

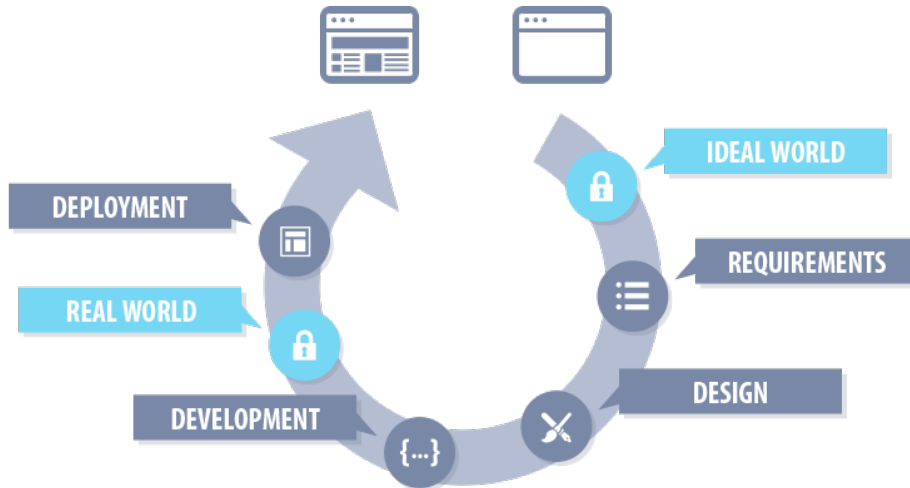


Kentico CMS 7.0 Security Guide

1. Security guide – Introduction	3
2. Designing secure web applications	3
2.1 Securing and protecting the system	4
2.1.1 Flood protection	4
2.1.2 Banned IPs	5
2.1.3 Clickjacking	6
2.1.4 Session protection	6
2.1.5 Autocomplete deactivation	8
2.1.6 Preventing duplicate poll voting and content rating	9
2.1.7 Screen locking	9
2.1.8 Spam protection (CAPTCHA)	10
2.1.9 The event log and security debug	10
2.2 Securing user accounts and passwords	11
2.2.1 Password encryption in database	12
2.2.2 Password strength policy and its enforcement	13
2.2.3 Password expiration	14
2.2.4 Invalid logon attempts	15
2.2.5 Forgotten password	16
2.2.6 Unlocking an account	18
2.2.7 Custom password calculation	19
2.3 Configuring permissions securely	20
2.3.1 Special permissions	22
2.4 Managing external authentication	24
2.5 Securing the Staging and REST web services	27
2.6 Configuring e-mail confirmations	27
2.7 PCI compliance	28
3. Developing secure web applications	30
3.1 Secure coding recommendations	30
3.1.1 Macros and security	30
3.1.2 Query string hashing	32
3.1.3 Handling error messages securely	33
3.1.3.1 Designing secure error messages	33
3.1.3.2 Creating custom error handling pages	35
3.2 Avoiding security vulnerabilities in code	36
3.2.1 Cross site scripting (XSS)	36
3.2.2 SQL injection	40
3.2.3 Argument injection	42
3.2.4 Command injection	43
3.2.5 Lightweight Directory Access Protocol (LDAP) injection	44
3.2.6 XPath injection	45
3.2.7 Cross site request forgery (CSRF/XSRF)	46
3.2.8 Directory traversal	47
3.2.9 Enumeration	49
4. Deploying web applications to a secure environment	50
4.1 Configuring SSL	51
4.1.1 SSL accelerator support	56
4.2 Restricting access to directories	57
4.2.1 Restricting access to the CMSHelp directory	59
4.3 Disabling unnecessary modules and services and keeping the system up-to-date	60
4.4 Hiding the system information	61
4.5 Minimal secure configuration	62
4.6 Web.config file settings	62
5. Security checklists	64
5.1 Security checklist – designing a website	65
5.2 Security checklist – developing a website	65
5.3 Security checklist – deploying a website	66

Security guide – Introduction

Security is an important factor in web development process. Ideally, security should be addressed at the beginning, before you even start planning and designing your projects. Reality though, is more complicated, and in many projects, security issues are considered after the development is done, right before deploying the website to a live environment.



This may be problematic, as dealing with security problems may become expensive in later stages of the process. The first thing you should do, is to determine how secure your website should be and which security protections you will have to implement.

Design phase

Configure Kentico so that it is safe for users and protected against spam bots and malicious users.

Development phase

See what you can do to secure your websites in the code. Utilize our secure coding recommendations and instructions on how to avoid vulnerabilities in code.

Deployment phase

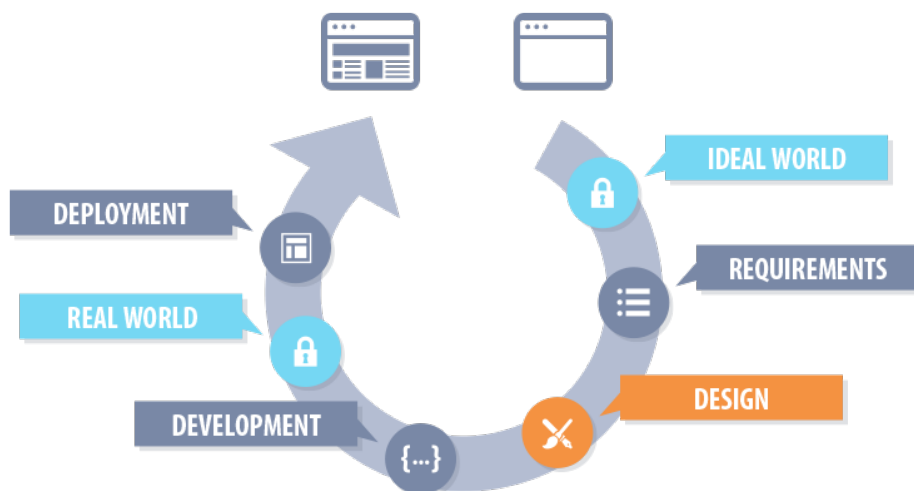
Configure your server and project before deploying it to a live environment.

Security checklists

After each stage, check whether you have addressed every important security issue:

- [Security checklist – designing a website](#)
- [Security checklist – developing a website](#)
- [Security checklist – deploying a website](#)

Designing secure web applications



Securing and protecting the system

In this section you will learn how to set up your system, so that it is protected against spam bots, fraudsters and malicious users.

Flood protection

Flood control is a form of spam prevention on forums and similar community services. It prevents the users from making posts to the forum in quick successions. The users usually have to wait for a short time period before making another post. This mechanism prevents spambots from flooding the forum with unsolicited messages.

Whenever a user makes a post, the mechanism checks, if the minimal time interval between posts has been exceeded. If the interval has been exceeded, the post is not saved. The checks can be performed against:

- **userID** (User based) - default for logged-in users.
- **IP** (IP based) - default for users that are not logged-in.

You can change the default settings using the **CMSUserBasedFloodProtection** web.config key:

```
<add key="CMSUserBasedFloodProtection" value="true" />
```

Supported modules

The flood protection is supported in these modules:

- Blogs (comments)
- Forums (posts)
- Message boards (posts)
- Messaging (sending messages)

This mechanism works across modules, so if a user gets blocked after posting comments on blogs, the user is also blocked on Forums, Message boards and in the Messaging module.



Do not forget to use CAPTCHA

Besides securing these modules using the flood protection, you should also include a CAPTCHA field in the comment, post and message forms.

Enabling and configuring the flood protection

You can enable the flood protection functionality in **Site Manager -> Settings -> Security & Membership -> Protection -> Flood protection** section.

Using the **Flood protection interval**, you can set the minimum time interval (in seconds) before the user can make another post.

Chat module

The Chat module has its own flood protection system. It is more complex and granular. The checks are performed when the user:

- creates a room,
- joins a room,
- posts a message,
- changes the nickname.

The checks are performed against the chat user ID. This module also has its own flood protection settings at **Site Manager -> Settings -> Community -> Chat**:

Flood protection	
Enable flood protection	<input type="checkbox"/>
Join room interval (seconds)	<input type="text" value="3"/>
Create room interval (seconds)	<input type="text" value="10"/>
Post message interval (seconds)	<input type="text" value="0.3"/>
Change nickname interval (seconds)	<input type="text" value="5"/>

Flood protection integration

If you want to integrate this mechanism in your own code or module, use the *FloodProtectionHelper.CheckFlooding* method:

```
if (FloodProtectionHelper.CheckFlooding(CMSContext.CurrentSiteName,
CMSContext.CurrentUser))
{
    // Don't save the message, display information about flooding to the user
}

// Save the message and continue
```

Banned IPs

IP banning prevents users with specified IP addresses from using your website. Kentico CMS provides these levels of IP address banning:

- **Access to the website** - users with the specified IP address cannot access the site at all.
- **Login** - users cannot log in to the site.
- **Registration** - users cannot register on the site.
- **All user actions** - users can enter the site, but they are not allowed to register or log in, and they are not allowed to add any content to the site (e.g., blog comments, board messages, etc.).

How IP banning works

In Site Manager -> Administration -> Banned IPs, you can specify, which addresses will be banned. When typing an IP address, you can use the asterisk (*) wildcard character to cover a range of IP addresses (for example, 192.168.0.*).

If a user has the **Access to the website** ban type, then the user gets redirected when trying to access your website. Other IP ban types are handled by individual modules and parts of the system. The user gets usually informed about not being able to complete the given action (log in or register).

The IP address is provided by the *HTTPHelper.UserHostAddress* property, which is a value either set in the system, or obtained from the *.NET (HttpContext.Current.Request.UserHostAddress)*.

When you ban an IP address globally, you have an option to **Allow sites to overwrite the ban**. This way, the site administrators can cancel the global ban (using the **Allow IP address for this site if the IP address is banned globally** option) on their sites.

Banned IPs are fully integrated into the system and affect all modules.

Banned IPs integration

If you want to integrate the IP banning module into your own code or modules, use the *BannedIPInfoProvider.IsAllowed* method:

```
if (!BannedIPInfoProvider.IsAllowed(CMSContext.CurrentSiteName,
BanControlEnum.Complete))
{
    // User is completely banned
}
```

Clickjacking

Clickjacking is a type of attack where the attacker tricks website users into clicking something different than what they see, thus performing an action that may, for example, reveal confidential data or have any other negative impact on the user.

In a typical clickjacking scenario, the attacker places a transparent frame with a page, that contains a button or a link, over another element on a website. The underlying element can be an image or a video, which the users expect to play when they click it. Instead, they click the concealed link or button. This way the attacker can make the users perform unintended actions, usually on websites, where the users are authenticated.

To prevent such attacks, Kentico CMS disallows embedding pages it renders into frames. It does that by including a special entry in the HTTP response headers:

```
X-frame-options: SAMEORIGIN
```

The header ensures that pages, which are displayed in frames, originate on the same server as the parent page. If they don't, browsers do not render them.

```
<add key="CMSXFrameOptionsExcluded" value="/Services" />
```

As a value, you can enter any alias path. All documents under this path will be excluded from the protection. You can specify multiple paths divided by a semicolon (;). Entering "/" turns off the protection altogether.

Session protection

We use sessions, because web is running on HTTP, which is a stateless protocol. However, in many web applications, we need to keep some state information, some context. This is the purpose of sessions. When a user opens a browser and navigates to some website, the web server of that website generates a session ID for this user. The session ID is sent with every request and it is a key for any session data (session data = state/context). These data are stored on the server.

The session ID can be passed to the requests in two ways:

- via a URL parameter,
- or using a cookie.



It is recommended to use cookies for passing the session ID. You can disable the cookieless sessions by changing the cookieless attribute of the form element on the **<system.web>** section of the web.config file:

```
<authentication mode="Forms">
    <forms cookieless="UseCookies" />
```

The session ends (in a typical case) after the user closes the browser or after the user is inactive for a specified amount of time.

There are basically three types of session attacks:

- Session stealing
- Session prediction
- Session fixation

Session stealing

A session can be stolen by stealing a session ID. When an attacker steals a user's session ID, he can get access to all of the session data. Because all these session IDs can be read by JavaScript, the most popular method for this type of attack is XSS. An attacker can send a crafted link to a victim with a malicious JavaScript. When the victim clicks on the link, the JavaScript runs and sends the cookie value of the current session to the attacker.

Session prediction

Can an attacker simply guess some random session ID? Most implementations of session IDs are long strings and guessing a correct session ID in linear time is impossible. But there are also bad implementations when an attacker can generate session IDs from known values. This technique is called session prediction. For example, a session ID can be a user name encoded in base64. Fortunately, in ASP.NET, session ID is a 120bit random number represented by a 20-character string. So, it is relatively safe.

Session fixation

In this case, the attacker lets the server generate a session ID. Then, the attacker sets a user's session ID to the generated ID. This is quite easy when session ID is given in a URL parameter. After that, the user and the attacker share the same session. For example, when the user gets authenticated, the attacker is authenticated as the user too.

The goal of all kinds of session attacks is the same – to get user's session data or achieve an identity forgery. This topic focuses mainly on **session fixation**, as XSS is explained in a different topic and we cannot influence how session IDs are generated. But generally, the implications of all three session attacks and the protective measures against them are similar.

Example of a session fixation

Let's have a simple .aspx page which saves a value to a session and also shows it:

```
<asp:Literal runat="server" ID="ltlSession"></asp:Literal>
<asp:TextBox runat="server" ID="txtValue"></asp:TextBox>
<asp:Button runat="server" ID="btnSend" Text="Save" />
```

In code behind, we handle the OnClick() event of the button:

```
Session["MyPrivateData"] = txtValue.Text;
```

On page load, we display the value via the literal:

```
ltlSession.Text = "My private Data:" + SessionHelper.GetValue("MyPrivateData") +
"<br/>";
```

How is the attack executed:

1. The attacker forges a link to this page with a session ID and sends the link to the user.
2. The user, unaware of the forgery, clicks the link.
3. The user sees the page normally and does not register anything unusual.
4. When the user saves some private data now, the attacker will be able to see them.

What can session fixation attack do

The main goal of the attacker is to read and manipulate with session data or an identity forgery. In both cases, it depends on the particular application how dangerous this can be. If the application stores sensitive data to sessions (for example, user passwords in plain text) and allows to show these data or allows to change them, the damage can be severe.

Finding session fixation vulnerabilities

In Kentico CMS, session fixation is possible, depending on application settings. You have to ensure that you do not store any sensitive information in sessions. The best way is to determine which variables are stored in sessions. Then, check how you can manipulate with them (read/change) and think about the risks – what damage can the attacker do by manipulating with them. You can find these risks just from the user perspective by inspecting application reactions, parameters and so on.

However, we still recommend code inspection. You can simply find all manipulations with session data by searching for **SessionHelper** and the **Session[]** array.

Avoiding session fixation

First of all, there is no native support for protecting against session fixation in ASP.NET. The best practice for protecting your application against session fixation is to regenerate the session ID after a user logs on. You can achieve this by changing the session ID to an empty string and letting ASP.NET generate a new one. However, by this action, you lose the session data.

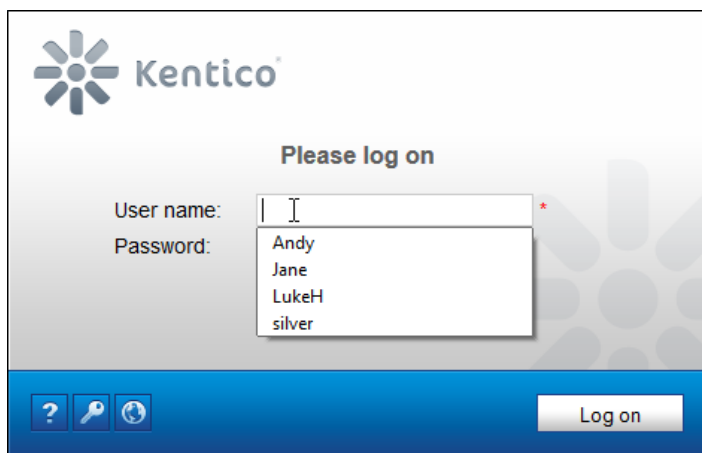
In Kentico CMS, you can utilize the **CMSRenewSessionAuthChange** key (insert it into the **appSettings** section of your **web.config** file), which enforces a change of a session ID on logon or logout. If you enable this setting, users will not be able to preserve their session data after logging in or out.

```
<add key="CMSRenewSessionAuthChange" value="true" />
```

There is also the "Impersonation" functionality which allows a global administrator to log on as another user. By implementing this functionality, we have basically secured Kentico CMS against session fixation because it tests if the authenticated user (taken from HttpContext – authentication cookie) is the same as the user in the session. If not, the user info in the session is changed to the right one. But it is still possible for an attacker to manipulate with other data. All you need to take care of is **not to save sensitive and critical information in sessions**.

Autocomplete deactivation

Autocomplete is a feature, which **remembers submitted user names** in login forms and also all words submitted through any forms in the system. In this topic though, we will focus only on the autocomplete functionality in login forms:

The image shows a screenshot of the Kentico CMS login interface. At the top left is the Kentico logo. Below it, the text "Please log on" is centered. There are two input fields: "User name:" and "Password:". The "User name:" field is active, and a dropdown menu is open below it, displaying a list of user names: "Andy", "Jane", "LukeH", and "silver". At the bottom of the form, there are three icons (a question mark, a key, and a globe) and a "Log on" button.

When users try to log in using a form, the autocomplete feature provides them with a list of already remembered user names. This is convenient for the users in many ways:

- The users do not have to type the whole user name every time they want to log in.
- If the users forget their user names, this feature can help them log in.
- It reduces discomfort of having to type the user names repeatedly on mobile devices.

However, using the autocomplete can pose a **security risk**. A malicious user who obtains user names from the autocomplete feature may gain access to the users' accounts, for example using a dictionary attack. Thus, you should always consider the damage a malicious user can do to the users' accounts. This threat mainly depends on the type of application you are creating and how this application will be used (on private computers only or in public places like schools, libraries, etc.).

You should disable autocomplete in applications working with:

- bank accounts,
- social media,
- sensitive information.

On the other hand, autocomplete can be useful in applications like:

- intranet,
- interest and hobby forums.

Disabling autocomplete

Autocomplete functionality can be disabled for the login forms using a HTML attribute:

```
<input name="Login1$UserName" class="LogonTextBox" id="Login1_UserName" type="text" maxLength="100" autocomplete="Off" />
```

In Kentico, you can use the following setting to disable autocomplete in login forms:

- In **Site Manager** -> **Settings** -> **Security & Membership** -> **Protection** uncheck the **Enable Autocomplete** option.

Preventing duplicate poll voting and content rating

When you include polls and content rating functionalities on your website, you usually want to assure, that each user can vote only once. This is very difficult or nearly impossible to achieve. You should keep in mind that there will always be users that will vote more than once and users who will not be allowed to vote at all. You can come across two known possible solutions to this problem:

- **Store a cookie in the users' browsers** - this is the solution **implemented by Kentico**. The problem is that technically skilled users can overcome this protection. They can switch to different browsers or delete the particular cookie in their browsers and vote again.
- **Store the votes in the database with the users' IPs** - the problem with this solution is that users and IP addresses are not mapped one to one. Many people share the same IP, for example in office buildings, and banning one user would ban the whole network in the building. Moreover, people can easily switch their IP addresses by moving between access points, using proxies or mobile devices.

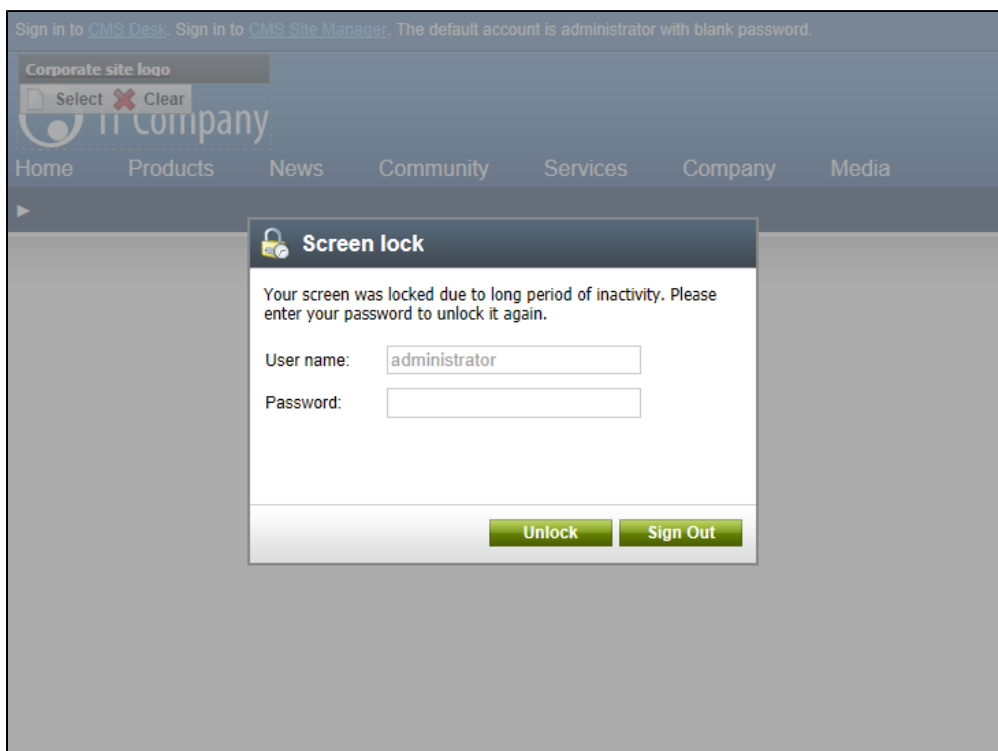
There is also another solution, which is preferable to the previous two:

- **Allow only signed-in users to vote and store a flag in the database** - you will have to ensure that users cannot easily create multiple accounts by, for example, manually approving registrations or by tying accounts to e-mail addresses. However, you will protect your voting systems, at least to a certain extent.

You should consider the level of protection the voting systems in your application require and implement your own solution if needed.

