Mitigating Top Security Threats

Peter Kellner
MICROSOFT ASP.NET MVP AND CONSULTANT

@pkellner linkedin.com/in/peterkellner99 PeterKellner.net

Category:OWASP Top Ten Project

Help

Home
About OWASP
Acknowledgements
Advertising
AppSec Events
Books
Brand Resources
Chapters
Donate to OWASP
Downloads
Funding
Governance
Initiatives
Mailing Lists
Membership
Merchandise
News
Community portal

Main | OWASP Top 10 for 2017 Release Candidate | OWASP Top 10 for 2013 | OWASP Top 10 for 2010
Translation Efforts | Project Details | Some Commercial & OWASP Uses of the Top 10

The release candidate for public comment was published 10 April 2017 and can be downloaded here. OWASP plans to release the final OWASP Top 10 - 2017 in July or August 2017 after a public comment period ending June 30, 2017.

Constructive comments on this OWASP Top 10 - 2017 Release Candidate should be forwarded via email to the OWASP Top 10 Project Email List. Private comments may be sent to Dave Wichers. Anonymous comments are welcome. All non-private comments will be catalogued and published at the same time as the final public release.

Injection

Injection

SQL

injection

attacks

Injection

SQL injection attacks

Active Directory attacks

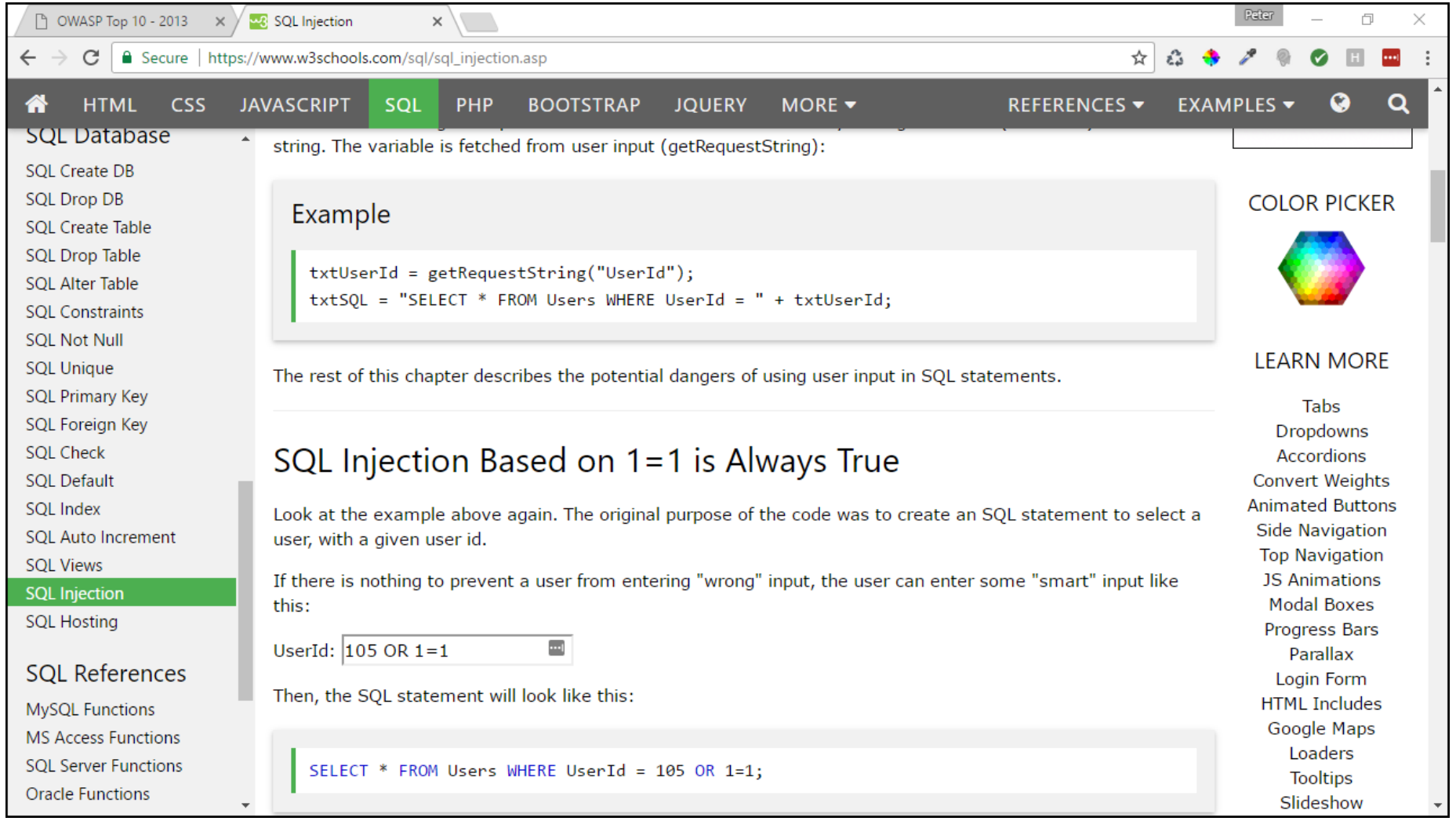
Injection

SQL injection attacks

Active Directory attacks

LDAP attacks

SQL Injection



SQL Injection

Example

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

The rest of this chapter describes the potential dangers of using user input in SQL statements.

SQL Injection Based on 1=1 is Always True

Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id.

