ASP.NET Security

Keith Brown, DevelopMentor

www.develop.com

Outline

- A Tale of Three Security Contexts
- Forms Authentication
- Three Common Security Holes you can Avoid

A tale of three security contexts

- **You need to decide what security context your ASP.NET pages should run within**
  - This is not as simple as picking some account…
- **It's helpful to have a picture of how the ASP.NET pipeline processes an HTTP request**

The ASP.NET HTTP pipeline

SYSTEM

?

aspnet_isapi.dll

INETINFO.EXE

ASPNET

?

HttpModule

HttpHandler
(page)

ASPNET_WP.EXE

The two unmanaged security contexts

- **Process Token**
  - If present, used by:
    - CreateProcess – child will get copy of process token
    - DCOM calls – unless “cloaking” is turned on
    - Any attacker who exploits a buffer overflow
- **Thread Token**
  - If present, used by:
    - File system, to make access control decisions
    - Most other operating system access control decisions, such as printing, accessing services, opening kernel objects
    - SQL Server, when using integrated security

Accessing the unmanaged security contexts

- **Accessing the unmanaged security context**
  - Use WindowsIdentity.GetCurrent
  - This will give you an IIdentity representing the thread token
  - If no thread token present, gives you the process token
- **If you want to see both your thread and process token**
  - RevertToSelf strips off the thread token
  - Convenient for experimentation, but something you probably won’t normally do in practice

Controlling the process token

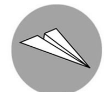
- **Remember, this affects the entire worker process**
- **Edit machine.config**
  - <processModel userName='name' password='password'…
- **Two special settings you can use**
  - Run as a local, low privilege machine account (ASPNET)
    - userName='Machine'
    - password='AutoGenerate'
  - Run as SYSTEM (yikes!)
    - userName='System'
    - password='AutoGenerate'

Controlling the thread token

- **Each web app can control this independently**
  - By default you have no thread token
- **Edit web.config**
  - <identity impersonate=‘true’/>
  - Will use a copy of IIS thread token, so use IIS admin tools to decide what you want your token to look like
- **Or, heaven forbid**
  - <identity impersonate=‘true’ userName=‘name’ password=‘cleartextPassword’/>

The managed security context

- **The third security context is more abstract**
  - Can be a simple wrapper over a token from IIS
  - Can be something completely different
    - Application defined identity and roles from Forms Auth
    - MS Passport identity
    - Anything else you can cook up
- **Accessing the managed security context**
  - Use HttpContext.User, or Page.User (same thing)



Controlling the managed security context

- **Each web app can control this independently**
  - Affects entire web app, must configure at virtual root
- **Edit web.config**
  - <authentication mode='None | Windows | Forms | Passport'/>
- **None**
  - Managed security context says every user is anonymous
- **Windows**
  - Mimics what IIS decides the security context should be
  - This is the default setting in IIS config
- **Forms**
  - You take authentication into your own hands via a login form
- **Passport**
  - Use MS Passport authentication and notion of identity

Forms Authentication

- **Forms Authentication is the most common way to make use of the managed security context**
  - You can implement your own user account database, or integrate with an existing one
  - Doesn't require clients to have an NT account
- **This is basically just a formalization of code we all used to write anyway**
  - Redirect the user to a login page
  - Send a cookie to the user on successful login
  - Look for the cookie at each future request to determine who is logged in
- **Here's the basic data flow…**

Client

Server

HTTP Request

GET /foo/foo.htm

Initial page request

No authentication cookie - redirect to login.htm

Login

Redirect to login.htm

login.htm

POST /foo/login.htm
username='Bob'
password='geek'

Login request

Authenticate Bob grant cookie, and redirect to foo.htm

Cookie granted
Foo.htm content sent

HTTP/1.1 200 OK
Set-Cookie: AC=99AF12...
<foo.htm contents>

HTTP Request

GET /foo/bar.htm
Cookie:AC=99AF12...

Subsequent page request

Valid authentication cookie - provide page

Bar.htm content sent

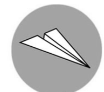
HTTP/1.1 200 OK
<bar.htm contents>

Forms Auth Demo

- Let’s implement forms auth and see just how it works

Things to remember

- **Stealing the cookie == stealing the login**
  - How will you protect the cookie from being stolen?
  - Most commonly done using SSL
  - Remember the cookie is sent with every request to your site, even for images
  - “Secure” cookie is an option, but this is a SHOULD, not a MUST in the HTTP spec
  - Persistent cookies are stored in the users profile
    - Can be stolen if attacker gets physical access to the machine



Three common security holes you can avoid

- SQL injection attacks
- Relying on client side validation
- Cross site scripting

SQL injection attacks

- **Be very careful when building SQL queries**
  - string concatentation that includes user supplied data is evil!
- **What's wrong with the following C# code?**
  - imagine this was the code behind an ASP.NET login form

```
string sql = "select * from users where name='" +
             txtName.Value +
             "' and password = '" +
             txtPwd.Value +
             "'";
cmd.CommandText = sql;
IDataReader reader = cmd.ExecuteQuery();
```

SQL injection attacks, cont.

- **Here's the input you'd expect to get from a legitimate user**
  - Name: **Bob**
  - Password: **nU3!gx7**
  - Resulting SQL:

```
select * from users where name='Bob' and pwd='nU3!gx7'
```

- **Here's what an attacker might send instead**
  - Name: **Bob' --**
  - Password: **hack**
  - Resulting SQL:

```
select * from users where name='Bob'--' and pwd='hack'
```

Avoid SQL injection attacks

- **Use parameterized queries**
  - don’t build SQL statements using string concatenation
- **Use regular expressions to filter incoming user input**
  - you should do this for all user input
  - create an expression that represents legal input
  - don’t try to guess the illegal cases
  - Check out the validation controls in ASP.NET!

Don’t rely on client side validation for security

- **Know what client side validation is for**
  - gives clients a better user experience
  - reduces load on your server from accidental bad input
- **Client side validation provides no real security**
  - clients don’t have to use your form to submit requests
  - always validate input when it arrives at the server
  - think about Perl’s “taint checking” and try to apply the same ideas to your own code
- **ASP.NET validation controls are a great utility**
  - provides client with immediate feedback on errors via Jscript
  - provides server protection by validating input on server side

Never echo unfiltered input back as HTML

- **Known as “cross site scripting”**
- **The basic problem**
  – one can submit HTML that is then served to another
  – HTML can contain scripts
- **What can happen to a victim**
  – cookies can be stolen
  – COM objects can instantiated and scripted with untrusted data
  – user input can be intercepted
- **How to avoid cross site scripting**
  – all data a user submits should be carefully filtered before you output it as part of your HTML stream

For more information

- **Essential .NET Security, the short course**
  - 4 day course from DevelopMentor
  - New course starting in November of this year
  - http://www.develop.com
- **A tale of three security contexts**
  - Short article I wrote for DevX
  - http://portals.devx.com/summitdays