Screen locking

When users, who are signed in to the Kentico administration interface, leave their workstations unattended, someone else can tamper with the system. For this reason, Kentico allows you to set up automatic screen locking. This feature **locks the working area of the browser** with the administration interface after a specified time of inactivity. The users can unlock the screen after typing their password.



When using [Windows authentication](#), this feature does not work, as unlocking the screen would not require a password. The screen locking feature is useful only in a combination with the [forms authentication](#) or the [mixed mode authentication](#).



Note that when users try to unlock their screens, every failed attempt to log in is counted as an **invalid logon attempt** (if you have enabled this functionality).

Enabling screen locking

To enable screen locking, open **Site Manager** and navigate to **Settings** -> **Security & Membership** -> **Protection** -> **Screen lock**, then turn on the **Enable screen lock** setting.

The **Screen lock** settings group offers the following settings:

- **Lock interval** - time of user inactivity in minutes until their screen is locked.
- **Warning interval** - time in seconds before the actual lock during which the system displays a warning with countdown to the

lock.

When the warning is displayed, users can click **Cancel** to reset the lock interval. Note that the users have to either click Cancel or some button or label in the UI, as only moving the mouse is not enough to stop the countdown.



Spam protection (CAPTCHA)

Kentico CMS allows you to protect your website from automated spam bots. You can secure all forms where users enter data, by requiring users to type a security code called **CAPTCHA**.

You can use CAPTCHA to tell humans and computers apart in the following places:

- Blog comments
- Custom tables
- Document types
- Forms
- Forums
- Message boards
- Other web parts that allow user input

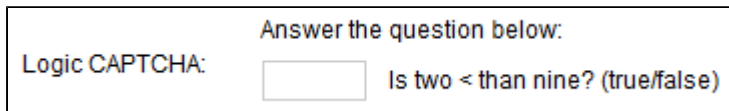
Changing the default CAPTCHA type

You can choose which CAPTCHA type will be used throughout the system:

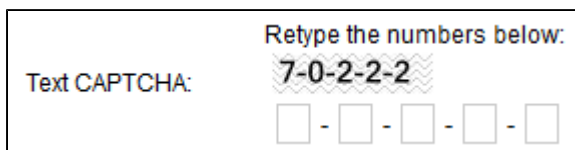
- **Simple** - prompts users to retype a sequence of numbers from an image.



- **Logic** - asks users to solve a simple arithmetic problem or to compare two numbers. Example: "one + four"; "Is six > than eight? (true/false)"



- **Text** - prompts users to retype a sequence of numbers, each number into an individual box.



We recommend to use **logic CAPTCHA**, as this is the most secure type we provide.

The default CAPTCHA type is **Simple**. To change the default type:

1. Go to **Site Manager -> Settings -> Security & Membership -> Protection**.
2. Under **CAPTCHA settings**, pick a **Control to use**.
3. Save the settings.

When you change the CAPTCHA type, all web parts that have CAPTCHA enabled use the new type. Also, all fields in custom tables, document types, and forms that use the Security code control, use the new type of CAPTCHA. Fields that use a specific CAPTCHA control don't change.

The event log and security debug

Event log

The event log is a place in the administration interface, where events and activities of the system are recorded. You can find it in **Site Manager -> Administration -> Event log**. From the security point of view, the events logged into the event log are mainly:

- Authentication attempts - both successful and failed.
- All bad requests - which includes attempts for SQL injections and other various hacking efforts.

You can find more information about event log settings in the [Viewing the system event log](#) topic.

Security debug

The security debug is a part of the debugging interface in Kentico. It allows you to view user security checks performed by the system. Information stored in this section include:

- IsGlobalAdministrator, IsEditor, IsInRole, IsInSite, IsCultureAllowed checks,
- URL hash validation,
- redirects to the AccessDenied page - for missing permissions and UI elements.

You can find the security debug interface in **Site Manager -> Administration -> System -> Debug -> Security**.

The screenshot shows the 'System' interface with the 'Debug' tab selected. Under 'System > Debug', the 'Security' sub-tab is active. A checkbox for 'Show complete context' is present. Two tables of security checks are displayed for the user 'Jane'.

Table 1: /KenticoCMS8_2/cmsitemanager/default.aspx?username=administrator (02:29:38)

User name	Operation	Result	Resource / Class / ID	Permission / UI element / Value
1 Jane	IsGlobalAdministrator	False		
2 Jane	RedirectToAccessDenied			

Table 2: /KenticoCMS8_2/default.aspx (02:29:36)

User name	Operation	Result	Resource / Class / ID	Permission / UI element / Value
1 Jane	IsInRole	False	_notauthenticated_	
2 Jane	IsInRole	False	_notauthenticated_	
3 Jane	IsInRole	False	_notauthenticated_	
4 Jane	IsGlobalAdministrator	False		
5 Jane	IsEditor	True		
6 Jane	IsAuthorizedPerTreeNode	Allowed	4	
	> IsGlobalAdministrator	False		
	> IsGlobalAdministrator	False		
7 Jane	IsGlobalAdministrator	False		

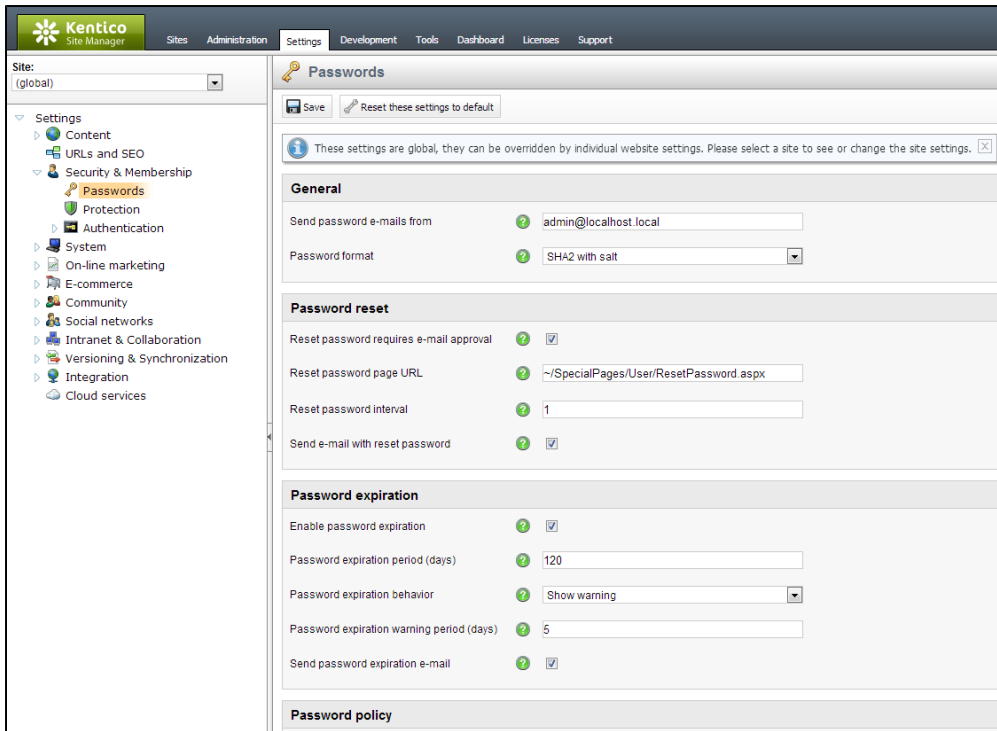
Enabling the security debug functionality

If you cannot see the security page, you have to enable the security debug first in **Site Manager -> Settings -> System -> Debug** by checking the option **Enable security debug**.

Securing user accounts and passwords

Passwords are a critical part of any authentication process. Kentico CMS provides various passwordrelated features that you can leverage to achieve the level of security required by your website.

These settings can be found in **Site Manager -> Settings -> Security & Membership -> Passwords**.



The features include:


- A set of password encryption format options.
- Ability to reset and recover user passwords.
- Ability to specify password strength and policy to enforce the specified strength.
- Ability to set time after which passwords expire and users are required to change them.
- Locking a user's account when they enter an incorrect password too many times.

Password encryption in database

There are multiple different formats that can be used to store passwords in the database. They may be saved either in plain text or as the result of a security hash function.

You can choose which option should be used in **Site Manager -> Settings -> Security & Membership -> Passwords** via the **Password format** setting:

- The default and recommended option is **SHA2 with salt**.

 The Password format setting is stored in a database table *Users*.

Password salt

Passwords are usually stored using the **SHA2** hash format with the additional application of a **salt**. Salt is a string that is appended to passwords before they are encrypted, which helps protect the passwords against dictionary or other types of brute force attacks. It also ensures that every user has a different password hash, even if multiple users have the same password.

In Kentico, we add two types of salt to the password:

- **Custom salt** - by default, the `UserGuid` column is used to append the GUID of each user to the passwords. You can customize this setting to use a different table column as a password salt. Add the following key into the **<appSettings>** section of the `web.config` file and type the column name as a value:

```
<add key="CMSUserSaltColumn" value="UserCreated" />
```

- **Password salt** - to increase the length of the salt (to further improve the security of hashed passwords), you can define a custom string, which will be appended to every password. You only need to include the following key into the **<appSettings>** section of your `web.config` file:

```
<add key="CMSPasswordSalt" value="SaltText" />
```



Password and salt are composed in this way:

Password Custom salt Password salt



Please keep in mind that, if you change the password format, it only affects how future passwords will be stored. Existing passwords will remain unchanged. You will need to reset all passwords, so that they are stored in the new format.

For this reason, it is recommended to set the appropriate format directly after installation, before you create user accounts or allow users to start registering.

Password strength policy and its enforcement

The system can be configured to use a password policy, which means that new passwords entered by users will be validated according to a certain set of requirements. Passwords that do not meet the specified conditions will be rejected.

Configuring a password policy

To enforce a password policy on your website, enable the **Use password policy** setting in **Site Manager -> Settings -> Security & Membership -> Passwords**. The specific rules of the policy can be configured through the remaining settings in the category:

- **Minimal length** - sets the minimum number of total characters required for user passwords.
- **Number of non alphanumeric characters** - sets the minimum number of non alphanumeric characters (i.e. any character except for numbers and letters) that must be present in a password in order for it to be accepted.
- **Regular expression** - can be used to enter a regular expression that will be used to validate user passwords. For example: `^(?=.*[a-z])(?=.*[A-Z]).*$`
This sample expression would require passwords to contain at least one lower case letter, upper case letter and number. The minimum amount of characters would be determined by the other policy settings.

The requirements defined by all three settings are combined together to form the final password policy.

How is password policy applied

The policy is applied in all sections of the website where a new password can be entered. This includes various types of web parts that display forms on the live site, such as **My account** or the **Registration form**, and the administration interface (**Administration -> Users**). The requirements of the policy, except for the regular expression, are additionally observed when the system automatically generates new passwords. This is also the case if the **Use password policy** setting is disabled, so you can affect how passwords should be generated even if you do not wish to set a policy for your website's users.

When you introduce a password strength policy, existing users are by default allowed to keep their passwords unchanged. To force existing users to observe the policy, enable the **Force password policy on logon** setting. With this setting enabled, the system will check whether a user's password meets the policy requirements every time a user logs in. When the password doesn't meet the requirements, the user is presented with a form that allows to change the password.

The screenshot shows a 'Reset password' form with a message box at the top stating: 'Your password doesn't meet all security requirements, please change it to ensure safety of your user account. Minimal password length is 6 characters. Password has to contain at least 1 non alpha numeric characters. Recommended password should be 12 characters long and should contain 2 non alpha numeric characters.' Below the message, there are input fields for 'Password:' and 'Confirm password:', a 'Password strength:' indicator showing a low level of strength, and a green 'Reset' button.

Password strength indicator

When a user types in a password, it is validated in real time and its status is reflected by an indicator below the field. If a policy is set, passwords that do not fulfill the requirements will be rejected with the *Not acceptable* status.

First name:	<input type="text" value="Andrew"/>
Last name:	<input type="text" value="Jones"/>
E-mail:	<input type="text" value="andy@localhost.com"/>
Password:	<input type="password" value="••••"/> Password strength: Not acceptable
Confirm password:	<input type="password"/>
<input type="button" value="Register"/>	

Valid passwords will have a different status displayed according to their relative strength, which is calculated based on the recommended values for the total password length (12 by default) and number of non alphanumeric characters (2 by default). If a password policy is not enabled for the website, the current strength status of passwords will still be shown, but only as a recommendation and all passwords will be accepted.

To help users come up with an appropriate password, you can use the **Policy violation message** setting to specify a text message that will be displayed to users who attempt to enter a password that does not fulfill the requirements of the password policy. If left empty, a default message will be shown, informing about the minimum password length and number of non alphanumeric characters. If you wish to use a regular expression, it is recommended to describe its requirements in a custom message. If your site has multiple cultures (languages) assigned to it, you can enter a different message for each language via the **Localize** (🌐) action available next to the setting's field.

i Customizing the password strength indicator

You can change the recommended values that are used to calculate the password strength by editing the code of the appropriate controls:

To set different values globally for the entire application, edit the code behind of the `~/CMSModules/Membership/FormControls/Passwords/PasswordStrength.ascx` control and enter different numbers into the `mPreferredLength` and `mPreferredNonAlphaNumChars` variables.

You can also override the values for specific instances where this control is used through its **PreferredLength** and **PreferredNonAlphaNumChars** properties (e.g. in the code of the *Registration form* web part).

The appearance of individual password strength status labels may be customized through CSS styles. Each one has a different class assigned, e.g. *PasswordStrengthNotAcceptable*.

i Password policy and strength in custom forms

When creating custom forms, you can easily add password fields that validate according to the specified policy and display password strength.

To do this, specify either the **Password strength** or **Password with confirmation** form control for the given field, which will automatically ensure the functionality described above.

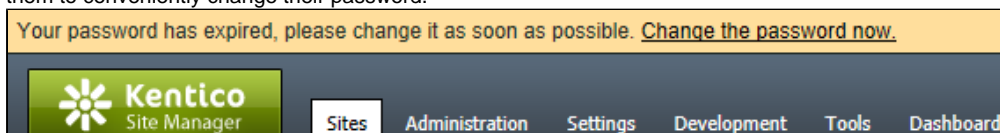
Password expiration

With the available password settings in **Site Manager -> Settings -> Security & Membership -> Passwords**, you can set the passwords to expire after a specified amount of time.

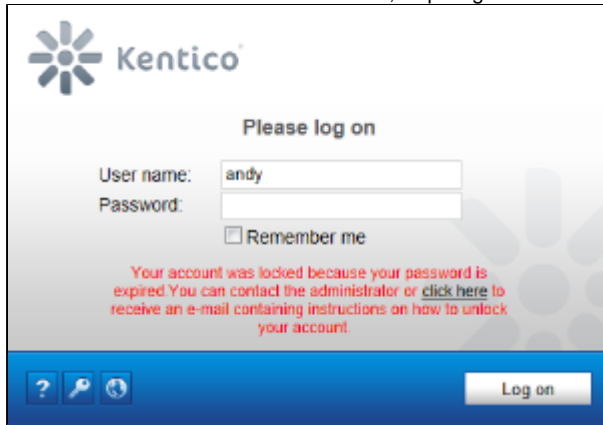
You can turn on password expiration with the **Enable password expiration** setting. When a user logs in to the system, the password expiration period (specified in days by the **Password expiration period** setting) is added to the time when the user last modified their password, and then compared with the current time. If the resulting time is earlier than the current time, the particular user's password has expired.

You can set how the system behaves after the password expires with the **Password expiration behavior** setting:

- **Show warning** - displays a warning text. The user can click the **Change the password now** link to open a dialog that will allow them to conveniently change their password.



- **Lock account** - locks the user's account, requiring the user to unlock their account and change their password.



i To display a friendly message (as you can see on the picture above) to the users, check the **Display account lock information message** option in **Site Manager -> Settings -> Security & Membership -> Protection**. If you do not check this option, the users will see only a general message without knowing that their account has been locked.

The system can warn the users that their password is about to expire. You can adjust the period during which users will be displayed with the warning via the **Password expiration warning period** setting.

Notifying live site users

By default, notifications related to password expiration are displayed only in the administration interface. To notify also live site users, place the **Password expiration** web part on a page.

Resetting a password

Users can change their expired password on a special page. You can either use the default page (*~/CMSModules/Membership/CMSPages/ResetPassword.aspx*), or specify a custom page in the **Reset password page URL** setting.

A custom password reset page should contain one of the following components:

- **Reset password web part** - a web part you can use in the Portal engine development model.
- **ResetPassword control** - an alternative to the Reset password web part, which can be placed on an ASPX page. The control is located in *~/CMSModules/Membership/Controls/ResetPassword.ascx*.

Notifying users by e-mail

By turning the **Send password expiration email** setting on, you can specify whether you want to notify users about the expiration of their password via email.

The **Site Manager -> Administration -> E-mail templates** section contains a predefined template (**Membership - Password expiration**) that is sent to users when their password expires. The template contains the *{% ResetPasswordUrl %}* macro, which is resolved to a link that points to the URL of the page that allows to change the user's password.

Extending password validity

To extend the validity of any user's password, edit the user in **CMS Desk or Site Manager -> Administration -> Users** and on the **General** tab, click **Extend validity**. The password's validity will be reset to the **Password expiration period** setting's value and the user will be enabled in case their account has been locked due to expired password.

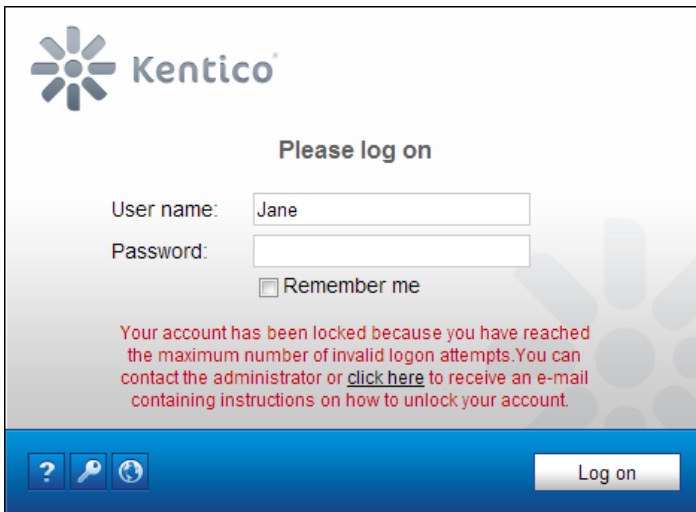
Created:	10/21/2008 4:37:52 PM
Last logon:	N/A
Last logon information:	
Invalid logon attempts:	0 attempt(s)
Password expires in:	4 day(s)
Starting alias path:	<input type="text"/>

Invalid logon attempts

One of the most common threats to website security is stealing user accounts. To compromise an account, attackers use a simple method, which tries to guess the password for that account, either by combining different characters, or by selecting passwords from a

dictionary.

This threat can be easily eliminated by introducing a **limit of invalid logon attempts**, which means that users will have their account locked after entering an incorrect password for the specified number of times.



The screenshot shows the Kentico login interface. At the top left is the Kentico logo. Below it, the text "Please log on" is centered. There are two input fields: "User name:" with the value "Jane" and "Password:". Below the password field is a checkbox labeled "Remember me". A red message states: "Your account has been locked because you have reached the maximum number of invalid logon attempts. You can contact the administrator or [click here](#) to receive an e-mail containing instructions on how to unlock your account." At the bottom right is a "Log on" button. On the bottom left, there are three small icons: a question mark, a person, and a refresh symbol.

To display a friendly message (as you can see on the picture above) to the users, check the **Display account lock information message** option in **Site Manager -> Settings -> Security & Membership -> Protection**. If you do not check this option, the users will see only a general message without knowing that their account has been locked.

Users cannot log in to a *locked* account. The global or site administrator has to unlock the account for them.



Using this protection may also lead to another security risk. If the users have easy-to-guess user names, then an attacker can block their accounts anytime by submitting wrong passwords with their user names on purpose.

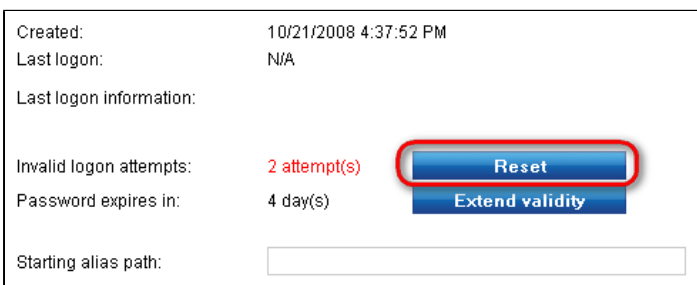
Limiting the number of invalid logon attempts

You can limit the number of allowed invalid logon attempts in **Settings -> Security & Membership -> Protection** in the **Invalid logon attempts** group, which contains the following settings:

- **Maximum invalid logon attempts** - specifies the number of attempts to log in that the user can try before the system locks their account and denies access. If set to zero, account locking will be disabled.
- **Send unlock account email** - indicates whether an email should be sent to the user if their account gets locked.
- **Unlock user account path** - allows selecting the path (or typing in the URL) of a custom page, on which the user can unlock their account.

Resetting the number of invalid logon attempts

When you edit a user in **Site Manager -> Administration -> Users**, you can view the number of invalid logon attempts the user made in the **Invalid logon attempts** field. To reset the number back to zero and unlock (enable) the user's account in case the user has reached the limit, click the **Reset** button.



The screenshot shows a user profile page with the following fields: "Created:" (10/21/2008 4:37:52 PM), "Last logon:" (N/A), "Last logon information:", "Invalid logon attempts:" (2 attempt(s)), "Password expires in:" (4 day(s)), and "Starting alias path:". A blue button labeled "Reset" is highlighted with a red border, and a "Extend validity" button is visible below it.

Forgotten password

If users forget their password, they may retrieve or reset it, provided they have access to the email address specified for their account. A password may be recovered by submitting a request through one of the website's logon forms.

By default, a forgotten password button is included on the logon page of the CMS Desk and Site Manager administration interface.