If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

UserId: 105 OR 1=1

Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

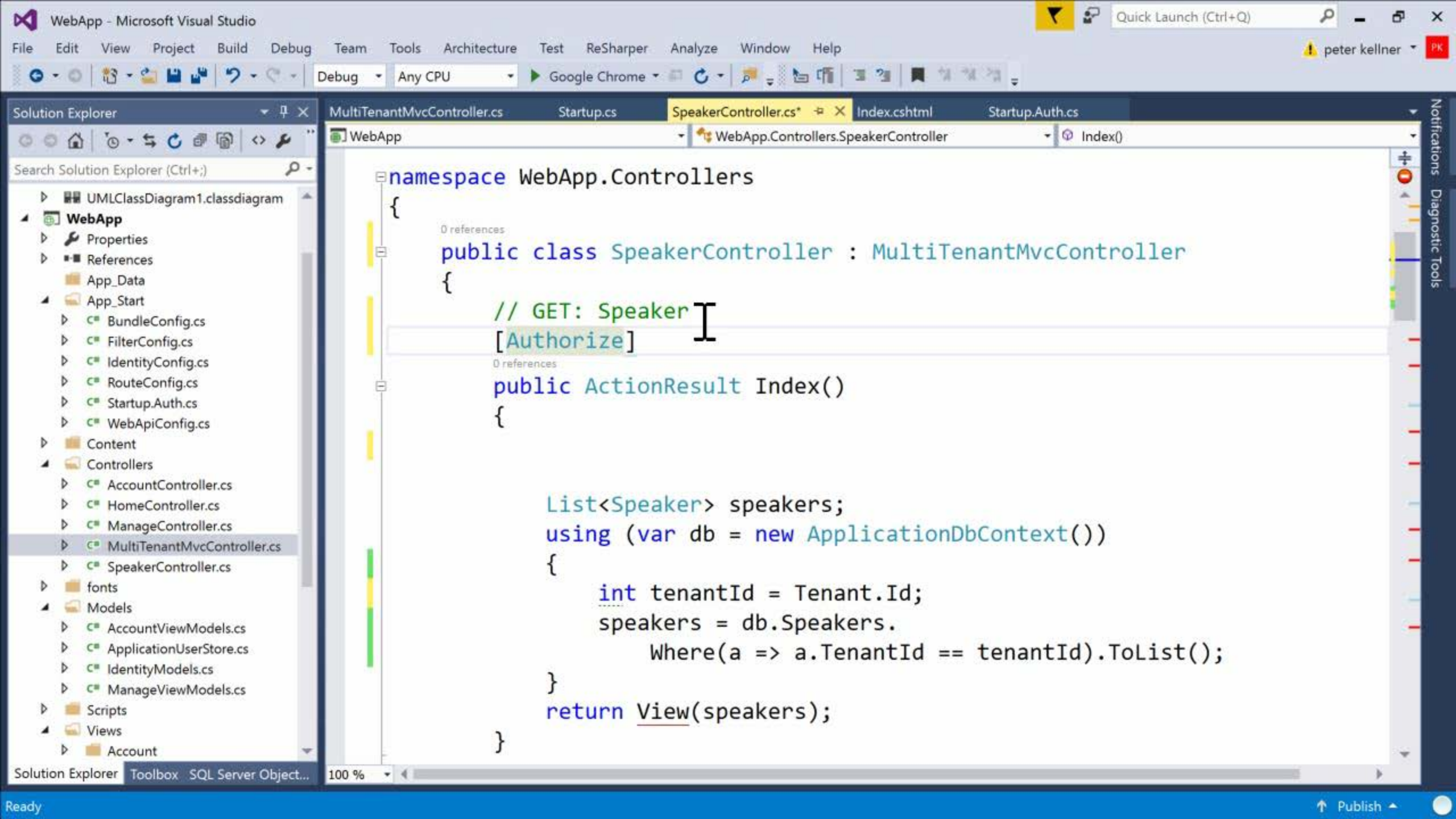
OWASP TOP 10

- A1 – Injection
- A2 – Broken Authentication and Session Management
- A3 – Cross-Site Scripting (XSS)
- A4 – Broken Access Control
- A5 – Security Misconfiguration
- A6 – Sensitive Data Exposure
- A7 – Insufficient Attack Protection
- A8 – Cross-Site Request Forgery (CSRF)
- A9 – Using Components with Known Vulnerabilities
- A10 – Underprotected APIs

Broken Authentication and Session Management

Cross-Site Scripting (XSS)

Broken Access Control


WebApp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Architecture Test ReSharper Analyze Window Help
peter kellner
Debug
Any CPU
Google Chrome
Solution Explorer
Search Solution Explorer (Ctrl+;)
UMLClassDiagram1.classdiagram
WebApp
Properties
References
App_Data
App_Start
BundleConfig.cs
FilterConfig.cs
IdentityConfig.cs
RouteConfig.cs
Startup.Auth.cs
WebApiConfig.cs
Content
Controllers
AccountController.cs
HomeController.cs
ManageController.cs
MultiTenantMvcController.cs
SpeakerController.cs
fonts
Models
AccountViewModels.cs
ApplicationUserStore.cs
IdentityModels.cs
ManageViewModels.cs
Scripts
Views
Account
Solution Explorer
Toolbox
SQL Server Object...
MultiTenantMvcController.cs
Startup.cs
SpeakerController.cs*
Index.cshtml
Startup.Auth.cs
WebApp
WebApp.Controllers.SpeakerController
Index()
namespace WebApp.Controllers
{
0 references
public class SpeakerController : MultiTenantMvcController
{
// GET: Speaker
[Authorize]
0 references
public ActionResult Index()
{
List<Speaker> speakers;
using (var db = new ApplicationDbContext())
{
int tenantId = Tenant.Id;
speakers = db.Speakers.
Where(a => a.TenantId == tenantId).ToList();
}
return View(speakers);
}
100 %
Notifications
Diagnostic Tools
Ready
Publish

Security Misconfiguration

Sensitive Data Exposure

Insufficient Attack Protection

Cross-Site Request Forgery (CSRF)

Using Components with Known Vulnerabilities

Underprotected APIs

Summary

OWASP top 10

ASP.NET Core related

Stay connected to OWASP