request.
* MVC endpoint authentication using Attributes to declare a set of permissions needed to access the endpoint (instead of putting the perm check directly in the MVC action body)
* API Key authentication using action filters
* Static analysis to ensure that all MVC controller actions in a specific part of the site have a permission attribute [i.e. if we know the attribute works correctly, and we know every possible endpoint declares an attribute, then QA doesn’t have to test each endpoint directly. And if QA has a data structure showing which endpoints should require which perms, that can be handled via static analysis as well]

I have a wealth of existing C# code to pull from. I may do a bit of research to include some Java samples as well, and if time permits I may research ways of applying these concepts to SPA type apps as well.