You can hide the button by adding the following key to the `/configuration/appSettings` section of your web.config file:

```
<add key="CMSShowForgottenPassLink" value="false" />
```

On the live site, users can recover their password through **Logon form** web parts that have their **Allow forgotten password retrieval** property enabled.

When submitting the request, users can either type in their user name or email address:

- If a user name is entered, the recovery email will be sent to the given account's address.
- If an address is used, the request will affect the password of the user account with the corresponding address.

Password recovery emails are sent from the address specified in the **Send password e-mails from** setting in **Site Manager -> Settings -> Security & Membership -> Passwords**.

Depending on the value of the **Reset password requires email approval** setting, one of two possible password recovery modes will be used:

Password reset without e-mail approval

If the **Reset password requires email approval** setting is *disabled*, then users who request their password will receive an email containing the password directly.

If the current password format is plain text, the existing password will be sent to the user. If an encrypted password format is used, the system will generate a new password for the user.

Password reset with e-mail approval

If the **Reset password requires email approval** setting is *enabled*, several steps will be added to the process.

i This option is **recommended**, as it is **more secure** than the previous option. When the *Reset password requires email approval* setting is *disabled*, then an attacker can easily lock other users' accounts by guessing their user names and using the forgotten password retrieval function.

Users who submit a password recovery request through a logon form will first receive an email containing a link to a page where they can manually set a new password. This option is **more secure**, because the password cannot be read from the email by potential attackers. Also, the reset link is only valid temporarily. The time period during which the link is valid can be specified in hours by the **Reset password interval** setting.

When users click the link in the email, they will be redirected to the default `~/CMSModules/Membership/CMSPages/ResetPassword.aspx` system page, where they will be able to enter a new password. The URL of the link contains a token in its query string that automatically identifies the user whose password should be changed. After someone visits the link, it becomes invalid and cannot be accessed again.

If you wish to use a custom page for this purpose, simply create a new page on the website and place the **Reset password** web part on it. This web part displays a form with the same functionality as described above for the **ResetPassword.aspx** system page. After you create the page, enter its URL into the **Reset password page URL** website setting, or into the same property of individual **Logon form** web parts.

If the **Send email with reset password** setting is enabled, users will receive another email containing their new password once they successfully reset it.



Recovering administrator password

If you happen to lose the password for your administrator account and cannot access the management interface, you can use one of the following techniques to recover:

- **Reset password via web.config key** - insert the following key to the **appSettings** section of your web.config:

```
<add key="CMSAdminEmergencyReset" value="<your username>;<your new password>;[true/false]" />
```

- The first and second parameter specifies your user name and your new password, delimited by a semicolon. The third parameter is optional and indicates whether you want to create a new user with global administrator rights.
- The key will be automatically deleted after you gain access to the user interface.
- **Clear password in database** - find your user record in the **CMS_User** table and clear the contents of the **UserPassword** column. Then sign in to the administration interface with a blank password and set a new password.

Password recovery email templates

The emails sent to users during the password retrieval process are based on [E-mail templates](#), which can be found in **Site manager -> Administration -> Email templates**. The following password-related templates are available:

- **Membership - Forgotten password** - sent to users when they use the password recovery feature and the *Reset password requires email approval* setting is disabled.
- **Membership - Change password request** - sent as a reply to password recovery requests if *Reset password requires email approval* is enabled.
- **Membership - Changed password** - sent to users if their password is changed by an administrator, either manually or by generating a new one.
- **Membership - Resend password** - used if the current password information is sent to a user from the administration interface (this can only be done if passwords are stored in plain text format).

These templates can be edited as needed, so you may fully customize the content of the emails. You can enter the following context macros to include dynamic values in their text:

- **{% UserName %}** - the name of the user's account. If you are using site prefixes for user names, all occurrences of this macro in email templates can have the prefix trimmed out with the following method: `{% TrimSitePrefix(UserName)%}`
- **{% Password %}** - the current (new) password of the given user.
- **{% LogonURL %}** - returns the URL of the page where the retrieval password request was submitted. Only available in the *Forgotten password* template.

The two macros below are available specifically in the **Change password request** template:

- **{% ResetPasswordURL %}** - resolves into the URL of the page where the user can change their password.
- **{% CancelURL %}** - returns the URL of a page that will cancel the request when opened. This can be used to create links that users can click in situations where someone else requested a new password for their user account (either intentionally or accidentally).

In addition to the special macros listed above, you can also use all other standard macro expressions in the templates. See the [Macro expressions](#) chapter for more information about macro expressions in Kentico CMS.

Unlocking an account

A user account can be locked for one of the following reasons:

- The user's [password expires](#).
- The user reaches the limit of [invalid logon attempts](#).

The following text describes how you can provide users with means to unlock their accounts.

Password expired

When an account is locked due to password expiration, the particular user will be asked to change their password in order to unlock their account.

Alternatively, you can extend the password's validity. You can find more information in the [Password expiration](#) topic.

Invalid logon attempts exceeded

When an account is locked due to exceeding the number of invalid logon attempts, the particular user will have to manually unlock their account. You can enable them to do that by directing them to the `~/CMSModules/Membership/CMSPages/UnlockUserAccount.aspx` page.

Alternatively, you can create a custom page for unlocking accounts, on which you can place one of the following components:

- **Unlock user account** web part - a web part you can use in the Portal engine development model.
- **UnlockUserAccount** control - an alternative to the Unlock user account web part, which can be placed on an ASPX page. The control is located in `~/CMSModules/Membership/Controls/UnlockUserAccount.ascx`.

Notifications

You can then specify whether you want users to receive an e-mail notifying them that their account has been locked in the **Send unlock account email** setting. The notification email uses the **Membership - User account locked** template. You can inset a link to the account unlock page with the `{% UnlockAccountUrl %}` macro.

Users can also be notified directly when logging in. To enable this option, set the **Display account lock information message** setting to true. However, enabling this feature is not recommended, since it can reveal to a potential attacker the fact, that they've managed to lock a user's account.

Custom password calculation

If you want to calculate the passwords in your own way, follow this procedure. You can find more information in the [Registering providers in App_Code](#).

1. Create a new class in the App_Code folder and set it to inherit from the `UserInfoProvider` class.
2. Override the `GetPasswordHashInternal` method.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using CMS.Helpers;
using CMS.Membership;

/// <summary>
/// Customized UserInfoProvider for password calculation.
/// </summary>
public class MyUserInfoProvider : UserInfoProvider
{
    protected override string GetPasswordHashInternal(string password,
string passwordFormat, string salt)
    {
        return SecurityHelper.GetSHA1Hash(password + "CustomSalt");
    }
}
```

3. Extend the `CMSModuleLoader` partial class inside the App_Code folder. You can either create a new code file for this purpose, or extend an existing definition of the class.
4. Create a new class inside the `CMSModuleLoader` that inherits from `CMSLoaderAttribute`.
5. Override the `Init` method inside the class:

```

/// <summary>
/// Summary description for CMSModuleLoader
/// </summary>
[CustomHandlerModuleLoader]
public partial class CMSModuleLoader
{
    /// <summary>
    /// Module registration
    /// </summary>
    private class CustomHandlerModuleLoader : CMSLoaderAttribute
    {
        /// <summary>
        /// Initializes the module
        /// </summary>
        public override void Init()
        {
            // Registers the 'MyUserInfoProvider' class as the UserInfoProvider.
            UserInfoProvider.ProviderObject = new MyUserInfoProvider();
        }
    }
}
}

```

The system now uses the customized provider (MyUserInfoProvider) instead of the default one.

Configuring permissions securely

Permissions allow you to **control access** to the particular sections of the Kentico administration interface by users. To learn how the whole system works, continue through the following sections:

- [User types](#)
- [Impersonation](#)
- [Permissions and UI elements](#)
- [Roles](#)
- [Memberships](#)
- [Access control lists](#)
- [Special permissions](#) - the special permissions include Edit ASCX code, Edit SQL code and Edit SQL queries.

The rule of thumb here is to **assign the least privileges possible**. You should only grant permissions to users who really need to perform the particular actions.

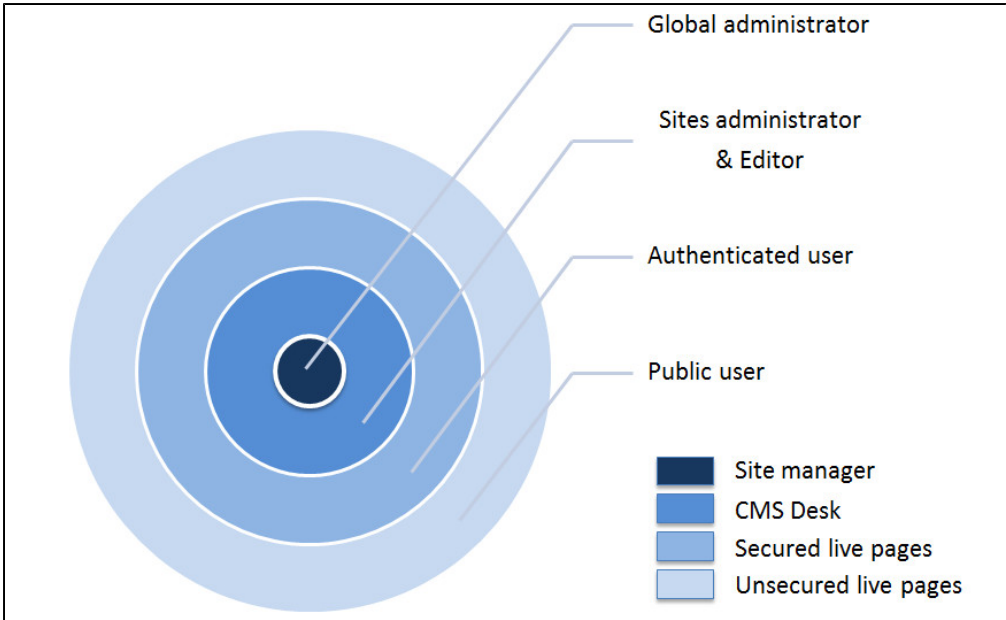
User types

Kentico CMS has three different user interfaces:

- **Live site** - front end for site visitors.
- **CMS Desk** - administration of one certain site. The place where content is edited.
- **Site manager** - administration of the whole CMS platform. The place where sites are managed.

Kentico CMS divides users on the following levels:

- **Public user** - has permissions to see live unsecured resources (pages, documents, images ...) on the live site, represents a site visitor who hasn't logged in.
- **Authenticated user** - a logged in site visitor, can see some secured resources on the live site (depending on assigned roles).
- **Editor** - has access to CMS Desk on the assigned sites.
- **Sites administrator** - has access to CMS Desk on all sites within the system. The site administrators can manage all objects of all sites but don't have access to Site Manager.
- **Global administrator** - has access to Site manager. User with all permissions.

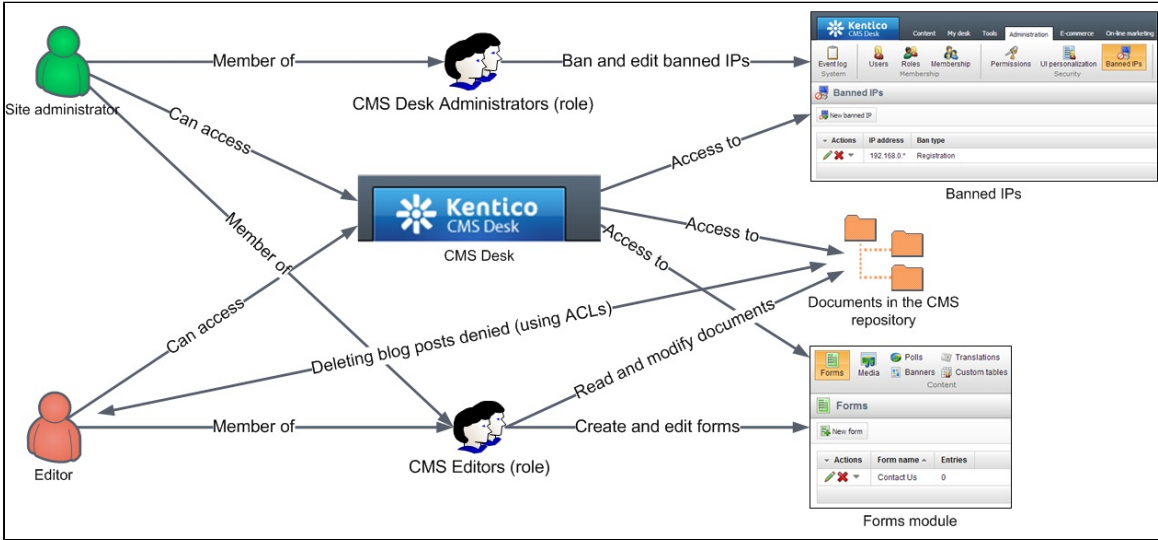


The Public user, Sites administrator and Global administrator user types have permissions determined by their user level. The other two types, Authenticated users and Editors, are more flexible and their permissions are determined by their roles.

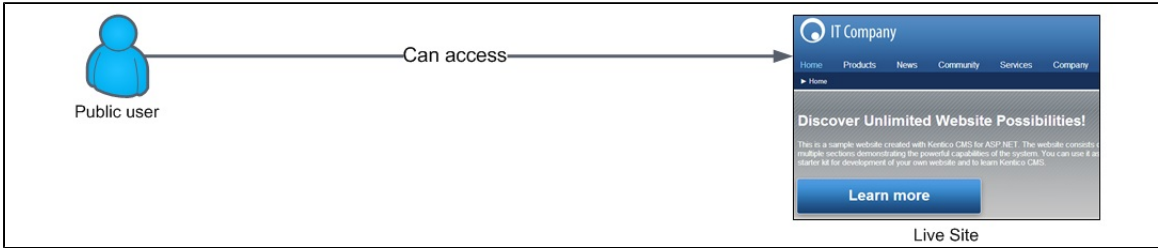
Global administrator is the only one who can access Site Manager.



Users who want to access CMS Desk (and see the content) must have the **Is editor** option checked in **Site Manager -> Administration -> Users -> edit a user -> General**. Moreover, they need to be assigned to roles with permissions configured according to these users' needs.



Public users (users who are not global administrators nor editors) can access only the Live site.



Impersonation

Global administrators can sign in to the system as other users. This allows them to view the user interface from the users' perspective.

Due to the security reasons, only the global administrators can impersonate other users. Additionally, it is not possible to impersonate other global administrators.

Permissions and UI elements

Permissions for the whole system can be managed in one place in the **Administration** interface. They are role based – you cannot assign specific permissions to a user directly, you always need to **assign the user to a role** and then give the role certain permission(s). There are two types of permissions:

- **Functional (permissions)** - permission check is done after the user performs an action. If the action is not permitted, an error message is shown in the interface.
- **Visual (UI elements)** - permission check is done during the page rendering. If a certain action is not available, the corresponding action button/link is not rendered and the user doesn't see it in the interface.

There are two standard permissions – read and modify (manage). Also, many modules have their own specific set of permissions for better granularity or for better handling of special scenarios. For example, the Users module has the special permission “Manage user roles” which allows a given role to add or remove a user from/to a role.



To allow roles to modify documents and other parts of the system, you need to assign them both the **read** and **manage** permissions.

If you assign only the **manage** permission to a role, then this role will not be allowed to view the specified pages.

There are also modules, for example the Forum module, where you can specify a special set of permissions directly in the module's configuration and even from the live site. It is assumed that these modules will be managed directly by Authenticated users who don't have access to the Administration interface (CMS Desk).



Be careful when assigning permissions, as some permissions can have other security implications. For example, you should assign the **Manage user roles** permission (from the Users module) only to a role with properly instructed users.

Roles

Each user can belong to any number of roles, their relationship is N:M. The roles are related N:1 to sites, every role belongs to a certain site.

You can learn how to manage roles in the [Role management](#) topic.

Memberships

Memberships group existing roles together, forming another security layer. Memberships are intended to be used mainly in the E-commerce module.

You can learn how to manage memberships in the [Managing memberships](#) topic.

Access control lists (ACL)

Every document (page) created in Kentico CMS has its own access control list (ACL). In this list you can specify which roles are permitted to read, modify, create, delete or destroy (delete permanently) the current document or its child documents.

You can learn how to work with ACLs in the [Document-level permissions](#) topic.

Special permissions

The special permissions include Edit ASCX code, Edit SQL code and Edit SQL queries and their settings can influence the possibility of privilege elevation attack. Find more information in the [Special permissions](#) topic.

Special permissions

The purpose of special permissions in Kentico CMS is to prevent the **privilege elevation** attack. In this type of attack, a lower privilege user can gain access to functions only available for higher privilege user. For example, CMS editors with permissions to edit content are able to elevate themselves to the global administrator level.

The special permissions cover three different areas:

- [ASCX code in layouts](#)
- [SQL code in web part properties](#)
- [SQL code in reports](#)



Only the global administrator can configure these settings.

Content editors and other **users cannot grant or remove these special permissions**. Even if you assign users to a role which is allowed to modify permissions, these special permissions will be disabled, which is indicated by a **yellow shield icon**:

Permissions				
Permissions for: <input type="text" value="Module"/>		<input type="text" value="Design"/>		
Report for user: <input type="text" value="(none)"/>		<input type="checkbox"/> Show only this user's roles		
Role	Wireframing	Design web site	Edit ASCX code	Edit SQL code
Authenticated users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMS Basic users	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

ASCX code in layouts

The **Edit ASCX code** permission for the **Design** module allows users to edit ASPX/ASCX code of page layouts and transformations.

Permissions				
Permissions for: <input type="text" value="Module"/>		<input type="text" value="Design"/>		
Report for user: <input type="text" value="(none)"/>		<input type="checkbox"/> Show only this user's roles		
Role	Wireframing	Design web site	Edit ASCX code	Edit SQL code
Authenticated users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMS Basic users	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMS Community administrators	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMS Designers	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CMS Desk Administrators	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

When you grant the Edit ASCX code permission to a certain role, then a user in that role can edit such code in layouts and transformations and can run any code in them – for example, the code that elevates privileges. Without the permission, the user cannot do this.

SQL code in web part properties

The **Edit SQL code** permission for the **Design** module allows users to edit **SQL WHERE** and **ORDER BY** properties in various web parts (e.g. the query repeater).

Permissions				
Permissions for: <input type="text" value="Module"/>		<input type="text" value="Design"/>		
Report for user: <input type="text" value="(none)"/>		<input type="checkbox"/> Show only this user's roles		
Role	Wireframing	Design web site	Edit ASCX code	Edit SQL code
Authenticated users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMS Basic users	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMS Community administrators	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMS Designers	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CMS Desk Administrators	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

If you grant this permission to a role, a user in that role can write any SQL statement into WHERE and ORDER BY. Without it, the user can only write simple WHERE conditions like "UserID = 1" and ORDER BY statements like "UserID".

SQL code in reports

In the **Reporting** module you can write any SQL statement in order to get data from the database into a chart. This could also be dangerous because a user can elevate privileges or get sensitive data from the database using the SQL code.

The **Edit SQL Queries** permission indicates whether the given role can edit SQL queries in reports.

Permissions					
Permissions for:	Module	Reporting			
Report for user:	(none)	<input type="checkbox"/> Show only this user's roles			
Role	Read	Save reports	Modify	Destroy	Edit SQL Queries
Authenticated users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMS Basic users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMS Community administrators	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMS Designers	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMS Desk Administrators	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Alternatively, you can change the connection string for reports. With that, you can create a special user account on the database level with limited permissions. For example, the database user will not be able to execute any **UPDATE/INSERT/DELETE** queries.

Another approach could be to separate the data into multiple databases. The first database would contain only reporting data, and the second database all other data including users. Then, the reporting module would be able to work only with the reporting data (and the user would not be able to execute malicious queries). Of course, in this scenario, you have to somehow deal with data synchronization.

Configuring multiple connection strings

1. Open web.config file and specify a new connection string in the **<connectionStrings>** section. The name attribute must be unique.
2. Go to **SiteManager -> Settings -> Security & Membership**.
3. Select the new connection string as the **Default report connection string** in Reporting category.

Reporting

Default report connection string ?

(default) ▼
(default)
CMSConnectionStringReporting

4. Set the **Set connection string** permission of the Reporting module.
 - Only a user in a granted role or the global administrator can change the connection string of a report.

Permissions							
Permissions for:	Module	Reporting					
Report for user:	(none)	<input type="checkbox"/> Show only this user's roles					
Role	Read	Save reports	Modify	Destroy	Edit SQL Queries	Subscribe	Set connection string
Authenticated users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMS Basic users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Allowing code editing for site administrators

In **Site Manager -> Settings -> Security & Membership**, notice the **Enable code editing for site administrators** option in the **Administration** section. Here, you can specify whether site administrators (global administrators without access to Site Manager) can edit ASCX, Reporting SQL queries and WHERE and ORDER BY in web parts.

Administration

Use SSL for administration interface ?

Automatically sign-in user when site changes ?

Enable code editing for site administrators ?

Conclusion

From a security standpoint, best practice is NOT to grant any of the permissions listed above to any roles. You should also not allow site administrators to edit code. If you do not follow this recommendation, you risk elevation of privileges.

Managing external authentication

External authentication services

Kentico supports several external authentication methods out of the box. To use them on your site, you have to configure them first (click on respective links for instructions) and place a corresponding web part on a page in your website.

- [Windows Active Directory](#)
- [Windows Active Directory - mixed mode](#)
- [Facebook](#)
- [Live ID](#)
- [OpenID](#)
- [LinkedIn](#)

Authenticating users against an external database

You can customize the authentication process to check the submitted user names and passwords against an external database outside of Kentico. For this purpose you can use the **SecurityEvents** class, which provides these events:

- **Authenticate** - fired upon user authentication.
- **AuthorizeResource** - fired upon security check for particular module permission.
- **AuthorizeClass** - fired upon security check for particular object type or document type permission.
- **AuthorizeUIElement** - fired upon permission check for particular UI element.

Example

Using the *Authenticate* event, you can customize the authentication process by extending the CMSModuleLoader partial class:

