Outline:

Intro – 5 min

SQL injection protection via parameterized sql (canonical example)

Output encoding (manual in ASPX, automatic in Razor)

[x] Authentication: “secure by default” vs [Authorize]

[x] Authorization: Permission checks via attributes

[x] Authorization: AOP for masked values

[x] Access control: in data access code

[x] Access control: Row-level security in SQL Server

[x] Anti CSRF tokens

[ ] Testing: static analysis

[x] API bearer token check via Attributes (Web API)

~~[x] AOP for encryption~~

Static analysis

* Controllers w/out permission check
* UI testing?
* HP Fortify

Anti CRSF Tokens

* ASP.NET
* ?

<https://github.com/jkuemerle/EncryptedType>

IL merge?

Build security into the process (case template w/ security section)

OAuth token management in the news recently

Conclusion – 5 min

“Census XML Gateway” = a proxy. “Oracle XML Gateway” (gives external access to ESB)

* Access control in data access code (C#)
* Permission checks via attributes (C#)
* API bearer token check via Attributes (Web API)
* Permission checks via Annotations (Java?)
* Static analysis
  + Find all controllers without a permission check

Hello, and welcome to “Don’t Write Secure Code”. I’m Seth Petry-Johnson, and unlike some of the other speakers in this track, I am not a security professional. I’m just a normal programmer, although I do have a security related confession to make.

**(click)**

That confession is that I hate writing secure code.

Now, I don’t hate *having* secure code, and I don’t *want* to be hacked or to leak my user’s data, but when I’m building a feature and I’m elbows deep in complex business logic or functional requirements, security concerns feel like a distraction to me. They just aren’t interesting because it’s always the same thing over and over again: the user has to be logged in to do this. They have to have a specific permission to do that. Alice shouldn’t be able to see Bob’s data. Bob shouldn’t be able to get all ticked off about something and drop the user database through SQL Injection. Etc, etc. When I’m really engaged in a business problem, I want to be “all in” on that problem, and the constant need to implement security requirements, feature after feature, bums me out.

After I wrote this line, I realized that a better way of expressing myself is that

**(click)**

I hate writing “secure features”. I want my application logic to be clean and simple and elegant, not sullied up with those tired old requirements restricting what Bob and Alice are allowed to do.

And after I wrote *this* line, I realized that what I’m *really* trying to say is that

**(click)**

I hate implementing *cross-cutting security concerns* by repeating the same patterns over and over again in my *feature-level code*.

And after I wrote *this* line, I realized that I needed a shorter title for this talk. But this is what the next 50 minutes are all about. I’m going to show you <x> different techniques for getting those security concerns out of your feature code and into your application framework. I want the feature code in my system to be clean, simple, elegant, and secure by default. And that’s what you’re going to see today.

**(click)**

Here’s our agenda:

First we’re going to define what I mean by “cross cutting” security concerns, because those are the things that we’re going to focus on.

Second, I’m going to show you <x> different techniques for moving those concerns into an application framework. My examples are in .NET and JS, but parallels exist in other platforms as well.

Lastly, I’m going to <???>.

**(click for “Cross Cutting”)**

My bottom line assertion here is that if you’re relying on developers to think about routine security concerns day in and day out, for every single feature they build, then you’re going to end up with security holes.

Instead, I want to show you some techniques for handling security concerns at the framework level. The idea is that while developers might hate writing secure code on a feature by feature basis,

**(click)**

… writing “security code” at the framework level isn’t so bad.

As an architect, my goal is to make security checks a global, invisible thing wherever possible. And if developers do need to consider security on a feature-by-feature basis, I want it to be as easy and seamless as possible to implement those checks so that they can stay focused on the business feature at hand.

**(click – agenda)**

So here’s the agenda for today. First I want to briefly explore the reasons that developers tend to overlook security when writing feature code, and then I’m going to show you a handful of different techniques that you can use to build secure systems at a lower level.

This talk is basically a hodgepodge of techniques that I’ve used over the years to be lazy. Some of these things are, by definition, for a specific language or platform only. Others are patterns that can be applied in many tech stacks. It’s doubtful that everyone in this room is going to find all of these techniques applicable to them, but I’m hoping that everyone here will leave with at least one or two concrete examples that you can build off of in your own systems.

**ANTI CSRF**

Normally requires two things:

* Html.AntiForgeryToken() inside the form (creates hidden field and sets cookie)
* [ValidateAntiForgeryToken] on the action

Instead, tweak framework to make that automatic