In this topic, you will find a list of the most important security settings which can be configured in the web.config file.

Authentication

You can set the default authentication mode for your website using the **mode** attribute, which has these values: Windows, Forms, Passport, None.

```
<authentication mode="Windows">
</authentication>
```

See page authentication Element (ASP.NET Settings Schema) for reference.

Forms authentication type

The **Forms** authentication type can be further configured using the **forms** element. It is recommended to use session cookies (not to use cookieless authentication) to prevent session hijacking. This can be done by changing the **cookieless** attribute of the form element. You can also set these attributes:

- **cookieless** – defines whether cookies are used and how they are used.
- **timeout** – specifies the number of minutes after which the authentication cookie expires.
- **slidingExpiration** – specifies, whether the expiration time of an authentication cookie should be reset upon each request in a session. If set to true, then the authentication cookie is refreshed with every request, so the cookie does not expire so easily.

```
<authentication mode="Forms">
        <forms loginUrl="CMSPages/logon.aspx" name=".ASPXFORMSAUTH" cookieless="
UseCookies" requireSSL="true" timeout="15" slidingExpiration="false" />
</authentication>
```

The most secure solution is to set short timeout interval (e.g., 15 minutes) and disable sliding expiration. This way, if the attackers stole the session, they would only have a limited time to abuse it. However, the users will have to submit their credentials and authenticate every time the session expires (set by the timeout attribute).

See page forms Element for authentication (ASP.NET Settings Schema) for reference.

Cookies

You should set the following attributes related to cookies:

- **httpOnlyCookies** – adds a *httpOnly* flag to cookies and makes it impossible to read cookies from the client. This serves as protection against XSS (for example prevents attackers from reading the session ID from cookies or the forms authentication ticket from the authentication cookie).
- **requireSSL** – configures cookies to require an SSL connection, which prevents communication eavesdropping. Note: Only enable the *requireSSL* attribute if your website is configured to use an encrypted HTTPS connection (see Configuring SSL).

```
<system.Web>
        <httpCookies httpOnlyCookies="true" requireSSL="true">
</system.Web>
```

Session

You should use cookies instead of query strings to pass the session ID. This prevents session stealing, as it is easier to steal the session ID from query strings than cookies. See Session protection.

To protect the sessions against attacks, you should set the following attributes in the **sessionState** element:

- **mode** – specifies where to store session state values.
- **cookieless** – specifies how cookies are used for a Web application.
- **timeout** – specifies the number of minutes a session can be idle before it is abandoned.

```
<sessionState mode="InProc" cookieless="false" timeout="20" />
```

See page sessionState Element (ASP.NET Settings Schema) for reference.

Error messages and disabling the debug and trace

You should unify handling of all types of errors and exceptions in your application by adding the **<httpErrors>** element into the **<system.webServer>** section of your web.config file. See Handling general errors for more information.

Before deploying your website to the live environment, you should also disable debugging and tracing in the web.config file, as this information should not be revealed to the users.

You can disable **debugging** in your application by including this code in the *<system.web>* section:

```
<system.web>
        <compilation debug="false"/>
</system.web>
```

You can disable **tracing** in your application by including this code in the *<microsoft.web.services3>* section of your web.config file:

```
<microsoft.web.services3>
        <diagnostics>
                <trace enabled="false"/>
        </diagnostics>
</microsoft.web.services3>
```

Note that you can also configure tracing for web pages individually using the **Trace** attribute in the **@ Page** directive at the top of your .aspx files.

```
<%@ Page Trace="true" %>
```

See How to: Enable Tracing for an ASP.NET Page.

Request validation

Request validation is a mechanism, which ensures that the ASP.NET application does not process potentially dangerous requests (possible XSS attacks). In Kentico, the request validation is disabled by default, because some parts of the system (for example, the WYSIWYG editor) send such requests, that would be suspicious to the validator. However, you can change this setting individually and enable request validation only for chosen live pages in the **@ Page** directive:

```
<@ Page validateRequest="false" %>
```

See Request Validation in ASP.NET.

View state validation

In Kentico, the view state validation is encoded using the machine key and also a private user key.

You can enable view state validation globally in the web.config file:

```
<configuration>
  <system.web>
    <pages enableViewStateMac="true" />
  </system.web>
</configuration>
```

You can find more information about the view state validation in Cross site request forgery (CSRF/XSRF).

It is also possible to encrypt the view state to further increase its protection. See Encrypt ViewState in ASP.NET 2.0 for more information.

Encrypting individual sections of the web.config file

You can encrypt chosen sections of the web.config file, which can prevent attackers from obtaining sensitive information (passwords, connection string, etc.) if they manage to get hold of the file. Encrypting can be done using:

- DPAPI – this tool provides better security, but is not suitable for web farms. See How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI.
- RSA – this tool is suitable for securing the web.config files on web farms. See How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA.

Encoding is supported natively by ASP.NET, so the web application does not have to provide any additional support.