```

[CustomHandlerModuleLoader]
public partial class CMSModuleLoader
{
    private class SampleAuthenticationModuleLoader : CMSLoaderAttribute
    {
        public override void Init()
        {
            SecurityEvents.Authenticate.Execute += Authenticate_Execute;
        }

        private void Authenticate_Execute(object sender, AuthenticationEventArgs e)
        {
            // Check if the user was authenticated by the system
            if (e.User != null)
            {
                return;
            }
            UserInfo externalUser = null;
            string username = SqlHelperClass.GetSafeQueryString(e.UserName);
            string password = SqlHelperClass.GetSafeQueryString(e.Password);

            // Path to XML database file
            string xmlPath =
HttpContext.Current.Server.MapPath("~/userdatabase.xml");

            // Read data from external source
            DataSet dset = new DataSet();
            dset.ReadXml(xmlPath);

            // Sample external user credentials
            DataRow[] rows = dset.Tables[0].Select("UserName = '" + username + "'
AND Password='" + password + "'");

            // If external user was found, set the UserInfo parameter and
authenticate the user
            if (rows.Length > 0)
            {
                externalUser = new UserInfo();
                externalUser.IsExternal = true;
                externalUser.UserName = username;
                externalUser.FullName = username;
                externalUser.Enabled = true;
            }
            e.User = externalUser;
        }
    }
}

```

The crucial in this example is the *e.User* parameter, which represents the *UserInfo* parameter of the user being authenticated. Depending on its value, the authentication process can have these outcomes:

- If the user is found in the Kentico database, then the *UserInfo* parameter is not null and the user is authenticated normally.
- If the user is NOT found in the Kentico database (the *UserInfo* parameter is null), then the system looks into the external database. If the user is found there, the system sets the *UserInfo* parameter and authenticates this user.
- If the user is NOT found even in the external database, then the *UserInfo* parameter remains null and the user is not authenticated.

You can find more information about event handlers in the [Global events](#) topic.

Importing users into Kentico

For importing users from external databases into Kentico, you have these options:

- **Using external authentication**
 - When a user signs in to Kentico using an external authentication service (e.g., Facebook), the system creates their account in the Kentico database and imports their profiles. See the respective links at the beginning of this page.
- **Importing users using Kentico AD Import Utility**
 - Using the [Kentico AD Import Utility](#), you can import users and groups (roles in Kentico) from Active Directory into a Kentico database.
 - See the [Using the AD import wizard](#) page for instructions.
- **Importing users using the Kentico Import Toolkit**
 - The [Kentico Import Toolkit](#) to import can import various data from external sources. You can also use it to import users from one SQL database into Kentico.
 - See the [Import toolkit initial steps](#) page for instructions.

Securing the Staging and REST web services

Kentico offers two services which provide communication and synchronization of content and objects between servers. Both services are disabled by default. You should turn the services on only if you know you will be needing them.

- You can enable Staging at **Site Manager -> Settings -> Versioning & Synchronization -> Staging**.
- You can enable REST at **Site Manager -> Settings -> Integration -> REST**.

Staging

The weakest spot of the Staging service is in the authentication process. If potential attackers obtained the user name and password for the service, they could stage the administrator and gain absolute power over the system.

You can secure the staging service using two authentication options:

- **User name and password**
- **X.509 certificate**

The recommended option is to use the X.509 certificates for authentication, as certificates generally provide better security. See [Using X.509 authentication](#) for more details.

REST

The REST service provides access to the objects in Kentico, so a potential attacker could obtain any data from the system or modify them.

You can secure the REST service using these authentication options:

- **Basic authentication** - it is strongly recommended to **use SSL** with this type of authentication. See [Configuring SSL](#) for more details.
- **Forms authentication** - this is the standard ASP.NET authentication.

The recommended option here is to use the basic authentication with SSL.

You can also use the **Hash parameter authentication** for authenticating individual REST requests. You only need to generate the hash in the administration interface and add the hash to URL. This URL then serves a particular REST request without the need of further authentication. See [Authenticating REST requests](#) for more details.

The REST service should optimally check the correct authentication with every request. However, because of other services in Kentico (e.g., chat), which need some HTTP context within WCF, the checks are not performed every time. You can change this behavior by changing the `aspNetCompatibilityEnabled` key to **false** in the `<system.serviceModel>` section of the web.config file:

```
<serviceHostingEnvironment aspNetCompatibilityEnabled="false" />
```



Note that setting this key also **disables the chat** functionality.

Best practice

The best practice with REST is to **assign a dedicated user** to the service, grant the user **permissions only for the desired objects**, configure access through **SSL** and disable the `aspNetCompatibilityEnabled` mode.

Configuring e-mail confirmations

It is recommended to use all kinds of e-mail confirmations Kentico provides. The e-mail confirmations **protect the users** from being subscribed to mass e-mails or having their passwords changed without their knowledge.

Password change confirmation

You can allow the users to retrieve their passwords (or be assigned new passwords) if they forget them. It is a good practice to **require confirmation from the users** that they really want to change their passwords. Otherwise, if passwords were changed automatically after clicking the Forgotten password button, other users could abuse this feature to lock the users' accounts. Although the system would send the users their new passwords by e-mails and they would be able to log in with the new password, but this would be very annoying for them.

To require the users to confirm the password change, check the **Reset password requires email approval** option in **Site Manager -> Settings -> Security & Membership -> Passwords**.

You can learn more about forgotten passwords in the [Forgotten password](#) topic.

E-mail confirmation for newly registered users

It is recommended to require the users to confirm their registrations on your website via e-mail. This protects the users and their e-mails from identity thefts – it prevents other users from registering with someone else's e-mail and then act as somebody else.

To require the users to confirm their registrations, check the **Registration requires e-mail confirmation** option in **Site Manager -> Settings -> Membership & Security**.

You can find more information in the [Registration approval and double opt-in](#) topic.

Administrator's approval of newly registered users

You can configure, that after the users register on your website (and confirm the registration via e-mail), their accounts will not be activated immediately. The system will require the site administrator to confirm their registration. This is useful for protecting the system from being overwhelmed by fake users and spam bots and it also allows the administrator to verify the users' identities and the account types they created.

Using this feature will improve your website's security, but it can also significantly slow down the registration process and fend off potential users. It depends on the purpose of your website and on how important the true identities of users are.

You can find more information in the [Registration approval and double opt-in](#) topic.

Double opt-in

The double opt-in functionality, also referred to as confirmed opt-in or closed-loop opt-in, adds an additional security layer to user subscriptions. When users subscribe to receiving mass e-mails in some module, the system sends a confirmation message to their e-mail address first. Only after the users confirm the subscription by clicking the link included in the message, will the system add their addresses to the subscription mailing list.

Using this functionality is much recommended, as it protects the users from receiving large amounts of unsolicited e-mails without their knowledge. It eliminates the cases when someone submits for subscriptions incorrectly typed e-mail addresses or someone else's addresses out of malice.

You can enable double opt-in for these modules:

- Blogs - [Blog user subscriptions](#)
- Forums - [Forum subscriptions](#)
- Newsletters - [Newsletter double opt-in](#)
- Message boards - [Message boards user subscriptions](#)

PCI compliance

PCI DSS

The Payment Card Industry Data Security Standard (PCI DSS) is a set of requirements meant to ensure that companies involved in the process of card payment maintain a certain level of security to **protect the cardholder data**. It was designed by major card brands in response to the growing number of data security breaches and the resulting unlawful uses of this data.

PCI DSS in its current version (2.0) is defined as a set of twelve rules, which the involved entities must adhere to. The following table lists the requirements organized into logically related groups, called control objectives.

Control Objectives	PCI DSS Requirements
Build and Maintain a Secure Network	<ol style="list-style-type: none">1. Install and maintain a firewall configuration to protect cardholder data2. Do not use vendor-supplied defaults for system passwords and other security parameters
Protect Cardholder Data	<ol style="list-style-type: none">3. Protect stored cardholder data4. Encrypt transmission of cardholder data across open, public networks

Maintain a Vulnerability Management Program	5. Use and regularly update anti-virus software on all systems commonly affected by malware 6. Develop and maintain secure systems and applications
Implement Strong Access Control Measures	7. Restrict access to cardholder data by business need-to-know 8. Assign a unique ID to each person with computer access 9. Restrict physical access to cardholder data
Regularly Monitor and Test Networks	10. Track and monitor all access to network resources and cardholder data 11. Regularly test security systems and processes
Maintain an Information Security Policy	12. Maintain a policy that addresses information security

Who must comply?

PCI DSS is a mandatory standard which applies to all entities that take part in payment card processing. This includes **retailers, e-commerce sites, acquiring organizations, card issuers** and any other subject which accepts, transmits or stores cardholder information.

In other words, if you are a merchant and want to accept payment cards, you must comply with the standard.

PA DSS

Payment Application Data Security Standard enforces the security of software used to process, transmit and store **cardholder data**. Similarly to PCI DSS, it defines a list of requirements the applications have to comply with. The current version (2.0) of PA DSS poses the following requirements:

Requirements
1. Do not retain full magnetic stripe, card validation, code or value, or PIN block data.
2. Protect stored cardholder data.
3. Provide secure authentication features.
4. Log payment application activity.
5. Develop secure payment applications.
6. Protect wireless transmissions.
7. Test payment applications to address vulnerabilities.
8. Facilitate secure network implementation.
9. Cardholder data must never be stored on a server connected to the Internet.
10. Facilitate secure remote software updates.
11. Facilitate secure remote access to payment application.
12. Encrypt sensitive traffic over public networks.
13. Encrypt all non-console administrative access.
14. Maintain instructional documentation and training programs for customers, resellers, and integrators.

Who must comply?

PA DSS aims at **software developers** and **integrators** that deliver online payment applications, which are sold, distributed or licensed to third parties.

Relationship to PCI DSS

Both these standards ensure cardholder security, but at different levels. PA DSS is for software vendors, while PCI DSS is required for all merchants who handle cardholder information.

Although PA DSS is based on the PCI DSS requirements, **using PA DSS certified software does not make a merchant PCI DSS compliant**. The best way to mitigate payment card security threats is to implement PA DSS inside a PCI DSS compliant environment.

Kentico CMS compliance with PCI standards

Since PCI DSS is focused on merchants and the institutions that process card payments, **Kentico CMS doesn't need to comply with**

the standard. However, you, as a merchant, may decide to employ Kentico CMS, particularly its built-in e-commerce module, as means to run your business. You may also wish to provide customers with the possibility to pay with their cards. This would require you to obtain a **PCI DSS certification**.

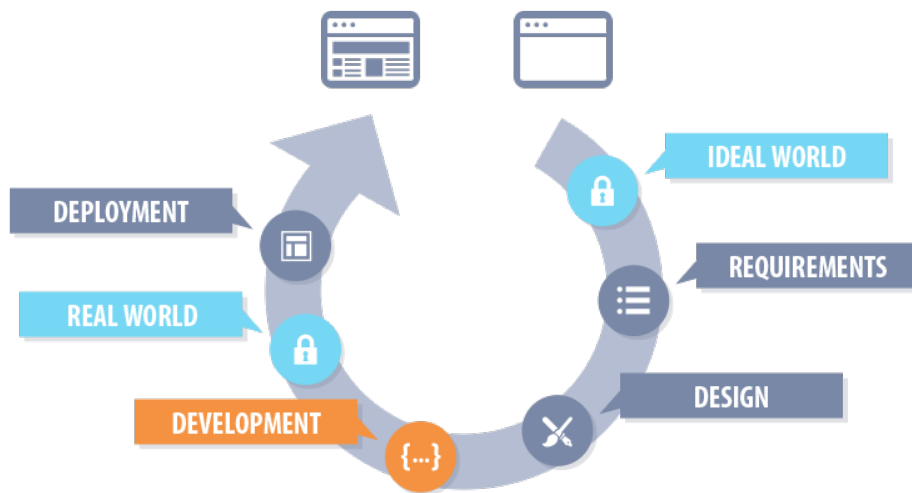
The PA DSS standard dictates that e-commerce solutions that offer online payment must be secured in order to protect cardholder data. Kentico CMS provides such an option. However, it is not a certified PA DSS compliant application. This means that **users of the E-commerce module would need to acquire the certification themselves.**

Despite the fact that Kentico CMS is not PA DSS certified, it is built in a way that doesn't prevent retailers from obtaining the required PCI DSS certification. The system doesn't store, transmit, or in any other way handle cardholder data, with the exception of a single feature – the Credit card payment method.

The built-in Credit card payment method uses the Authorize.NET payment gateway. However, when using this method of payment, customers do not enter their credit card numbers directly on the Authorize.NET website. Instead, they input the data on a page generated by Kentico CMS, which then passes the data to the Authorize.NET gateway. **This transfer is conducted over a secure protocol, hence it doesn't pose a security threat to sensitive data.**

To learn more about the standards discussed in this document and for information how to validate your compliance, visit the PCI Security Standards Council's website at <https://www.pcisecuritystandards.org>.

Developing secure web applications



Secure coding recommendations

In this section you will find information and recommendations on how to:

- Securely work with macros
- Protect query string parameters
- Securely handle error messages

Macros and security

Macros are an essential part of Kentico CMS. You can read more about them in the [Macro expressions](#) chapter. This topic focuses on the security mechanisms and best practices related to macros.

Signing macros

Whenever a user saves a complex macro expression, the system automatically adds a security signature. This signature contains the **user name** of the macro's author and a **hash** of the given expression.

You can recognize signed macro expressions by the **#** character, which the system automatically inserts before the closing **%}** parentheses when saving the text containing the macro:

- `{%CurrentSite.SiteID #%}`

A signed macro may look like this in the database:

- `{%CurrentSite.SiteID|(user)administrator|(hash)9056b7b76629a47a660c40cc2e5b0d92a13d0f9bce178847ca412c3a585552a1%}`



Hashing is omitted when submitting very simple macros (e.g., `{%FullName%}`). Macros are signed only when they contain an indexer or a dot.

The system checks the macro signature when accessing an object through a different object (for example the user settings through a user, which are two different objects) and evaluates, if the user (identified by the hash) has **permissions** to execute the macro.

Best practice is to submit macros signed by a user who has access to the given objects, but does not have access to anything else (least privilege). The problem is that there are **nested macros** – the result of one macro expression is a different macro expression. This way, a potential attacker could gain access to sensitive information using a macro, which is signed by an administrator and is therefore executed in the context of the administrator.

A simple example of a **nested macro**:

- You have this macro expression on a page: `{% QueryString.GetValue("test", "")#%}`,
- which calls a page with such query parameter: `http://localhost/Final70/Home.aspx?test={%CurrentUser.UserName%}`.

SQL injections

Macros can often become part of SQL queries, for example in *WhereCondition* and *OrderBy* fields of web part properties, which can lead to SQL injection vulnerability.

SQL injection protection in web part properties

Some web part properties are secured against SQL injection attacks, which may affect how macros are resolved in specific cases. By default, this is applied to macros entered into the **WhereCondition** and **OrderBy** web part properties.

If the macro returns a string value that contains single quote characters (`'`), they will be escaped and replaced by two single quotes (`''`). This may cause an SQL syntax error if you are using the macro to dynamically insert a part of a query, such as a **WHERE** clause.

It is possible to disable single quote escaping for a specific macro expression by adding the **handlesqlinjection** parameter and setting its value to *false*:

```
{% ... |(handlesqlinjection>false %}
```

To disable SQL escaping globally for all properties of a specific type of web part, edit its code behind file (e.g. `~/CMSWebParts/Viewers/Documents/cmsrepeater.ascx.cs` for the **Repeater** web part) and add the following line of code into the **SetupControl()** method:

```
this.SQLProperties = "";
```

The **SQLProperties** property is inherited from the **CMSAbstractWebPart** base class by all web parts, but you can override its value to set which properties the system protects.

If you wish to enable SQL escaping for additional web part properties, enter their names into the **SQLProperties** value separated by semicolons, for example:

```
this.SQLProperties = "wherecondition;orderby;sqlquery";
```



Disabling SQL protection may create security vulnerabilities if the macro resolves its value according to data that can be modified by the website's users, such as in the case of *QueryString* macros.

SQL injection protection outside of web part properties

If you are using macros outside of web part properties or you have disabled SQL escaping, then you need to use the **SQLEscape** method to handle SQL macros.

```
WHERE UserName LIKE '%{ SQLEscape(QueryString.GetValue("test", ""))#%}'
```

However, if you expect a different data type than string (for example an integer), you need to convert the macro to the correct data type. The `SQLEscape` method is useless in this case.

```
WHERE UserID = {% ToInt(QueryString.GetValue("test", ""), 0)#%}
```

Output encoding of macros

Macro encoding is an essential protection against XSS. You must encode output macros properly, unless you want to display some HTML code using the particular macro. There are several methods for encoding macros in Kentico:

- `{% UrlEncode(CurrentUser.FullName)%}` - macro is encoded into the query URL.
- `{% HTMLEncode(CurrentUser.FullName)%}` - macro is a part of standard page output text and text between HTML tags.
- `{% JSEncode(CurrentUser.FullName)%}` - macro is a part of JS code.
- `{% StripTags(CurrentUser.FullName)%}` - encodes macros, which are part of standard page output text and text between HTML tags, leaving the the tags out.

Writing custom macro methods

When you create a custom macro method, you have to ensure its security yourself. The macro engine cannot predict, which data are accessed in the method, and thus cannot secure the method.

Unsafe macros

Query macros, cookie macros and data macros (information from the database, e.g. user display name) and their equivalent properties can be potentially dangerous. When you use these macros, you have to secure them against SQL injection and XSS.



It is NOT possible to gain access to user passwords using macros.

Query string hashing

Query strings in the URLs are useful and important in many ways, for example, in passing various values between pages or in retrieving data from the database. In some cases though, an unauthorized user could obtain sensitive data or harm the system by entering a URL with **tampered query string parameters**.

To prevent tampering with the query string parameters, the system adds the result of a **hash function** at the end of each URL. Such URL can look like this:

```
http://localhost/Kentico70/CMSAdminControls/UI/UniSelector/SelectionDialog.aspx?SelectionMode=Multiple&hidElem=m_c_u_content_su_s_hiddenField&params=891991d5-2e75-45f6-afbd-7247d1d13a44&clientId=m_c_u_content_su_s&localize=1&hash=d41aa76091347291c3bc772aaa5dfd90751110e43c0543c6d580da4ca8de3b37
```

The part in bold is the hash added to the URL. If an attacker modifies the parameters and tries to submit such URL, the system will not accept it, because the query string parameters and hash would not match.

The hash function is calculated from the query string parameters using [SHA-2](#). The hashing is used in various parts of the system:

- Dialog boxes
- [Macro signatures](#)
- In the links for downloading files (e.g., `getamazonfile.aspx`, `getazurefile.aspx` or sometimes even with `getfile.aspx`) – to allow the users to download only the file specified in the original URL and nothing else.

Salt

The protection of query string parameters only using the hash function would not be enough because the attackers are able to compute the hash with modified parameters. For this reason, the system adds **salt** to the URL before hashing it.




Hash calculation:

URL parameters + salt -> all this is hashed using [SHA-2](#) and added to the original URL.

The salt is some secret string of characters, which the users do not have access to. In Kentico, the default salt added to the URL is the **connection string** stored in the `web.config` file. You can, however, configure your own custom salt using the `CMSHashStringSalt` key in the `web.config` file (for example as a randomly generated GUID):


```
<add key="CMSHashStringSalt" value="e68b9ad6-a461-4707-8e3e-ece73f03dd02" />
```


 If you have already stored some persistent information (links for downloading files, image links, macros) in the system and you change the salt calculation, then these links and **information may become broken**. You will have to re-save the content to create hashes with the new salt.

User specific hash

Sometimes it is useful to add a user-specific information to the hash. Either the user session ID is used for this purpose or the user's IP address (if there is no session ID). When some attackers manage to eavesdrop some URL and try to exploit it to gain sensitive data, they would not succeed, as their session ID would not match the hash.

The user specific hash is used mainly for non-persistent information, such as displaying dialog boxes. You can specify the user specific hash using the *userSpecific* parameter of the *ValidationHelper.GetHashString()* method.

 Note that if you use user specific hashing and save some content with it, users other than the one who saved the content will not be able to use it.

Custom salt

You can also add a custom string to the hash. This string can represent a salt, which is unique for specific situations (for a given control, page, etc.). In some cases, the attackers could be able to generate hash in one dialog box and use the same hash in a different dialog box with the same parameters. To prevent this, use the custom salt.

You can specify the custom salt using the *customSalt* parameter of the *ValidationHelper.GetHashString()* method or in a class attribute.

Custom hash validations

We recommend that you also use the hashing features in your custom dialog boxes. The following methods are available:

Hash calculation:

- *ValidationHelper.GetHashString()* - this method computes the SHA2 hash from the query string parameters, the hash salt and other values which you can specify using its parameters:
 - *userSpecific* - a Boolean parameter which indicates, if the system adds user specific information to the value being hashed.
 - *customSalt* - a custom string, which is added to the value being hashed.

Hash validation:

- *QueryHelper.ValidateHash()* - this method works directly with the query string and can exclude individual parameters.
- *ValidationHelper.ValidateHash()* - this method is more general and can be used in macro signatures.


You can validate hashes on pages individually:

```
if (!QueryHelper.ValidateHash("hash"))
{
    URLHelper.Redirect(ResolveUrl("~/CMSMessages/Error.aspx?title=" +
    ResHelper.GetString("imageeditor.badhashtitle") + "&text=" +
    ResHelper.GetString("imageeditor.badhashtext")));
}
```

Handling error messages securely

Displaying information to the users in error messages is an important issue which you should pay attention to. Revealing some pieces of information (for example stack trace or debug data) can pose a security risk to your site, while not providing enough information would not help the users at all. To ensure that the error messages in your system are handled properly, follow these procedures:

- [Designing secure error messages](#) - learn how to design error messages in a way that the potential attackers cannot gain exploitable knowledge about the system.
- [Creating custom error handling pages](#) - create your custom error pages, which will be displayed to the users instead of the default ones. This is an important step, as you need to unify the error messages throughout the system.

 It is a good practice to disable displaying debugging and tracing information in the error messages before going live with your website. See [Web.config file settings](#).

Designing secure error messages

When designing error messages, you should always consider the level of information revealed to the user. If you reveal too much information, the user may be overwhelmed and confused. Moreover, malicious users may exploit this information to gain detailed understanding of the system.

On the other hand, if you do not provide enough information for the user to understand the problem, seeing such error messages may be very frustrating for the user.


Information you should NOT include in the error messages:

- Stack trace
- Debug information


Information you should include in the error messages:

- What is the problem (generic description)
- What can the user do to fix the problem (suggestions)
- What can the user do to prevent this problem in the future

Wrong

 **System error**
Object reference not set to an instance of an object. Line 159: FormInfo fi = FormHelper.GetFormInfo(dci.ClassName, false);
[Click here to go back to the home page](#)

Correct

 **SQL Connection error**
The application could not connect to the database, please check the connection string in the web.config file and SQL server availability.
Original error:
The SqlConnectionString property has not been initialized.



Handled and unhandled errors

The error pages should be **consistent** throughout the whole system. Configuring different error pages for handled errors and unhandled errors (a page redirected by ASP.NET using the `<customErrors>` web.config key) can be a severe security risk.

You should have only one error page for both of these cases. Find more information in the [Creating custom error handling pages](#) topic.

Time based errors

The time needed for processing a page after encountering an error can be considerably different from the processing time in other cases. The attackers can use this difference to guess if their input has caused any problems in the system.

There is no general recommendation on how to solve this trouble. However, you can try to provide some malicious input yourself and observe how much time it takes the system to complete the request. This way, you can find weaknesses in the system.

Storing the debug and trace information in the event log

Instead of showing detailed information about the problem in the error message, store the debug data and stack trace into the event log.

Configuring the error messages

To configure the system to display custom error messages, modify the web.config file, as described in the [Web.config file settings](#) topic.

Disabling debugging and tracing

To disable debugging and tracing before going live, see [Web.config file settings](#).

Creating custom error handling pages


You can configure the system to display custom pages instead of standard error messages. Custom pages help reduce the inconvenience caused to visitors if they run into an error while browsing your website, and also improves the security of the site by hiding potentially sensitive internal data (such as code in stack traces). You can create custom pages for this purpose with any kind of content, such as an apology or additional instructions, and then configure the system to display the pages in the appropriate situations.

Adding custom Page not found error pages

The *Page not found* error (404 HTTP status code) is one of the most common problems encountered by visitors. Kentico CMS provides several features that allow you to conveniently set up a custom page as a response. For *page not found* errors, the error page can either be a physical *.aspx* file placed under the web project or a dedicated document created in a specific website's content tree.

To assign your custom page to a particular website (or globally):

1. Go to **Site Manager-> Settings -> Content**.
2. Enter the URL of the given page as the value of the **Page not found URL** setting, for example: `~/SpecialPages/PageNotFound.aspx`

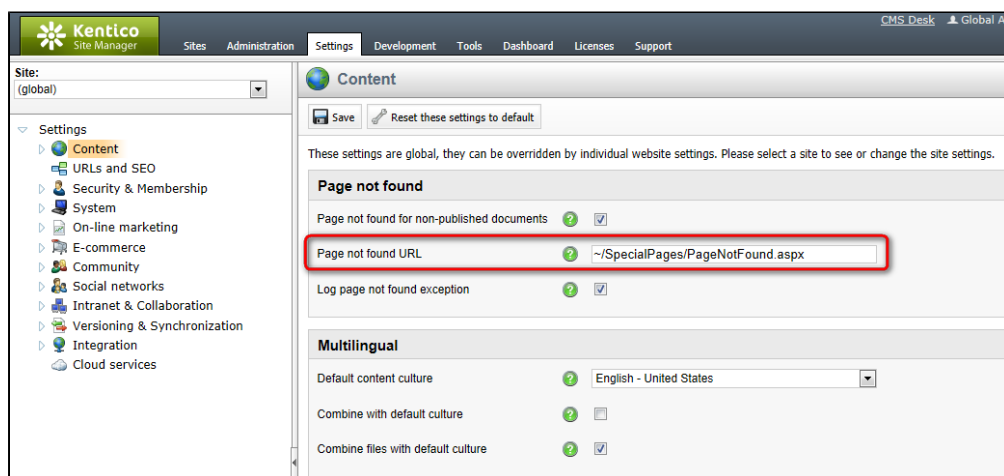
 Since there are two possible types of error pages, the system interprets the URL in two different ways. The sample URL value above specifies either:

- The URL of a physical page named *PageNotFound.aspx* located in the web project under a folder called *SpecialPages*.
- If such a file does not exist, the system attempts to select a Kentico CMS document under the current website, with an alias path equal to */SpecialPages/PageNotFound*.

3. Click  **Save**.

It is recommended to use Kentico CMS documents for page not found error pages. With this approach, you can define the error page's content using the portal engine and leverage all of its features. For instance, you can translate the page not found document on a multilingual website and the CMS automatically displays the culture version that matches the language selected by the user.

You do not need to manually handle the HTTP response code of the page specified by the setting. The page automatically returns a 404 status code when accessed (applies to both documents and physical pages). This allows applications, services and web crawlers to find out that a *page not found* error has occurred.



The screenshot shows the Kentico CMS administration interface. The left sidebar shows the 'Settings' menu with 'Content' selected. The main area displays the 'Content' settings for the 'global' site. Under the 'Page not found' section, the 'Page not found URL' field is highlighted with a red box and contains the value '~/SpecialPages/PageNotFound.aspx'. Other settings include 'Page not found for non-published documents' (checked), 'Log page not found exception' (checked), and 'Multilingual' settings (Default content culture: English - United States, Combine with default culture: unchecked, Combine files with default culture: checked).

Setting a custom Page not found page



Handling 404 errors for general content

To ensure that the system returns your custom *Page not found* error page for invalid requests that target all types of site content, not just the pages processed by the Kentico CMS engine, you need to fulfill several additional requirements.

- The site must be hosted on IIS version 7 or newer.
- The *Managed pipeline mode* of the application pool needs to be set to *Integrated*.

If your environment meets the conditions above:

1. Edit your application's **web.config** file.
2. Find the **system.webServer** section directly under the root (i.e. not under a specific `<location>` element).
3. Add the following attribute to the section's `<modules>` element:

```
<modules runAllManagedModulesForAllRequests="true">
```

Handling general errors

Kentico CMS is a standard ASP.NET application, so you can configure the handling of all types of errors and exceptions by adding the **<httpErrors>** element into the **<system.webServer>** section of your web.config file:

```
<system.webServer>
...
<httpErrors existingResponse="Auto" errorMode="Custom">
  <clear />
  <error statusCode="500"
path="/<Virtual_Directory>/CMSMessages/CustomError.aspx"
responseMode="ExecuteURL" />
</httpErrors>
...
</system.webServer>
```

The **errorMode** attribute sets the basic error page behavior. Use one of the following options:


- **Detailed** - in this mode, the default error pages display details of the encountered error to all users.
- **DetailedLocalOnly** - with this option, the default error pages show detailed error information only to the local host. Users who access the site remotely see simplified error pages.
- **Custom** - use this mode if you wish to replace the default error pages with a completely custom page.

When specifying custom error pages:

1. Set the **errorMode** to *Custom*.
2. Add any number of child **<error>** elements representing individual types of HTTP errors that you wish to handle.
 - You need to enter the HTTP response code of the given error into the **statusCode** attribute and the URL of the appropriate error page as the **path** value.
 - If your Kentico CMS application is running in a virtual directory, replace the *<Virtual_Directory>* string in the sample path value above with the directory's name.
 - It is recommended to add general error pages directly into your web project as *.aspx* files, e.g. under the **CMSMessages** folder. For pages placed in your web project, set the **responseMode** attribute of the corresponding **<error>** elements to *ExecuteURL*.
 - You can define the content of the error page to match your specific requirements.
 - Keep in mind that error pages should always return the appropriate *HTTPResponse* status code.

Before the deployment

Do not forget to set custom error messages as described in this procedure before deploying your website and going live. Also, remember to disable debugging and tracing for your ASP.NET application. See [Web.config file settings](#).

 **Note:** The approach described above works for applications running on IIS 7 or newer with an application pool using *Integrated* Managed pipeline mode. On older versions, you can instead edit the **<customErrors>** element under the *<system.web>* section of the web.config to achieve similar results.

Avoiding security vulnerabilities in code

In this section you will learn how to protect your system against various types of hacker and scripting attacks.

Cross site scripting (XSS)

Cross site scripting happens when somebody (an attacker) inserts a malicious input into a form (for example, a piece of HTML code). Depending on what happens after that, we divide XSS attacks into these types:

- **Persistent XSS** - a web application (like an instance of Kentico CMS) stores the malicious input in the database. Then, on some other page, the input is rendered. A browser renders the attacker's malicious input as a part of the page's HTML and this is the spot, where the input can cause problems.
- **Non-persistent XSS** - the main difference is that a web application doesn't store the malicious input in the database. Instead, the application renders the input directly as a part of the page's response.
 - A special case of non-persistent XSS is called **DOM-based XSS** - this type of attack is done without sending any

requests to the web server. The attacker injects JavaScript code directly.

Types of XSS attacks

Persistent XSS

In this example, we have a standard .aspx page with a textbox, a label and a button:

```
<asp:TextBox runat="server" ID="txtUserName"></asp:TextBox>
<asp:Button runat="server" ID="btnGetMeNameOfUser" Text="Get me name of user"
onclick="btnGetMeNameOfUser_Click" />
<asp:Label runat="server" ID="lblUser"></asp:Label>
```

In code behind, we handle the **OnClick** event of the button. In the handler, we get a `UserInfo` object from the database. Finally, we render the first name entered into the text box via the label.

```
protected void btnGetMeNameOfUser_Click(object sender, EventArgs e)
{
    UserInfo ui = UserInfoProvider.GetUserInfo(txtUserName.Text);
    if(ui != null)
    {
        lblUser.Text = ui.FirstName;
    }
}
```

Now there is a possibility of XSS attack. The attacker can simply create a user with a first name "`<script>alert(1)</script>`". Anyone who sends a request to a page where the attacker's first name would normally be displayed, gets the HTML code injected and executed on their machine (the user gets an alert showing "1").

Non-persistent XSS

Now we have another standard .aspx page with a label:

```
<asp:Label ID="lblInfo" runat="server" EnableViewState="false"
CssClass="InfoLabel" />
```

In code behind, we have this code:

```
protected void Page_Load(object sender, EventArgs e)
{
    // Get localization string from URL
    string restring = QueryHelper.GetString("info", "");
    lblInfo.Text = ResHelper.GetString(restring);
}
```

This is an example of a potential vulnerability to a non-persistent XSS attack. The attacker "creates" a URL like `www.ourweb.tld/ourpage.aspx?info=<script>alert(1)</script>`. This URL is sent to a victim. The victim clicks on it, and JavaScript code `alert(1)` gets executed.

DOM-based XSS

Let's have another .aspx page with this JavaScript code:

```
<select>
<script>
  document.write("<OPTION value=1>" + document.location.href.substring(
  document.location.href.indexOf("default=") + 8) + "</OPTION>");
  document.write("<OPTION value=2>English</OPTION>");
</script>
</select>
```

When a user navigates to `http://www.mydomain.tld/XSSDOMBased.aspx?default=<script>alert(document.cookie)</script>`, the JavaScript code is executed because it is written to the page by the `document.write()` function. The key difference here is that no part of code is handled by the server. The whole attack is done only in the victim's browser.

What can XSS attack do

In cross site scripting, the attacker can insert any kind of code into a page which is interpreted by the client browser. So, the attacker can, for example:

- change the look of your site,
- change the content of the site,
- insert any JavaScript code to your site,
- redirect the page to an evil one,
- force the users to download malicious code (typically a virus).

With JavaScript, the attacker can read and steal the user's cookies. With stolen cookies, the attacker can log on to a site (even with an administrator account) without a user name and password. This is the most dangerous thing the attacker can do when your site is XSS vulnerable.

You may think that any at least a little advanced user wouldn't click a link like:

```
blahblah.tld?input=<script>alert('I am going to steal your money');</script>
```

Yeah, but what about a link like:

```
blahblah.tld?input=%3d%3c%73%63%72%69%70%74%3e%6a%61%76%61%73%63%72%69%70%74%28%9
1%49%20%61%6d%20%67%6f%69%6e%64%20%74%6f%20%73%74%65%61%6c%20%79%6f%75%72%20%6d%6
f%6e%65%79%92%29%3b%3c%2f%73%63%72%69%70%74%3
```

Can you read that string? It is the same string as the first one, only in hexadecimal encoding, so the browser receives the same text.

Finding XSS vulnerabilities

The first way to find XSS vulnerabilities is based on trying. Simply insert a test string into all inputs and URL parameters. Use strings like:

- `<script>alert(1)</script>`
- `alert(1)`
- `'alert(1)`

See if your input was executed, if it changed the page or if it caused a JavaScript error. All these signs point to a page vulnerable to XSS. You can also try to change the HTTP request (for example, if you are using the Firefox browser, you can use add-ons to change the user agent, referrer, etc.).

Another way is to find vulnerabilities in code. You can search for all spots where properties (of Kentico CMS's objects) are used and check whether they are HTML encoded when they get rendered on the page. This is the best technique of searching for persistent XSS.

You can also search for spots where context variables (`HttpContext.XXX`, `HttpContext`, `Server`, `Request`, `request.querystring` - this one is the most important) or URL parameters (`QueryHelper.GetString()`) are used. This way, you can find non-persistent XSS. If you want to search for DOM-based XSS, search your code for the following JavaScript objects (these can be influenced by the attacker):

- `document.URL`
- `document.URLUnencoded`
- `document.location` (and many of its properties)
- `document.referrer`
- `window.location`

The last way is to use automatic tools for vulnerability searching. These tools are based on similar techniques as manual searching, while they work automatically. However, they often find far too many false positive vulnerabilities and using them is much less effective. The reason for that is simple – these tools (at least those that aren't based on formal verification – about 99% of them) use brute force, while

you can use your brain.

Avoiding XSS in Kentico CMS

Why would the attacker be able to perform all types of XSS attacks in the previously specified code examples?

Mainly because the dynamic parts of code (inputs, database data, ...) were not encoded properly. So, everything that goes to output (is rendered to HTML) must be properly HTML/JavaScript encoded. What does it mean to properly encode something? In Kentico CMS, we have these methods to avoid XSS attacks:

- **HTMLHelper.HtmlEncode()** – encodes HTML tags, replaces the < and > chars with their HTML entities.
- **QueryHelper.GetText()** – gets a HTML encoded string from the query string.
- **ScriptHelper.GetString()** – replaces special chars like an apostrophe (alt+39).
- Correctly keep the type, using for example **QueryHelper.GetInteger()**.
- Use transformation functions: **HtmlEncode()**, **StripTags()**.

Where should you protect your code?

Always on the output before rendering. The reason is that you should secure your web, but you do not want to change the users' input data. You also should not exclusively rely on input validation.

It is also a good practice not to let anyone read cookies, as cookies are usually the main target of XSS attackers. You should make it impossible to access cookies on clients by configuring cookies to be http only – see [Web.config file settings](#).



Server-side encoding cannot protect you from the DOM-based XSS type of attack. Kentico currently does not have any special client-side mechanism (e.g., a JavaScript function) for avoiding DOM-based XSS. There is only one strong recommendation: When you are working with functions/objects that can change output (e.g., *document.write()*, *document.location*, ...) in client-side code, do not use the input directly. Let the server encode the potentially malicious input first.

Examples of server-side protection from XSS vulnerabilities

If you have a string value which is taken from the database, you must encode it with **HTMLHelper.HtmlEncode()** before you set it to a control's property (e.g., label text). For example:

```
lblUser.Text = HTMLHelper.HtmlEncode(ui.FirstName);
```

If you have a string value taken from query string (even if it's called id) and you plan to render it to output (i.e. some information message), use **QueryHelper.GetText()** instead of **QueryHelper.GetString()**. For example:

```
string restring = QueryHelper.GetText("info", "");
```

If you are putting some JavaScript into your code, for example:

```
ltScript.Text = ScriptHelper.GetScript("alert(' + dynamic + ');");
```

Always encode your dynamic parts with **ScriptHelper.GetString()**, even if you have already HTML encoded them. In this case, the attacker does not have to insert any HTML tags (e.g., <script>) because you have already inserted them. The protected code should look like this:

```
ltScript.Text = ScriptHelper.GetScript(
    "alert(' + ScriptHelper.GetString(dynamic) + ');");
```

Protect your code against XSS when writing **transformations**. You can use the **HtmlEncode** method and validation methods. For example, you can encode the manufacturer name in product listings:

```
<%# IfEmpty(Eval("SKUManufacturerID"), "-",
    HTMLHelper.Encode(EcommerceFunctions.GetManufacturer(Eval("SKUManufacturerID"), "ManufacturerDisplayName").ToString())) %>
```

You can find other transformation examples in [Transformations](#) topic.

Summary

- Encode strings before they are rendered.
- Encode all strings from any external source (user input, database, context/environment, URL parameters, method parameters).
- Use **HTMLHelper.HtmlEncode()** to encode strings from any external source.
- For URL parameters, you can use **QueryHelper.GetText()** if that value goes directly to output (i.e. the value is not saved to the database, filesystem, etc.).
- Values from any external source rendered as a part of JavaScript code must be encoded with **ScriptHelper.GetString()**.
- In JavaScript code, never render anything from any external source directly – let the server encode these values.
- Configure cookies as http-only.

SQL injection

SQL injection is a well known web application vulnerability. The attacker's aim is to execute his own SQL code on the victim's database through a web application. How can the attacker do that? The attack is similar to XSS. The attacker inserts a special string into the web application via a form or a URL parameter. If that string is used as a dynamic part of an SQL query (e.g., a part of a WHERE condition) and not protected properly, the attacker can inject a query before it is executed.

We can divide vulnerabilities into two kinds – *classic* and *blind*:

- Classic SQL injection – the attacker can see the real error message from the SQL server.
- Blind SQL injection – the attacker sees only a general error page. These injections are harder to exploit because the attackers do not know how exactly they can inject the code. The attackers usually use enumeration in this type of attack. Then, according to the time based errors or the displayed error message can they determine which of their queries have passed and which did not.

Example of SQL injection

We have a simple web page with a textbox and a button which is used for searching users:

```
<asp:TextBox ID="txtUserName" runat="server"></asp:TextBox>
<asp:Button ID="btnSearch" runat="server" Text="Search" onclick="btnSearch_Click"
/>
```

In code behind, we have this code:

```
protected void btnSearch_Click(object sender, EventArgs e)
{
    DataSet ds = UserInfoProvider.GetUsers("UserName LIKE '%" + txtUserName.Text +
"%'", null);
    if (!DataHelper.DataSourceIsEmpty(ds))
    {
        Response.Write(ds.Tables[0].Rows[0][ "FullName" ] );
    }
}
```

Now if a user inserts something like "admin", the user gets the full name of the global administrator on the output. The SQL query looks like this:

```
SELECT * FROM CMS_User WHERE UserName LIKE '%admin%'
```

This is a correct query which doesn't cause any problems. But if the user inserts something like "a"; DROP table CMS_User --", the resulting query is:

```
SELECT * FROM CMS_User WHERE UserName LIKE 'a%'; DROP table CMS_User --%'
```

The query is executed, resulting in a deleted table.

What can SQL injection attack do

With an SQL injection vulnerability, the attacker can do exactly the same operations with the database as the web application itself. For example, the attacker can read all data or the database schema, change it, edit it, etc. Also, T-SQL supports the xp_cmdshell() function, which executes operating system commands. So, the attacker can basically manage the whole server.

Finding SQL injection vulnerabilities

The first technique is based on trying. Insert strings like:

- 'whatever – basic test,
- DROP,
- something,

to all inputs/URL parameters/whatever. Do not test only the apostrophe character (in the next chapter, you will see that you can exploit an application even without the apostrophe character).

The second way is to search for vulnerabilities in code. You can search for methods executing SQL queries. Then, you can check variables which are inputs of these methods. The aim of checking is searching for protection against SQL injection.

You can also use automatic tools. There are two types of them. The first is based on trying – it simply runs an application, tries to insert some payloads to output and checks the application's reaction. The second type is based on searching for patterns (via regular expressions) in code. These automatic tools are very inaccurate and we do not recommend using them.

Avoiding SQL injection in Kentico CMS

There are many ways of protection against SQL injection. In Kentico CMS we use SQL parameters and apostrophe escaping. Both these methods have advantages and disadvantages and we use them in different situations.

Kentico CMS has its architecture divided into layers. One of them is the data layer, which provides operations for manipulation with database data. This includes executing SQL queries. The **GeneralConnection.ExecuteQuery()** method is the most frequently used method for executing SQL queries in Kentico CMS. This method has many overloads. We will use the following overload:

```
ExecuteQuery(string queryName, QueryDataParameters parameters, string where,
string orderBy, int topN, string columns)
```

We recommend that you avoid making your own direct query calls and that you retrieve system objects using providers or our strongly typed collections.

SQL Parameters

The second parameter of **ExecuteQuery()** is a **QueryDataParameters** object. To fill this parameter, create an appropriate object, for example:

```
QueryDataParameters parameters = new QueryDataParameters();
parameters.Add("@param", param);
```

SQL server simply replaces **@param** with your value. An important fact is that the value is treated as literal. It means that even if your value contains a piece of SQL code, the SQL server doesn't execute it.

Parameters are almost 100% secure. But if you build a query, which is executed with the built-in **exec()** function, the parameters are processed the standard way (the query is executed with them, even if they contain malicious SQL code). This typical example shows how **not** to do it:

Wrong

```
CREATE PROCEDURE injection( @param varchar(30) )
AS
SET NOCOUNT ON
DECLARE @query VARCHAR(100)
SET @query = 'SELECT * FROM ' + @param
exec(@query)
GO
```

Parameters are used in Kentico CMS mainly in the INSERT, UPDATE and DELETE query types, and also in stored procedures.

Apostrophe escaping

The second protection technique is replacing the dangerous apostrophe character with an escape sequence of two apostrophes. In code, you often build WHERE conditions for SELECT queries. When a part of a condition is a dynamically obtained string (e.g., from the database, input by a user, etc.), you must enclose it with apostrophes and perform replacing.

In Kentico CMS, we have a dedicated method for this purpose – **SqlHelperClass.GetSafeQueryString()**. This method also protects the

database against DoS attacks.

This is a correct solution, but only for string values. Never use this method for other types than strings. For example:

Wrong

```
Guid guid = ... ;
string where = "SomeGUID = '" + guid.ToString().Replace("'", "'") + "'";
```

In this example, replacing is unnecessary because class Guid can only contain letters and numbers in a specific format. So, this code is secured without any protection.

A worse situation can occur when you believe that you are using non-string data types (for example int), but the variable you are actually using is a string:

Wrong

```
string id = ... ;
string where = "SomeID = " + id.Replace("'", "'');
```

There are two reasons why this code is wrong. First of all, you don't need the apostrophe character to drop a table. In the previous examples, you needed an apostrophe to inject a string constant inside an SQL command, but there are no enclosing apostrophes here. You may think that the attacker cannot use WHERE conditions because you escape apostrophes. And that's the second problem. In SQL, there is the Char() function which converts numeric values to their ASCII representations. And these ASCII letters can be concatenated, so the attacker can write anything into the query.

You can solve these situations by adding enclosing characters or by converting values to correct data types. Always use the second choice (the first one causes performance issues). A correct piece of code should look like this:

```
string id = ... ;
string where = "SomeID = " + ValidationHelper.GetInteger(id, 0) ;
```

You can also use the **QueryHelper.GetSQLSafeText()** method to get query strings for SQL.

Summary

- Protect dynamic parts in INSERT, UPDATE and DELETE queries with SQL parameters.
- Don't ever use the **exec()** function in your SQL code.
- When you build a SELECT query in code, all used strings taken from external sources must be protected with the **SqlHelperClass.GetSafeQueryString()** method or use SQL parameters.
- Always escape values from array(list, ...) when you are getting them and putting them into a string (typically in foreach loops).
- Never rely on JavaScript validation. JavaScript is executed on the client side so the attacker can disable validation.
- When you work with other than string types, always convert data types to that type or validate the value via regular expressions.

Argument injection

Argument injection is a type of attack based on tampering with input parameters of a page. This can enable attackers to see data which they normally cannot see or to modify data which they normally cannot modify, via the user interface.

Example of argument injection

In this example, we have a simple .aspx page which allows users to change their passwords:

```
<asp:TextBox runat="server" ID="txtPassword" >
</asp:TextBox><asp:Button runat="server" ID="btnConfirm"
onclick="btnConfirm_Click" Text="Change password" />
```

In code behind, we handle the **OnClick()** event of the button:

```
UserInfoProvider.SetPassword(QueryHelper.GetString("username", ""),
txtPassword.Text.Trim());
```

The standard way of using this page is that the user adds a link to this page on a user profile page with URL parameter "username" equal to the current user's user name. In this code, there are two security issues:

- The user can change the password without entering the original one. If the user forgets to log out of a computer in an Internet café, then anybody can change the user's password.
- The page is vulnerable to argument injection. Any site visitor can change a password of any user of the system just by typing the URL address of the page with an appropriate user name in the parameter.

What can argument injection attack do

Argument injection can usually be used to obtain various information. The attacker can, for example, read documents or view images belonging to different users and so on. The threat usually depends on the sensitivity of the information. But sometimes, you can read invoices or other kind of sensitive information. And the worst case is if you can change something. You could probably never change a user's password. But what if an application has a DeletePicture.aspx page which deletes a picture whose IDs is provided in a URL parameter?

Finding argument injection vulnerabilities

The problem with argument injection is that any input from an external source can cause it. And there is no exact way of finding these spots. You should examine every single input. However, there are some practices which may help you find the most vulnerable places:

- Check all inputs of pages in paths containing "CMSPages". These directories contain, among others, pages which send files to the client browser and in one of the URL parameters, there is usually a path or an identifier to a file.
- Check pages which work with IDs/names taken from query string or via a form field. Especially those that take user IDs or site IDs from query string. These values can usually be taken from context instead.

Avoiding argument injection in Kentico CMS

Let's get back to the provided example and consider the problem that anyone can specify any user name. We can solve this by:

- Changing the user name to an identifier which is harder to guess – a GUID.
- Taking the user name from CMSContext.
- Checking that the current user has permissions to change his password.

But what solution is the best? The ideal solution is to combine all of them. Every time you do an action (displaying a picture is an action too), you must check the user's permissions. Also, if you can take data from current context, do not take it from another external source. Data in the current context, for example the information about the current user, is always correct (users can not manipulate with them). And if you have to manipulate with non-context data, use GUIDs instead of names or simple IDs.

Summary

- Always check that users have sufficient permissions to perform an action (if it's possible).
- Do not use query string/form values if is not necessary. Instead, use CMS context values.
- If you have to use query string/form values, use GUIDs instead of IDs or names.
- Secure query string parameters with [hash code validation](#).
- Combine these rules.

Command injection

Code injection in ASP.NET is not a well known issue. It is because in ASP.NET, code files are not inserted one into another dynamically (like in PHP). Programmers can only register controls in the web.config file or on a page. But dynamic code injection in ASP.NET is still possible. The aim is to insert C# (or VB.NET, etc.) code that is executed directly.

The attacker can achieve this in the following situations:

- When you use the **ProcessStartInfo** class in your code and execute commands which are put together from external sources.
- When your virtual path provider is able to read files from different servers, and parameters are taken from an external source.
- When you load a control dynamically and the source of the control is loaded from an external source.

The attacker can also manage to insert a file with code into your application directory.

What can command injection attack do

Simply anything that can be achieved programmatically.

Finding command injection vulnerabilities in Kentico CMS

There is no direct procedure to find code injection, but here are some tips for discovering possibly vulnerable places in Kentico CMS:

- Search for **ProcessStartInfo** in source code and check its input parameters.
- Analyze the virtual path provider module and search for any possibility of getting a file which is not a regular Kentico CMS virtual file.
- Try to edit a transformation without administrator privileges.
- Search for usages of the **LoadControl()** method and check the input of the method.

Avoiding command injection

You will probably never have to deal with this issue because code injection only poses a threat in the special cases described at the beginning of this chapter. Nevertheless, the general recommendations are:

- Never load controls dynamically when their path is taken from an external source.
- Do not ever use **ProcessStartInfo** and other classes which execute commands or run .NET code.
- If you want to customize the virtual path provider or transformation management, be very careful.

Lightweighth Directory Access Protocol (LDAP) injection

LDAP is a protocol for accessing and modifying the directory services over TCP/IP. This protocol has many implementations. However, we will focus only on the Active Directory Lightweight Directory service in this topic, as it is used by Kentico.

The LDAP injection attacks are similar to SQL injection attacks in principle. The attacker tries to exploit a web application to construct a malicious LDAP statement. If the application does not sanitize the user input, the attacker may be able to execute various commands.

Examples of LDAP injections

Obtaining user information

Original query	http://www.example.com/people_search.aspx?name=Andy)(zone=public)
Malformed query	http://www.example.com/people_search.aspx?name=Andy)(zone=*)

Effect: Using the malformed query, the attacker can obtain all user information of the user Andy, not only the public information as in the original query.

Elevation of privileges

Original query	http://www.example.com/document.aspx?path=Documents&recursive=yes
Malformed query	http://www.example.com/document.aspx?path=Documents (level=*)&recursive=yes

Effect: Using the malformed query, the attacker can obtain all documents, not only those he/she has permissions for in the original query.

What can LDAP injection attack do

The attacker can, for example, obtain user logins and employee information, achieve privilege elevation, or modify the content inside the LDAP tree.

Finding LDAP injection vulnerabilities

- Black box testing – try to insert LDAP queries into **logon**, **forgotten password**, or **search user** fields. Also try to insert bogus LDAP queries into these fields (see **Examples of LDAP injections** section).
- Searching in code – try to find usage of LDAP API in the code (see [Directory Services in the .NET Framework](#) for information). Then, make sure that the input passed to the API is sanitized and validated correctly.

How to prevent LDAP injection in Kentico CMS

In Kentico, LDAP is used during the authentication process via forms. The input data is not authenticated against a database, but against Active Directory. Therefore, this spot in the system is vulnerable to LDAP injection attacks. Knowing that, we have made sure, that the input data is correctly validated to prevent these attacks.

However, if you plan to create a custom web part which operates with the AD data, be sure to check and secure the input before sending the LDAP queries to the server. Good techniques are:

- restrict user input using regular expressions - where a number is expected, validate the input for numbers; where a name is expected, validate for characters.
- escape special characters.

Escaping special characters

The required escape sequence depends on whether the user input is used to create the Distinguished Name or used as a part of the search filter.

Used in DN - escape using / is needed:

&	!		=	<	>	,	+	-	"	'	;
---	---	--	---	---	---	---	---	---	---	---	---

Used in filter - escape using {ASCII} is needed:

({28}
)	{29}
\	{5c}
*	{2a}
/	{2f}
Null	{0}

It is also a good practice to include '\ ' at the beginning of escaped character listings to prevent recursive replacements.

XPath injection

The principle of XPath injection is very similar to SQL injection. The goal of the attack is very similar too. The only difference between these attacks is that XPath injection uses an XML file for data storage instead of a database. One way to get data from an XML file is to use a special query language called XPath.

Example of XPath injection

Let's say that a developer stores authentication data in an XML file with the following structure:

```
...
<user>
  <name>UserName</UserName>
  <password>Password</password>
</user>
...
```

On authentication, the developer builds an Xpath expression this way:

```
string//user[name/text()='txtUserName.Text' and password/text()='txtPassword.Text ' ])
```

The txtUserName and txtPassword variables are standard ASPX textboxes. When the attacker inserts an expression with an apostrophe (') to one of the textboxes, the attacker terminates the string and is able to write his own XPath expression. The scenario is basically the same as with SQL injection.

What can XPath injection attack do

An attacker can get, modify or delete anything stored in the given XML file.

Finding XPath injection vulnerabilities

The first technique is based on trying. Try to insert strings like:

- 'whatever – basic test
- DROP
- Something

to all inputs/URL parameters/whatever. If you see any error related to classes which provide manipulation with XMLs in ASP.NET, you have probably found an XPath injection threat.

The second way is to search for vulnerabilities in code. You can search for the following strings:

- Xpath – many classes which work with XPath have the 'xpath' string in their name.

- **SelectSingleNode()** and **SelectNodes()** – methods used in Kentico CMS for getting data from XML files via XPath.

Avoiding XPath injection in Kentico CMS

We do not store sensitive data in XML files in Kentico CMS at the moment. We also do not have any code where an XPath expression is built as described in the provided example. But this may change in the future. You can avoid XPath injection by following these rules:

- Validate input from external sources before you put it into XPath expressions.
- For characters like ', <, >, etc., use replace entities. "" is a replace entity for an apostrophe.

Cross site request forgery (CSRF/XSRF)

A browser typically uses two ways of requesting web applications. It sends data via URL parameters where HTTP GET request is used and sends data via forms where HTTP POST is used. The application typically does some action, for example, inserts a new user into a table, deletes a forum post, etc. And here is a problem. The web application typically does not check if the requests are generated by the web application itself (user clicks a link or sends a filled form). An attacker can create a link for some action and send it to the user. The user then clicks the link and the action is performed without the user even noticing. This is called **Cross Site Request Forgery**.

So a user has to click an attacker's link or fill an attacker's form. Another condition is that the user must be logged on to a vulnerable web. These days, almost every application provides the "keep me logged in" functionality, so this condition is easily met.

ASP.NET complicates a successful attack because of ViewState. If ViewState is turned on, you cannot send tampered POST requests to an ASP.NET application because validation of ViewState fails. For this reason, many developers think that an ASP.NET application is bulletproof against CSRF. However, this is not exactly true:

- GET requests can still cause CSRF.
- ViewState can be generated outside an application if you do not use machine keys as keys for ViewState encoding.
- Even if you use machine keys to encrypt your ViewState, it is not 100% safe. ASP.NET does not take form values from **Request.Form** but from **Request.Params**. This is the reason why it is possible to perform a so-called **One click attack**. It is a special case of CSRF. An attacker simply sends ViewState and values of form fields via GET. The trick is that the attacker can use ViewState generated by ASP.NET after POST and change values of fields and the validation will still succeed.

Example of CSRF

Let's have a simple page without any content and this code in code behind:

```
if (!string.IsNullOrEmpty(Request.Form["UserID"]))
{
    DoSomeAction(Request.Form["UserID"]);
}
```

It does not matter what the DoSomeAction() method does. The important thing is that, in this case (if a machine key is not used to encode ViewState), the action is performed with the UserID specified in the UserID field. Anyone who is authorized and sends a form with this field can perform the action. And there is no way to check if the actual user really wants to do this action.

Let's have another page without content with this code behind:

```
if (CMSContext.CurrentUser.IsGlobalAdministrator)
{
    int userID = QueryHelper.GetInteger("UserID", 0);
    if (userID != 0)
    {
        Response.Write("I've just deleted a user with id: " + userID);
    }
}
else
{
    Response.Write("You don't have enough permissions to delete the user");
}
```

This code is similar to the previous example. The only difference is that now, UserID is taken from a query string (by the GET method).

The third example shows a one-click attack. Let's have a simple page with a textbox and a button. This code handles the Onclick action of the button:

```
protected void btnSend_Click(object sender, EventArgs e)
{
    Response.Write(txtUserID.Text);
}
```

Users typically insert a value into the txtUserID textbox and click the button. But the attacker can forge a link to the user:

http://site/Page.aspx?_VIEWSTATE=/wEPDwULLTE0MDM4MzYxMjNkZlI5PxpCoDI4Dt3C2LKzz8CnHkbd&txtUserID=<anything>&btnSend=Send&_EVENTVALIDATION=/wEWAwLdr4fPBgLT8dy8BQKFzrr8AbhBL27NfMMamif/pHIFUlo41HNI

The attacker can change the **<anything>** macro to anything else. ViewState is taken from the page that can be generated after postback on that page and the validation is successful.

What can CSRF attack do

It depends on the application and on the security of the web server. For example, if the application is poorly implemented and encoding of ViewState based on machine key is turned off on the server, then the attacker can do anything that the victim of the attack could normally do.

Finding CSRF vulnerabilities in Kentico CMS

If you find any page/control/etc., that does an action on GET request, there is a possibility of a CSRF vulnerability. For example, try to find the following and similar strings in your source code:

- QueryHelper.GetString("action")
- EnableViewState="false"
- EnableViewStateMac="false"

If you find any of these strings in the **<%@ page** directive, it means that a developer turned off ViewState validation (first case) or machine keys for ViewState encoding (second and third case).

The ViewState validation helps a lot to avoid POST CSRF, so globally, it must always be turned on. Also, if you find any page that does not inherit from a Kentico class, it means that there is a possibility of CSRF.

Of course, if a page doesn't have these features and the page does not do any actions, it also does not have to be protected from CSRF.

Avoiding CSRF

Even if your application encodes ViewState properly, a special case of CSRF, a **one click attack**, is still possible. The problem is simple – ViewState is the same for all users. However, you can specify a key corresponding to the current user by the **ViewStateUserKey** page property. The recommended value is a user's session ID. All CMS pages must inherit from **CMSAbstractPage** (the base page for all CMS pages). CMSAbstractPage page sets ViewStateUserKey to a unique value for every user and thus avoids one click attacks.

Because ASP.NET partially secures web applications from POST CSRF and we add the least needed functionality to protect all POST requests by default, use POST requests only for actions. To enable machine key encoding or ViewState validation, you do not need to perform any actions.

By default (on the level of the global web.config file), ASP.NET is set to:

```
<machineKey validationKey="AutoGenerate,IsolateApps"
decryptionKey="AutoGenerate,IsolateApps" validation="SHA1" decryption="Auto"
compatibilityMode="Framework20SP1" />
<pages buffer="true" enableSessionState="true" enableViewState="true"
enableViewStateMac="true" viewStateEncryptionMode="Auto">
```

It means that machineKey is generated automatically, ViewState is validated and encoding of ViewState based on machineKeys is also enabled. If a control requests it, ViewState is encrypted by SHA1.

Summary

- Do not use GET requests to perform actions, always use POST.
- Never turn off validation of ViewState on a page (key **EnableViewState**) globally.
- Never turn off encoding of ViewState based on machineKey (**EnableViewStateMac**) on a page globally.
- Do not set the **CMSUseViewStateUserKey** key to false (it is an internal key which can cause insertion of the current user's key to ViewState encoding).
- If you insert a new page, always make it inherit from some of the CMS pages.
- If you create a new CMS page class, check that your page directly or indirectly inherits from **CMSAbstractPage**.

Directory traversal

This type of attack is also known as path traversal. The main goal is to show content of a file or directory via an application. Applications read data from the file system in many cases. Paths to these files or directories are often taken from input. If a user's input is not handled carefully, users can read data from the root directory of the server's file system.

Example of directory traversal

Let's inspect the following code:

```
Response.Write("<strong>Absolute path e:\\: </strong><br />");
string[] dirs = Directory.GetDirectories("e:\\");
foreach (string dir in dirs)
{
    Response.Write(dir + "<br />");
}

Response.Write("<br /><strong>Read e:\\StorageHelper.cs with absolute path:
</strong><br />");

Response.Write(File.ReadAllText("e:\\StorageHelper.cs") + "<br />");

Response.Write("<br /><strong>Root path of C: taken by ../../../../: </strong><br
/>");
string[] dirs2 = Directory.GetDirectories("../../../../");
foreach (string dir in dirs2)
{
    Response.Write(dir + "<br />");
}

Response.Write("<br /><strong>Read c:\\StorageHelper.cs with ../../../../:
</strong><br />");

Response.Write(File.ReadAllText("../../../../StorageHelper.cs") + "<br />");

Response.Write("<br /><strong>Directory with application: taken by
Server.MapPath("~/../"): </strong><br />");
string[] dirs3 = Directory.GetDirectories(Server.MapPath("~/../"));
foreach (string dir in dirs3)
{
    Response.Write(dir + "<br />");
}

//This code throws an exception - but the security is poor here
Response.Write("Directory with application: taken by ../: <br/>");
string[] dirs4 = Directory.GetDirectories(Server.MapPath("../"));
foreach (string dir in dirs4)
{
    Response.Write(dir + "<br />");
}

Response.Write("<br /><strong>Directory with
application: taken by Server.MapPath(\"/\") + \"../: </strong><br />");
string[] dirs5 = Directory.GetDirectories(Server.MapPath("/") + "../");
foreach (string dir in dirs5)
{
    Response.Write(dir + "<br />");
}
```

The result of this code will look like this:

Absolute path e:\:
e:\\$RECYCLE.BIN

e:\ASP.NET_TEMP
e:\AzureSLN

Read e:\StorageHelper.cs with absolute path:

***** Some code 2 *****

Root path of C: taken by ../../../../:

../../../../\$Recycle.Bin
../../../../Backup
../../../../Boot

Read c:\StorageHelper.cs with ../../../../:

***** Some code StorageHelper.cs

Directory with application: taken by Server.MapPath("~/../"):

C:\inetpub\wwwroot\3855_10914
C:\inetpub\wwwroot\3889_32518
C:\inetpub\wwwroot\3968_07397

Directory with application: taken by Server.MapPath("/") + "../:

C:\inetpub\wwwroot\..\AdminScripts
C:\inetpub\wwwroot\..\custerr
C:\inetpub\wwwroot\..\history

These are listings from one particular hard disk. The exact paths or listings aren't important, the important thing is that you can simply read any file or list any directory. And you can either use a relative or an absolute path. In this case, it is even a more severe vulnerability called full path disclosure. This bug enables the attacker to see a full path in an error message. The attacker finds out your directory structure and can effectively exploit a directory traversal vulnerability.

What can directory traversal attack do

The first dangerous thing is that the attacker knows your directory structure. And if there are any sensitive information in files in your application directory, the attacker can simply download these files. A worse case is when your application enables the attacker to read files. The attacker can then read your configuration files (e.g., the connection string in web.config file) or the configuration files of the whole system (if the application is used by a highly privileged user).

Finding directory traversal vulnerabilities

Search for parts of your application where the application reads files and directories. Then try to change the input parameters to get content from a place outside of the application directory. In code, you can search for these strings:

- Server.MapPath
- FileStream
- StreamReader

Avoiding directory traversal

Validate user inputs, check for absolute paths for sequences of "../", and so on. Also, if you read a file, check if the file is within the application path. The application should never read data from a random path on a hard disk.

Enumeration

Enumeration, in terms of security, is a vulnerability, which enables a potential attacker to guess some hidden system information. There are many types of enumeration threats, but we will discuss only two of them in this topic:

- Username enumeration
- File enumeration

Username enumeration

The username enumeration is an activity in which an attacker tries to retrieve valid usernames from a web application. The web applications are mostly vulnerable to this type of attack on login pages, registration form pages or password reset pages.

If the system is vulnerable to the username enumeration attack, the attacker may be able to obtain a list of existing usernames in the system by submitting input (valid and invalid user names) and analyzing the server response (error messages). The attacker can then run a dictionary attack to further exploit the obtained information.

Real-world example of a username enumeration attack

There is one well-known username enumeration vulnerability related to previous versions of Apache web server. Some distributions contained a misconfiguration, which enabled potential attackers to identify existing usernames.

When an attacker submits an HTTP request for a possible user's home page, `http://www.example.com/~<username>`, the server responds differently depending on whether the username exists or not:

- If the username exists and does not have a homepage, the server responds with HTTP result code 403, and the server message "You don't have permission to access /~username on this server."
- If the username does not exist, the server responds with HTTP result code 404 and the message "The requested URL /~username was not found on this server."

Because the server responds differently in these cases, the potential attacker can test and enumerate existing usernames. The attacker can then exploit this data for further attacks on the server.

Example of a web application username enumeration vulnerability in web application

When you create a web part in your web application, which enables users to log in, do not reveal information about existing usernames:

Wrong
If the submitted username is incorrect:

```
Entered login does not exist.
```

If the submitted username exists, but the password is incorrect:

```
Entered password is incorrect.
```

This way, the attacker can learn, which usernames exist in the system and which do not.

How to prevent username enumeration attacks in Kentico CMS

The default configuration of Kentico is protected against this type of attack.

If you plan to create a custom login web part, be sure to **show only generic error messages**:

Correct
In both situations (username does not exist or the password is incorrect):

```
Authentication failed.
```

This way, the attacker cannot distinguish between valid and invalid usernames.

In registration web parts, you cannot completely eliminate this vulnerability, because you have to check (and tell the user) if the submitted username already exists. Therefore, **always include CAPTCHA** in these web parts to prevent automatic collecting of the data by scripts.

File enumeration

In this type of enumeration attack, the attacker tries to guess the file names on the server and manage to gain access to them.

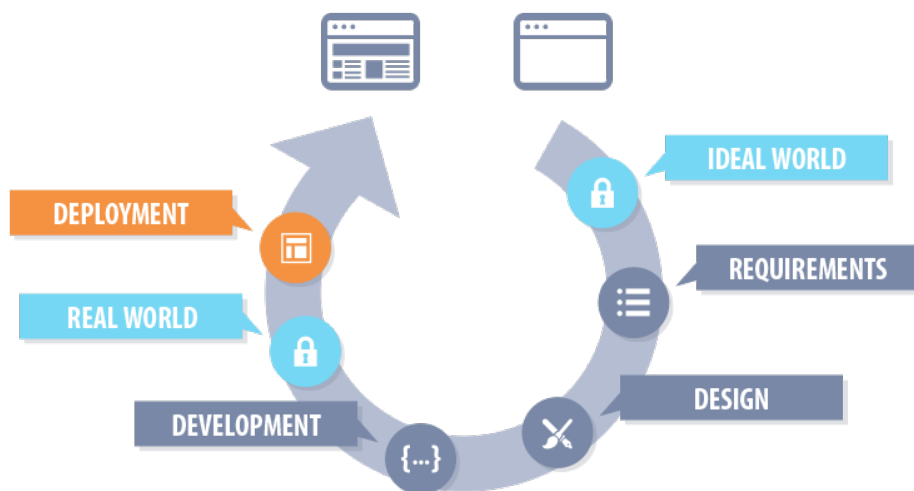
In Kentico, the attacker can for example try to guess the name of an export file. The export files are generally located in the `<web project>\CMSSiteUtils\Export` folder. The attacker can learn the structure of the file name and eventually guess an existing one (for example, by trying out different time stamps).

To protect your servers against this type of attack, **forbid access to sensitive directories** in the `web.config` file. See [Restricting access to directories](#).

Summary

- Do not reveal username and password details on your web pages (how long the username/password should be, which characters it should contain, etc.)
- Show only generic error messages (*Login failed. / The combination of username and password is invalid.*) in custom login web parts.
- Use CAPTCHA in registration forms.
- Forbid access from the outside to directories, whose names could be guessed by potential attackers.

Deploying web applications to a secure environment



Configuring SSL

The Secure Sockets Layer protocol is used to encrypt Internet communication. This is important for protecting privacy of your users and for safekeeping sensitive information that is being sent. If you do not protect the sensitive information on your website, you can become susceptible to man-in-the-middle attacks. Such attacks happen, when the attacker intercepts a communication between two systems (e.g., a server and a client web browser). The attacker can eavesdrop messages being sent and also possibly alter them without being detected.

How to start using SSL on your website

1. Obtain a certificate from a certificate authority or [create a self-signed certificate](#).
2. [Bind the certificate to your website through IIS server](#).
3. [Configure pages in Kentico CMS, which should redirect users to appropriate secure URLs](#).

What are SSL certificates

SSL certificates are electronic documents, which use a digital signature to bind a public key (needed to establish a secure connection) with a server. The digital signature is provided either by the certificate itself (a self-signed certificate) or by a Certificate Authority:

Self-signed certificates

- You can create them easily by yourself, but they are less secure than those signed by a Certificate Authority.
- Great for testing servers and suitable for collecting personal (non-financial) information.
- You should not use a self-signed certificate on an e-commerce site.

Certificates signed by a Certificate Authority

- Must be issued by a Certificate Authority, which verifies that the subject (server) of the certificate is who he claims to be.
- Can be expensive, but provide high security.
- It is generally recommended to use this type of certificate.

When should you use the SSL protocol

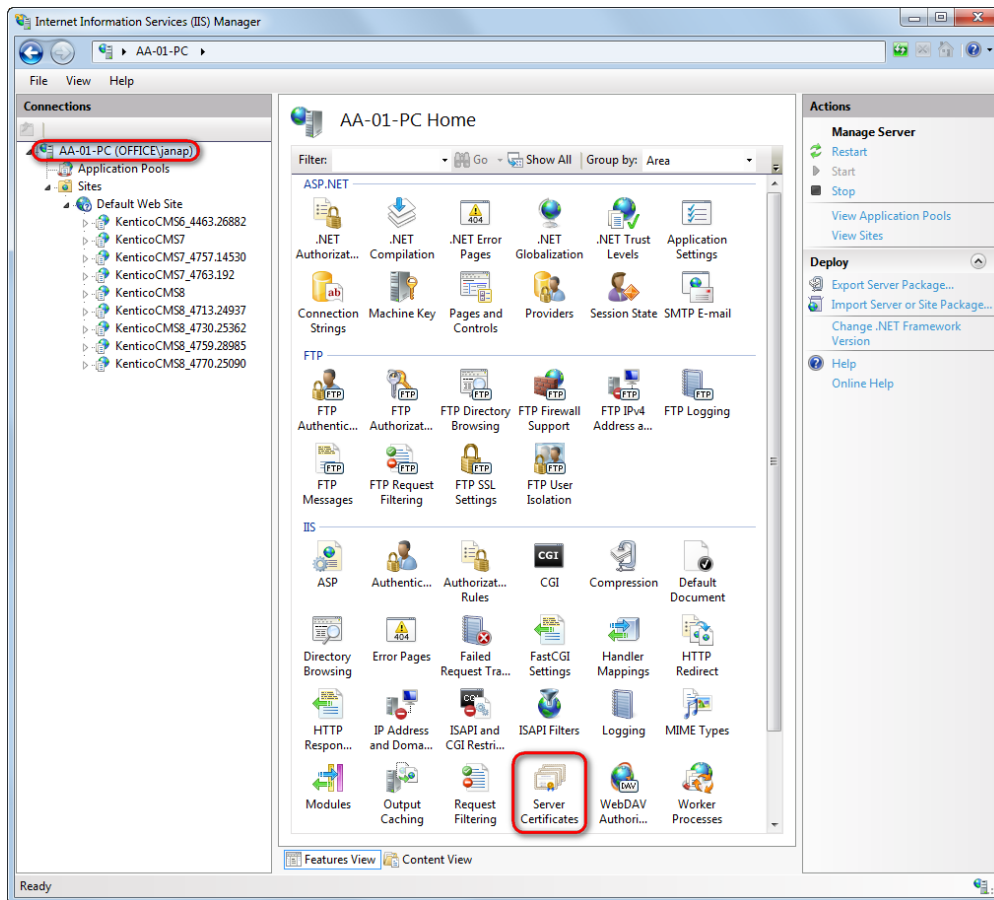
In general, every time you transmit any sensitive data. This includes cases:

- When your website contains login forms.
- When your website transmits personal information (e.g., social security numbers) and otherwise sensitive information (e.g., e-mails).
- Especially when your website collects and transmits credit card information (to prevent man-in-the-middle attacks).

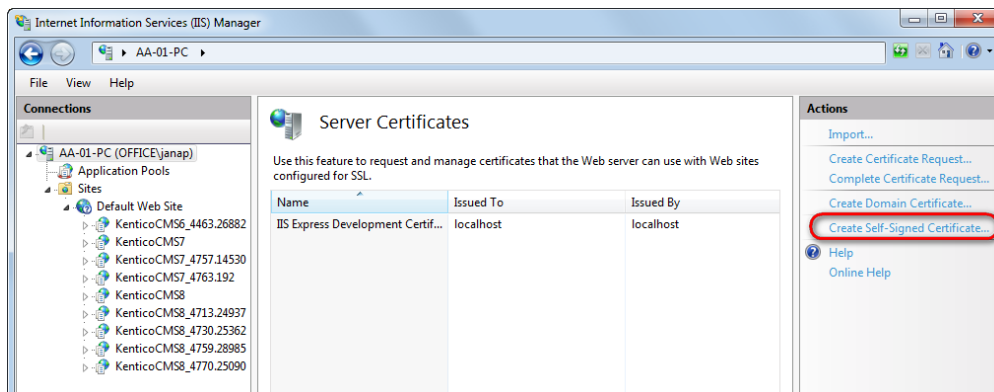
You do not need SSL if your website only forwards users to third party payment processors (like PayPal) for entering credit card information. However, you have to make sure, that users do not enter credit card information on your site.

Creating a self-signed certificate in IIS

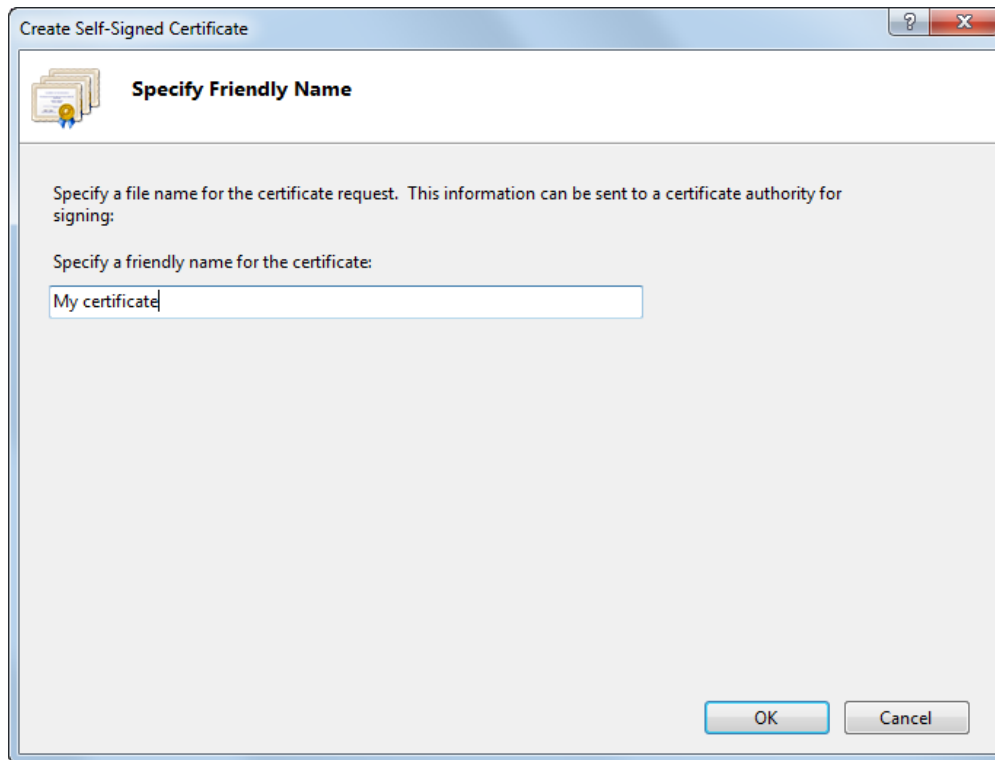
1. Open IIS Manager.
2. Click on the name of your server in the **Connections** column on the left.
3. Double-click the **Server certificates**.



4. In the **Actions** pane, click **Create Self-Signed Certificate**.



5. Type a friendly name for the certificate in the **Specify a friendly name for the certificate** box, and click **OK**.

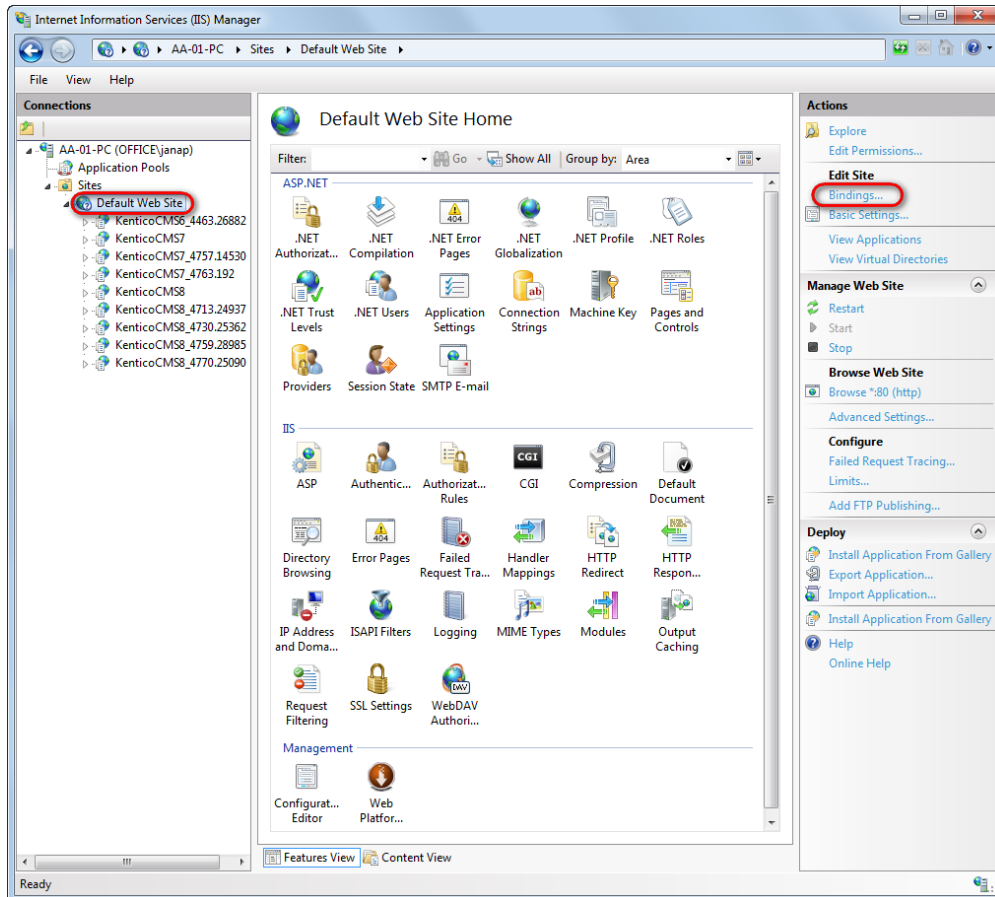


The IIS server creates a new self signed certificate with the name of your server as its common name.

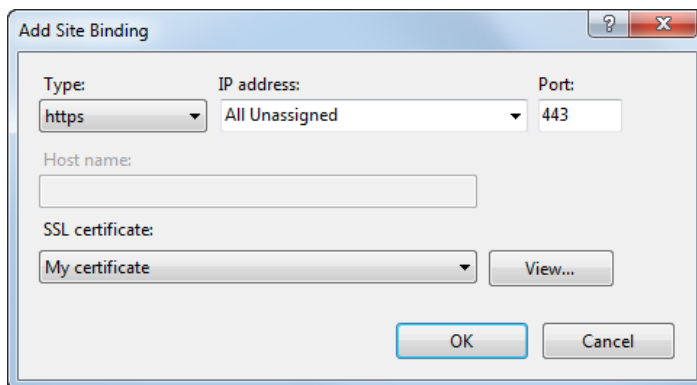
Binding a certificate to a website

If you already have a SSL certificate (self-signed or from a Certificate authority), you need to configure the IIS server so that it can use it for incoming secure connections.

1. Open IIS Manager.
2. Select a website, you want to bind the certificate to, in the **Connections** column on the left.
3. Click **Bindings...** in the **Actions** column.



4. Click **Add...**
5. Change the Type to **https**.
6. Select the certificate from the list.



7. Click **OK** and close the **Site bindings** dialog.

The IIS server is now ready for using the assigned certificate with SSL handshake for incoming connections.

Specifying pages in Kentico CMS, which should be accessible through the SSL protocol

Kentico CMS allows you to specify which of the website's pages should only be accessible over the secured SSL protocol. When users try to open such a page with the standard HTTP protocol, they will be automatically redirected to the secured version of the same page (using the **https** URL scheme).



When you set up a page to require SSL access, the system only redirects HTTP requests to the secure HTTPS protocol. Kentico CMS doesn't configure the website to use the HTTPS protocol. You still need to adjust your IIS settings to use SSL/HTTPS.

Securing individual web pages

1. Navigate to **CMS Desk -> Content**
2. Select a document in the content tree.
3. In **Properties -> Security -> Access -> Requires SSL**, select **Yes**.

Now all users attempting to access the page will always be redirected to the HTTPS version of the page's URL.

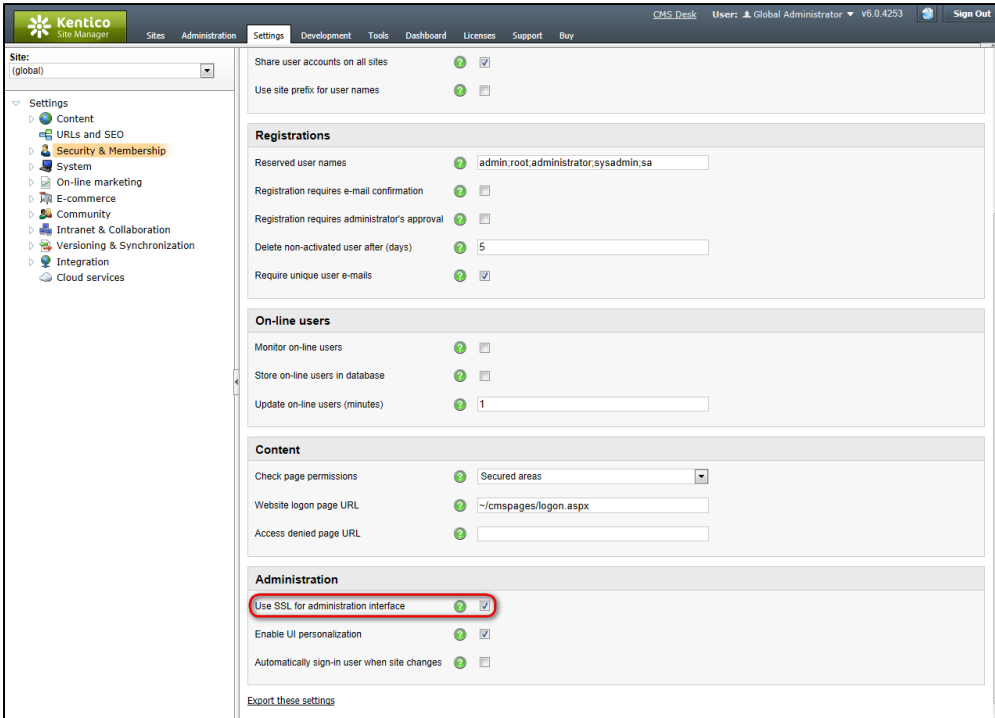
The screenshot shows the 'Properties' dialog box for a document, with the 'Security' tab selected in the left sidebar. The 'Access' section contains two radio button groups: 'Requires authentication' (with 'Inherits' selected) and 'Requires SSL' (with 'Yes' selected and highlighted by a red box). Below the 'Requires SSL' group are buttons for 'Add users', 'Select roles', and 'Remove'.

	Allow	Deny
Full control	<input type="checkbox"/>	<input type="checkbox"/>
Read	<input type="checkbox"/>	<input type="checkbox"/>
Modify	<input type="checkbox"/>	<input type="checkbox"/>
Create	<input type="checkbox"/>	<input type="checkbox"/>
Delete	<input type="checkbox"/>	<input type="checkbox"/>
Destroy	<input type="checkbox"/>	<input type="checkbox"/>
Browse tree	<input type="checkbox"/>	<input type="checkbox"/>
Modify permissions	<input type="checkbox"/>	<input type="checkbox"/>

Securing the administration interface

You may enable that all pages belonging to the administration interface (**CMS Desk** and **Site Manager**) will be accessible only through the SSL protocol.

1. Navigate to **Site Manager -> Settings -> Security & Membership**.
2. In the **Administration** category check the **Use SSL for administration interface** field.



Now all editors and administrators will be redirected to UI pages using the **https** URL scheme when they log in to CMS Desk or Site Manager.



This setting applies to all sites in the system. It is available only if you select the *(global)* option from the **Site** drop-down list.

SSL accelerator support

In some scenarios, SSL decryption and encryption may not be performed directly by your application's server. Instead, the decryption and encryption is performed via a reverse proxy, which is equipped with an SSL offload hardware (for example, an SSL accelerator). This means that requests are forwarded to the application internally using the standard HTTP protocol, even when the client accesses the page through HTTPS. If the settings for using SSL are enabled for the website, it may result in a redirection loop.

You can solve this issue by adding custom code to the application's request handlers. It is necessary to appropriately set the **IsSSL** static property of the **CMS.GlobalHelper.URLHelper** class. If set to *true*, the system will treat all requests as secured, regardless of their URL format, and redirection to HTTPS page versions will not be performed by the application. Of course, it is necessary to correctly identify which requests originally used SSL, e.g., by checking the request headers.

Setting the IsSSL property

1. Open your web project in Visual Studio, expand the **App_Code** folder (or **Old_App_Code** if you installed the project as a web application) and add a new class into it called **SSLRequestLoader.cs**.
2. Edit the class and add the following references:

```
using System.Collections.Specialized;  
  
using CMS.SettingsProvider;  
using CMS.GlobalHelper;
```

3. Extend the **CMSModuleLoader** partial class and define a new attribute for it:


```
[SSLRequestLoader]
public partial class CMSModuleLoader
{
    /// <summary>
    /// Module registration
    /// </summary>
    private class SSLRequestLoaderAttribute : CMSLoaderAttribute
    {
        ...
    }
}
```

4. Enter the following code into the **SSLRequestLoaderAttribute** class:

```
/// <summary>
/// Called automatically when the application starts
/// </summary>
public override void Init()
{
    // Assigns a handler which is called before each request is processed
    CMSRequestEvents.Begin.Before += HandleSSLRequests;
}

// Checks if requests are forwarded as SSL
private static void HandleSSLRequests(object sender, EventArgs e)
{
    if ((HttpContext.Current != null) && (HttpContext.Current.Request !=
null))
    {
        // Loads the request headers as a collection.
        NameValueCollection headers = HttpContext.Current.Request.Headers;

        // Gets the value from the X-Forwarded-Ssl header.
        string forwardedSSL = headers.Get("X-Forwarded-Ssl");

        URLHelper.IsSSL = false;

        // Checks if the original request used HTTPS.
        if (forwardedSSL == "On")
        {
            URLHelper.IsSSL = true;
        }
    }
}
```

In this example, we used the override of the **Init()** method (executed automatically when the application starts) to assign a handler that will be called before each request is processed. We also used the **X-Forwarded-Ssl** header to check if the original request was submitted via HTTPS before the SSL accelerator forwarded it to the application. If this is the case, the **IsSSL** property is set to *true* and the system processes the request as if it used the HTTPS protocol.

The proxy device may use a different method to identify requests that were originally secured by SSL. In such case, you will have to write a condition that fits your specific scenario (another typical approach is to check if the value of the **X-Forwarded-Proto** request header is *https*). You may also include additional custom code to fulfill any other security requirements, such as validation of the proxy's IP address.

Restricting access to directories

It is recommended that you allow users to access only those directories they actually need. This means that you must forbid access to the chosen directories for all users that do not need them. This can be configured in the web.config file. This example **forbids access** to the *CMSSiteUtils* directory **for all unauthorized users**:

```

<location path="CMSSiteUtils">
  <system.webServer>
    <security>
      <authorization>
        <remove users="*" roles="" verbs="" />
        <add accessType="Allow" users="" roles="" />
      </authorization>
    </security>
  </system.webServer>
</location>

```

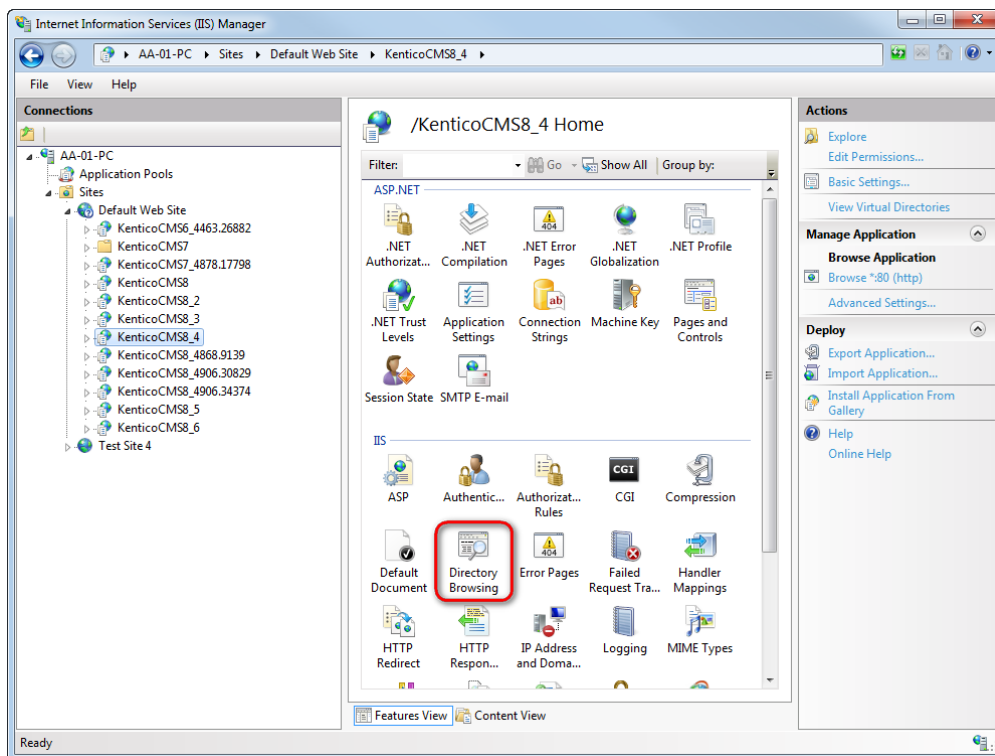
The *CMSSiteUtils* directory contains export files and is therefore the most vulnerable and must be protected properly.

 Learn how to restrict access to the CMSHelp directory in the [Restricting access to the CMSHelp directory](#) topic.

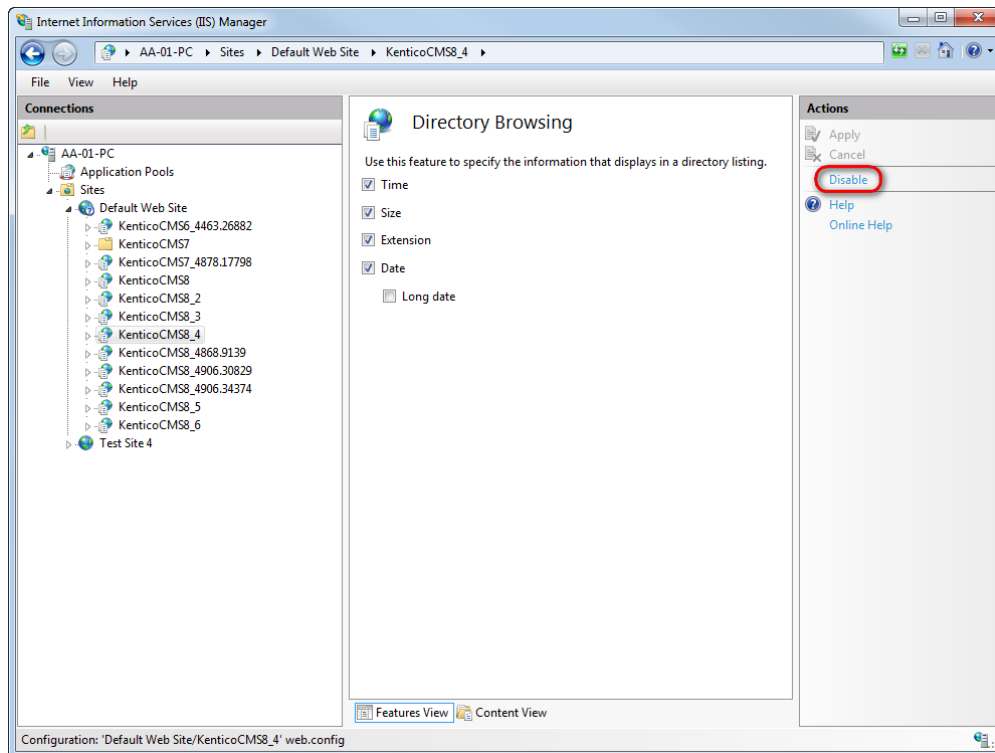
Disabling directory browsing

Another important security precaution is to disallow listing of files in directories. This can be set in the IIS (and should be already set as default configuration). It is recommended to disable directory listing for the whole website, although you can also disable this feature only for individual directories. In such case, do not forget to disable directory browsing for the *CMSSiteUtils* directory.

1. Open the IIS Manager.
2. Select the project for which you want to disable the listing of files.
3. Double-click the **Directory Browsing** icon in the IIS section.



4. Click **Disable**.



It is now not possible to list files in directories on your website.

Disabling unnecessary execution of scripts

You should forbid the execution of scripts where it is not required. This mainly applies to directories with multimedia or directories where you allow uploading of images. This can be set in the IIS, see [Edit Feature Permissions for the Handler Mappings Feature \(IIS 7\)](#) for instructions.

Keeping the web servers clean

The server where your web presentation is located should not contain any other unnecessary data. It is not wise to store any sensitive information there (e.g., database exports).

CDN and external storage

In Kentico, it is possible to store data to Azure blob storage and Amazon S3 storage. Both can be configured to allow public access so that anyone can download files, which were stored in Kentico.

The thing is that, to enable distribution of data over CDN, you need to **enable public access** to these data. This can pose a security risk, as you do not usually want everyone to be able to download all files from these storages. Therefore, you can set only certain containers (Azure blob) and buckets (Amazon S3) to be publicly available.

You can find more information in the [Configuring storage providers](#) topic.

Restricting access to the CMSHelp directory

Kentico CMS comes with an online help reference that is available in most parts of the administration interface. Users can view it to learn context-specific information about the current section of the application's interface. By default, any users (including public) can open the HTML content of the online help by entering the appropriate URL. This behavior may not be desirable in certain scenarios, e.g. in the case of high-security websites or if you are creating a rebranded solution.

There are several ways to solve this issue. The simplest is to delete the `~/CMSHelp` folder from the project of your production website. This removes the possibility of public users opening the help files, but the online help in the Kentico CMS administration interface will no longer be available to the users.

If you wish to keep the online help on your live website, you can limit access to the content of the help directory so that only users with the appropriate authorization are allowed to view it. Follow these steps to perform the required configuration:

1. Edit your application's **web.config** file.
2. Find the `<system.webServer>` section directly under the `web.config` root (i.e. not under a specific `<location>` element).
3. Configure the application to handle the requests for the HTML help files:
 - a. One option is to add the `runAllManagedModulesForAllRequests` attribute to the `<modules>` element:

```

<system.webServer>
  ...
  <modules runAllManagedModulesForAllRequests="true">
    ...
  </modules>
  ...
</system.webServer>

```

Setting this attribute to *true* ensures that the CMS application processes all types of requests and requires authentication if needed.

- b. If you do not want the application to process all additional request types, only **.html** and **.htm**, add the following two handlers into the *<handlers>* element:

```

<handlers>
  ...
  <add name="HTMLRequestHandler" path="*.html" verb="*"
modules="IsapiModule"
scriptProcessor="C:\Windows\Microsoft.NET\Framework64\v4.0.30319\aspnet_isapi.dll" resourceType="Unspecified" preCondition="" />
  <add name="HTMRequestHandler" path="*.htm" verb="*"
modules="IsapiModule"
scriptProcessor="C:\Windows\Microsoft.NET\Framework64\v4.0.30319\aspnet_isapi.dll" resourceType="Unspecified" preCondition="" />
  ...
</handlers>

```

Adjust the path in the **scriptProcessor** attribute as necessary according to your specific .NET environment.

4. Define the authorization rules applied to the content of the **CMSHelp** directory by adding the following section into your **web.config** file:

```

<location path="CMSHelp">
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</location>

```

This example only allows authenticated users to access the online help files. Public users cannot reach the files through a direct URL without being prompted to log in. To further increase the security, you can restrict access only for a specific set of roles by editing the *<authorization>* section:

```

<authorization>
  <allow roles="GlobalAdmin, CMSDeskAdmin" />
  <deny users="*" />
</authorization>

```

This ensures that only users who belong to the given roles (specified by their code names) have access to the directory.

Disabling unnecessary modules and services and keeping the system up-to-date

You should enable only those services, which your application needs. Otherwise, you provide more opportunities for attackers to infiltrate your system. Many services are installed by default, so you should take care to disable those you do not actually need.

Server security

If you run your applications locally on your own servers, then you should check which services run on your server and IIS. Then turn off everything your application does not need. You should also patch your operating system and server regularly. When a serious security issue is announced, you should patch your system as soon as possible, because the attackers are usually able to exploit the flaws within 24 hours.

If your applications run on remote servers (webhosting, cloud, etc.), all you can do is trust your provider to ensure the server security.

Kentico security

We recommend that you install only necessary modules (or that you uninstall unused modules after the installation). You can choose which modules will be installed with Kentico in the [Custom installation](#), and you can also add or remove modules and components after the installation – see [Adding and removing components from an installed Kentico web project](#).

You should also restrict public access to unused files located in `/CMSPages` and `/CMSModules/<some module>/CMSPages` directories. The following example restricts the public access for the `GetCMSVersion.aspx` page:

```
<location path="CMSPages/GetCMSVersion.aspx">
  <system.webServer>
    <security>
      <authorization>
        <remove users="public" roles="" verbs="" />
        <add accessType="Allow" users="*" roles="" />
      </authorization>
    </security>
  </system.webServer>
</location>
```

Hotfixing

We recommend installing hotfixes only when you need them – in cases when the hotfix repairs bugs that are causing you problems.

You should also know that we do not publicly announce our security issues. If we did, we would make it easier for the attackers to determine the issue and attack servers, that are not updated yet.

If you desire to be informed about security issues in Kentico, sign up to our security newsletter on the [client portal](#) (this newsletter is available only if you have prepaid maintenance service).

Hiding the system information

You should always try to hide the information about the server and operating system you are using. When the attackers are not able to determine this information, it is much more difficult for them to find flaws in the system and exploit them.

If attackers discover a flaw in a certain technology (IIS, ASP.NET, or Kentico), they could utilize this flaw to attack a large number of web servers with the same configuration.

Fingerprinting

Fingerprinting are techniques, that allow attackers to learn the exact version of web servers by querying the servers and analyzing their responses. Since the different versions of web servers have different implementations, they respond to special queries in different ways.

The same applies for content management systems. By analyzing the input code and the files located on the server, the attacker can figure out the type of CMS running on the server and its version.

The problem is, that you can never completely conceal the details about your system.

Server banners

The servers send greeting messages, called banners, with information about the server versions and used technologies. The servers send these messages in HTTP headers (in respond to fingerprinting queries) and you can also find them in page footers of directory listings.

Best practice is to hide as much information as you can. See the procedures on this page: [Configuring HTTP Response Headers in IIS 7](#).

Information about Kentico

Unfortunately, it is not possible to completely hide the fact that the server uses ASP.NET framework and the application running on it is Kentico. All you can do at the moment is:

- Forbid access to CMS Desk and Site Manager. You can set this by adding the *CMSThisDisableAdministrationInterface* key in the **<appSettings>** section of web.config file:

```
<add key="CMSThisDisableAdministrationInterface" value="true"/>
```

- Not to display information that the website was built using Kentico in the page footers.

Minimal secure configuration

Your web application should be run with the smallest set of rights that allow the application to function correctly. For example, a web application should NOT have access anywhere outside of the web application space. When attackers happen to find a flaw in the application, they will not at least gain access to sensitive information stored on the server (e.g., the SAM database, where the user passwords are stored).

You should explicitly ensure that:

- The IIS is run under a custom low-rights user account.
- The IIS application pool is run under a correctly configured user account. It is recommended to create a custom user account with minimum rights. See [Specify an Identity for an Application Pool \(IIS 7\)](#).
- The SQL Server Services are run under separate low-rights Windows or Local user accounts. See [Configure Windows Service Accounts and Permissions](#).
- Unsafe SQL functions, like `xp_cmdshell()`, are not turned on.
- The application runs in the most restricted trust level. Kentico can also run in the medium trust environment – it depends on the demands of your application.

Minimal Kentico requirements

This list provides the minimal configurations for the SQL and IIS user accounts in order to work properly with Kentico CMS.

Minimal configuration for an SQL user account

For **browsing the web**, this account must be:

- granted with permissions **connect, insert, select, execute, update, delete**
OR
- added to the database-level role name **db_owner**.

For **creating a database**, this account must be:

- granted with permissions **connect, insert, select, execute, update, delete, alter, references**
OR
- added to the database-level role **db_owner**.

If you want to limit the permissions for an SQL user, you have to first create a new SQL login in the SQL Management Studio, map it to the database and then assign the permissions for this login. For information about creating logins in SQL Management Studio, see [Create a Login](#).

Minimal configuration for an IIS user account to be able to use Kentico:

- this account must be granted with **Read, Write, Modify** permissions for the website directory.

Web.config file settings

In this topic, you will find a list of the most important 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 a protection against XSS (for example prevents attackers from reading the session ID from cookies or the forms authentication ticket from the authentication cookie).
- **requireSSL** – sets that the cookies require SSL connection, which prevents communication eavesdropping.

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

Session

You should use cookies instead of query strings for passing the session ID. This prevents session stealing, as it is easier to stole 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](#) in [Creating custom error handling pages](#) 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 disable the user key encoding using the **CMSUseViewStateUserKey** key (but we do not recommend it).

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 the [Cross site request forgery \(CSRF/XSRF\)](#) topic.

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.

Security checklists

In this section you can find lists of tasks, which we recommend you to perform in the given stages of development.

Security checklist – designing a website

This is a design checklist – facts you should consider before you begin developing your website.

Security requirements

Check	Description
	I know how critical the application safety will be (whether it is a blog, corporate website, e-shop, bank application, etc.).
	I know if my application will need any special certificates (PCI, Safe Harbor, etc.).
	I know which special requirements will be imposed on the application (custom authentication, premium sections, various types of administrators, etc.).
	I have an idea about the number of users accessing the system, which roles will the users be grouped under, which sections of the website will be accessible only to authenticated users, and so on.
	I know how large the scope of planned custom development will be.
	I know if security issues will be addressed during the development phase (possibly with the threat modeling) or after the application has been implemented.

Environment

Check	Description
	I know what environment will I deploy my application to (private server, web hosting or cloud).
	I know the security restrictions of the live environment (full trust/medium trust, etc.).
	I know what settings will I have access to in the live environment (which IIS settings).

Kentico

Check	Description
	I have mapped my security requirements to the Kentico system (for example, if you want to apply password policy, then you know Kentico ensures this and if the solution suits you).
	I am familiar with all Kentico system protections and I know how to utilize them.
	I know which modules and services will my application need and which I can uninstall or disable.
	I know how to use Kentico API securely.
	I have designed all custom authorization and authentication protections and I know how to implement them in Kentico.

Security checklist – developing a website

This is a design checklist – things you should keep in mind while developing websites.

User inputs

Check	Description
	User inputs are checked for type, length and content.
	User inputs with arithmetic operations are checked and validated for minimum and maximum values.
	All user inputs are validated on server side as well as on client side.
	Values stored in hidden files are validated properly.

Attack preventions

Check	Description
Cross-site scripting	
	User inputs are escaped and validated.
	Content is encoded before it is rendered on a page.
	Strings from external sources are encoded using the <code>HTMLHelper.HtmlEncode()</code> method.
	URL parameters are sanitized using the <code>QueryHelper.GetText()</code> method.
	Values from external sources rendered as part of JavaScript code are encoded using <code>ScriptHelper.GetString()</code> .
	Cookies are configured as http-only.
SQL injection	
	For dynamic parts of the SELECT, INSERT, UPDATE and DELETE queries are used SQL parameters.
	The <code>exec()</code> function is not used in the SQL code.
Cross-site request forgery	
	Actions are not performed using GET requests but using POST.
	View state mac validation is enabled globally in the web.config file.
	<pre><pages enableViewStateMac="true" /></pre>
LDAP injection	
	User inputs for LDAP queries are sanitized before execution.

Other issues

Check	Description
	User accounts are secured against all types of attacks.
	Error messages in the UI are configured so that they show only basic information and the whole information is logged only into the Event log.
File upload	
	Name, length, type and content of files is checked upon file upload.
Logging	
	All critical activities within the application are logged.
	The web application does not allow unhandled exceptions to occur.

Security checklist – deploying a website

This is a security deployment checklist – things to do before you deploy your site to a live environment.

Web.config:

Check	Description	Details
	The debug mode is turned off to prevent sensitive information leakage.	Web.config file settings
	Tracing is disabled to prevent sensitive information leakage.	Web.config file settings
	The error messages of web applications and application-server default error messages are not displayed in details to users.	Designing secure error messages
	Sensitive sections of the web.config file are encrypted (mainly the connection string).	How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI

	Access to sensitive directories is forbidden to protect the servers against the enumeration attack.	Enumeration
	Cookieless authentication is disabled to prevent session hijacking. This can be done by changing the cookieless attribute of the form element.	Session protection
	The <i>HttpOnlyCookies</i> flag is set so that the cookies are accessible only from the server-side code (this behavior is set by default in KenticoCMS).	Web.config file settings

IIS:

Check	Description	Details
	Directory listing is disabled in the application and web servers.	Export/import package security
	All HTTP methods except GET and POST are disabled if they are not in use.	Securing the Staging and REST web services
	Scripts and 3rd party libraries are up-to-date. If external libraries (e.g. for database access, XML parsing) are used, always use the current versions.	
	Sensitive links which should not be indexed by search engines are listed within robots.txt files.	Managing robots.txt
	The execution of scripts is disabled on folders where it is undesirable.	Edit Feature Permissions for the Handler Mappings Feature (IIS 7)

Kentico CMS:

Check	Description	Details
	All test user accounts are deleted or disabled.	
	All unnecessary modules are disabled.	Disabling unnecessary modules and services and keeping the system up-to-date
	All unnecessary pages are deleted.	
	File types that can be uploaded to the system are restricted. You can specify which extensions are allowed for uploaded files in general, including forms in Site Manager -> Settings -> System -> Files in the Security group.	
	UI personalization for specified roles is set correctly to prevent users from accessing unnecessary user interface. You can configure UI personalization in Site Manager -> Administration -> UI personalization.	UI personalization
	Permissions for specified actions in Kentico CMS modules are set correctly for all roles. You can configure permissions in Site Manager -> Administration -> Permissions.	Configuring permissions securely
	Users are allowed to use only strong and complex passwords. You can enable the Use password policy setting in Site Manager -> Settings -> Security & Membership -> Passwords.	Password strength policy and its enforcement
	The passwords are stored in a strong and secure format. Recommended option is SHA2 with salt. You can set password format in Site Manager -> Settings -> Security & Membership -> Passwords -> general group.	Password encryption in database
	The number of allowed invalid logon attempts is limited. You can set the limit in Site Manager -> Settings -> Security & Membership -> protection in the Invalid logon attempts group.	Invalid logon attempts
	You have consider if autocomplete function is needed. Autocomplete can be enabled in Site Manager -> Settings -> Security & Membership -> Protection -> General group.	Autocomplete deactivation
	Forms are secured with CAPTCHA (spam protection control).	Spam protection (CAPTCHA)
	Encrypted Internet connection (HTTPS) is configured properly.	Configuring SSL